CHAPTER 6
Verb Thinking

THE GREATEST OBSTACLE TO THE advancement of interactive storytelling is the difficulty of verb thinking. This is an unconventional and almost unnatural style of thinking that is nevertheless central to understanding interactivity.

Verb thinking can be appreciated only in the context of its yin-yang relationship with noun thinking.
Verb Thinking Versus Noun Thinking

What is the universe? Most people answer this question by describing a collection of objects existing in a space. The universe is so many stars, galaxies, planets, and so forth. Our corner of the universe is composed of so many rocks, trees, animals, plants, houses, cars, and so on. To completely describe the universe, you need to list all the objects within it. Practically, it’s an impossibly huge task, but conceptually, it’s simple enough.

But there’s another way to think of the universe: You can think of it in terms of the processes that shape it, the dynamic forces at work. Instead of thinking in terms of objects embedded in space, you could just as well think in terms of events embedded in time.

These two ways of looking at the universe compose the yin and yang of reality. They show up in every field of thought. In linguistics, you can see this yin and yang in the two most fundamental components of all languages: nouns and verbs. Nouns specify things, and verbs describe events. Nouns are about existence; verbs about action. Together they make it possible to talk about anything under the sun.

In other fields, the yin/yang shows up in economics as goods and services. Goods are objects, things that exist. Services are actions, desirable processes. Goods are (mostly) permanent, and services are transitory. In physics, you talk about particles and waves. Particles are things; waves are transitory processes.

Military theorists worry about assets and operations. Assets are the troops, weapons, and ammunition that can be used in a war; operations are the military actions carried out with these assets. Together, assets and operations define military science.

Programmers see the dichotomy even more clearly: They have memory for storing data, and the CPU for processing data. Data is the “noun” of computers. It’s what is chewed up, sorted out, and worked over. Processing is the “verb” of computers, what does the chewing, sorting, and working over.

Even a factory manager unconsciously thinks in these terms. The factory starts with parts, the nouns of the operation, and applies labor, the verb of the operation, to build its products.
A simple table of terms that touch on this dichotomy demonstrates just how fundamental it is:

<table>
<thead>
<tr>
<th>Noun</th>
<th>Verb</th>
</tr>
</thead>
<tbody>
<tr>
<td>goods</td>
<td>services</td>
</tr>
<tr>
<td>particle</td>
<td>wave</td>
</tr>
<tr>
<td>assets</td>
<td>operations</td>
</tr>
<tr>
<td>data</td>
<td>processing</td>
</tr>
<tr>
<td>number</td>
<td>algorithm</td>
</tr>
<tr>
<td>fact</td>
<td>idea</td>
</tr>
<tr>
<td>truth</td>
<td>logic</td>
</tr>
<tr>
<td>knowledge</td>
<td>understanding</td>
</tr>
<tr>
<td>parts</td>
<td>labor</td>
</tr>
<tr>
<td>character</td>
<td>plot</td>
</tr>
</tbody>
</table>

Because these two approaches of noun thinking and verb thinking constitute a dichotomy, we often think in terms of one side versus the other (nouns versus verbs, goods versus services, and so forth), but in fact the dichotomy is illusory; it's really a polarity. A big gray zone where the two sides blend together shows up in every one of the human activities listed previously:

- **Language:** Nouns can be verbified ("Trash that memo!") and verbs can be nounified ("That was a good play"). In the process, the word's definition can blur. Is "trash" intrinsically a noun or a verb? How about "play"?

- **Economics:** You purchase a hamburger at a fast-food restaurant. Are you purchasing goods (the burger itself) or services (the preparation of the burger)? When you hire a tax preparer, are you purchasing a good (the completed tax return) or a service (the efforts of the tax preparer)?

- **Physics:** The wave-particle duality drives everybody nuts. If you shoot electrons through a double slit, they'll diffract, just like a wave. Yet you can detect each electron individually, just like a particle. So are electrons waves or particles? It depends. Massive objects behave more like particles. Tiny objects behave more like waves—but you can never isolate a single aspect of
the behavior. You can only make one aspect, particlness or waveness, very small compared to the other aspect.

- **Military science:** A small number of fast ships aggressively patrolling a sea zone can accomplish just as much as a large number of sluggish ships hanging around port. And obviously, a large army with plenty of assets can still accomplish a great deal if handled properly.

- **Computers:** From the earliest days of computers, programmers have realized that any computation can be carried out with almost any combination of data and process. If you can throw more memory at the problem, you require less processing and the program runs faster. If you're short on memory, you can always rewrite the program to use more processing and less memory.

- **Factory:** The tradeoff between raw materials and labor consumption is well known. You can work "up the supply chain" by taking in more of the job yourself. Instead of purchasing finished printed circuit boards for your computer, you can purchase the parts and add employees to make the printed circuit boards. More labor means less parts.

Thus, the yin of nouns and the yang of verbs complement each other neatly; you can twist the dial one way to apply a "verbier" approach or the other way to get a "nounier" approach. Plenty of special circumstances, of course, cause you to lean more toward one side or the other. It's a lot easier to think of Jupiter as a particle than to think of it as a wave with a very tiny wavelength. In your factory, if labor is cheap, you want to use fewer finished parts and more labor, but if labor is expensive, you want to use more finished parts.

However, another factor at work heavily biases the noun/verb balance in everything we do: the natural bent of our minds. For some reason, it's easier to think in terms of nouns rather than verbs. This bias shows up in each of the aforementioned fields:

- **Language:** I leafed through my dictionary, picking pages at random and counting the number of nouns and verbs on each page. I counted a total of 101 nouns and 26 verbs, a four-to-one ratio. I repeated the experiment with a Latin dictionary and counted 57 nouns to 20 verbs, a mere three-to-one ratio. The English language seems to have a strong preference for nouns.
Contrast that with Hopi, for example, which is famous for its preference for verbs.

- **Economics**: Adam Smith's book *The Wealth of Nations* laid the foundations of modern economics, establishing all the basic principles: the relationship between price, demand, and supply; the utility of division of labor; the advantages of free trade; and so forth. Yet the concept of services as an economic entity was poorly developed in Smith's work. He was well aware of the role of labor in economic production, but only as a means of creating goods. Therefore, the father of modern economics never saw the yin/yang of goods and services; he thought almost exclusively in terms of goods. Indeed, the notion of services as an economic output didn't establish itself firmly in economic thought until the early twentieth century. The "goods" side of the dichotomy was figured out quickly; the "services" side took another century.

- **Physics**: Isaac Newton laid down the physics of particle motion in 1664; it took another 200 years before scientists worked out the physics of wave behavior in as much detail. Even for the brilliant people who created the science of physics, particles were much easier to understand than waves.

- **Military science**: From the beginning of military history, the concept of "assets" was recognized. The actions of generals in ancient times clearly demonstrate that they were quite cognizant of the importance of how many soldiers and weapons they had. Early generals, however, only dimly grasped the notion that the way those assets were applied was just as important as the assets themselves. Alexander the Great seemed to appreciate the idea, but his use of it seems spotty. Napoleon was the first general to demonstrate a consistent application of the idea that armies are only as useful as the extent to which they are maneuvered. It was the military theorists between the two World Wars (Liddell-Hart and Guderian) who first put down on paper the ideas behind the operational approach to military strategy, and it took the Blitzkrieg to wake up the rest of the world to the value of these ideas. Therefore, military thinkers took more than two millennia to recognize the importance of the yang of operations.

- **Factory**: The management of raw materials and parts is an advanced field now. Using computers and close communications with suppliers, factory managers can ensure that parts and raw materials arrive at the factory just before they are used, thereby dramatically improving the system's efficiency.
Labor management enjoys no corresponding system, however, so it remains a messy, uncomputerized task. Perhaps it's because labor is done by human beings, who are not so easily pushed around to fit the algorithms. But then, why couldn't algorithms be designed to take into account individual differences in human beings? The fact is, our understanding of the labor flow in a factory is far behind our understanding of the parts flow.

**Computers:** From the start, the relative importance of memory and processing were recognized, largely because memory was in such short supply that computer scientists were desperate to find any way to get the work done. Indeed, the history of the relationship of memory to processing has been almost the reverse of every other field: Hardware limitations forced early programmers to put more emphasis on processing, but as computer memories became larger, programmers put less emphasis on the processing side and more on the data side of the dichotomy. A second factor contributing to this process was the explosion of demand for programmers. In the '50s, '60s, and '70s, there wasn't much demand, so only the best and brightest became programmers. When the demand for programmers exploded in the '80s, however, we could no longer afford to limit the ranks of programmers to the cream of the crop: large numbers of un-brilliant programmers flooded the field, and these people weren't as comfortable working with algorithms as they were slicing bytes around.

The programming microcosm of games clearly shows this shift toward noun thinking. One of the early videogame systems, the Atari 2600, was equipped with a powerful processor: an 8-bit 6502 running at just under 1MHz. However, it initially allowed only 2KB of ROM and just 128 bytes of RAM. Games designed for this system were heavy on processing and light on data. Over the years, games have grown vastly in size. A typical game these days requires a CD-ROM with 650MB of storage. You can see the nature of the change most clearly by comparing the shift in hardware capabilities in videogame machines:

<table>
<thead>
<tr>
<th>Machine</th>
<th>Atari 2600</th>
<th>Microsoft Xbox</th>
<th>Ratio: Xbox to Atari</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM</td>
<td>128 bytes</td>
<td>84MB</td>
<td>512,000</td>
</tr>
<tr>
<td>ROM</td>
<td>2KB</td>
<td>8GB</td>
<td>4 million</td>
</tr>
<tr>
<td>Clock speed</td>
<td>1MHz</td>
<td>733MHz</td>
<td>733</td>
</tr>
<tr>
<td>Bus width</td>
<td>8 bits</td>
<td>64 bits</td>
<td>8</td>
</tr>
</tbody>
</table>
Note how data capacity has increased far more than processing capacity. The Xbox is millions of times better than the 2600 when it comes to data, but only hundreds of times better when it comes to processing. Games programming has shifted away from processing and toward data. Now, much of this change is caused by technological limitations; it’s easier to improve storage than processing. Nevertheless, this change also represents (to a lesser extent) what people seem to demand in their games as well as programmers’ proclivities. And a major question remains: If game machines are millions of times better, why aren’t the games themselves millions of times better?

These examples demonstrate just how frozen our minds are into noun thinking. We just can’t seem to “get” verb thinking, and when we do, it’s always late and second best.

In Chapter 3, “Interactivity,” one important point is that the degree or quality of the interactivity increases with the quantity and quality of the choices available to the user. To pursue that idea a little further, you can say that choice takes action as its direct object. You don’t choose between an éclair and cotton candy; you choose between eating an éclair and eating cotton candy. You choose between bowing your head and saying “Yes, sir” to your boss and pulling out a machine gun and blasting him to oblivion. You choose between verbs. Verbs lie at the heart of choice, and choice lies at the heart of interactivity. To put it bluntly:

**Lesson #15**

*Interactivity requires verb thinking.*

Lesson #15 is the core reason that software sucks. Good interactivity design requires clear verb thinking, yet most people just don’t “get” verb thinking. So we struggle forward with noun thinking, building software that’s noun-heavy and verb-light.

**Case in Point: Multimedia**

Does anybody remember the multimedia craze of the early 1990s? It all started when CD-ROM drives became standard equipment on personal computers, and software developers could take advantage of the storage capacity of these babies.
It might be hard to believe, but the average personal computer back then had only about 4MB of RAM and 100MB of hard disk storage. The floppy disk was the standard means of distributing software, and its capacity was only 1.4MB. By contrast, a CD-ROM boasted 650MB of storage. The transition from floppy disk to CD-ROM increased delivery capacity by a factor of more than 400! To make a similar-sized jump from CD-ROM to something bigger, that bigger medium would need to hold about 260GB—bigger than most hard disks these days.

Freed from the constraints of floppy disks, software developers began stuffing huge quantities of data onto CD-ROMs. That data took the form of graphics. Because most personal computers in those days had 640×480 8-bit displays, requiring only 307KB of data per screen, developers could stuff about 2,000 screen images onto a single CD-ROM. Overnight, the demand for artists to prepare all this artwork exploded.

Joy reigned throughout Software Development Land. The old constraints had been lifted; the sky was the limit. Artists who had previously been discouraged by computers because they required so much programming could now create lots of dumb software with great graphics, and that’s exactly what they did. The floodgates opened and out gushed a tidal wave of software called “multimedia.” In design terms, all multimedia software was pretty much the same: a big pile of images and sounds presided over by a tiny little program that did little more than shuffle bytes from the CD-ROM to the screen and speakers.

Amid the cheering masses at the grand parade stood one scowling curmudgeon: me. I rained all over their parade, dismissing CD-ROMs and multimedia as a waste of time. After all, I pointed out, the CD-ROM is a data technology, not a processing technology. It gives a huge boost to the noun side of the interactivity, but that’s useless: What matters is the verb side. Therefore, CD-ROMs and multimedia weren’t going to do much to advance the software revolution.

Nobody believed me; they thought I was crazy. And certainly the sales figures made me look like an idiot. People made millions and millions of dollars creating multimedia. The proof of the pudding is in the eating, and multimedia people were eating quite well.

I had seen it all before. Around 1983, a similar fad swept the world of arcade games: Dragon’s Lair. It was the first arcade game to use an optical disk, a precursor to the CD-ROM that was bigger, clumsier, and more expensive. The makers of this game stuffed actual animated video (from Don Bluth, no less!) onto
the optical disk with a simple branching system that permitted the internal computer to shunt different video snippets to the screen. The structure of the game was simple (see Figure 6.1).

![Game Structure Diagram](image)

**Figure 6.1:** Game structure of Dragon’s Lair.

The game was a huge hit because nobody had ever seen such glorious graphics in an arcade game. Naturally, every other arcade game maker rushed to deliver its own optical disk game. Within a year, arcades were full of optical disk games, which promptly flopped. Optical disk games were a fad, a flash in the pan, because they really weren’t fun to play. Sure, they looked great, but they didn’t do anything interesting. And play, as well as any form of interactivity, succeeds or fails on what the user does.

**Lesson #16**

*Crawford’s First Rule of Software Design: Ask “What does the user DO?”*

Getting back to the 1990s and multimedia, I saw that the whole thing was a fad, just like Dragon’s Lair. The multimedia craze would climb a little higher and last a bit longer, but it had no staying power. And that’s exactly what happened. The
craze lasted three, maybe five years, depending on how you measure it. By the late 1990s, multimedia was dead. The CD-ROM lived on. After all, it was a better distribution medium than the floppy disk. But people stopped selling software on the size of its multimedia content.

It is instructive that, in the games industry, multimedia was killed by 3D graphics. The distinction is lost on most observers, who figure that one graphics medium is pretty much the same as any other. In fact, there’s a huge difference between CD-ROM-based multimedia and software-based 3D graphics rendering: The former is data intensive, and the latter is process intensive. Multimedia merely shovels bytes from the CD-ROM onto the screen; multimedia designers sweat data capacity. But 3D graphics software calculates its imagery using the CPU; its designers sweat machine cycles. Because 3D graphics software is processed rather than shoveled, it’s far more responsive to the user. The interaction is tighter and more intimate because it’s process intensive. Remember, data intensity is noun based; process intensity is verb based.

**Getting Started with Verb Thinking**

Okay, so I’ve convinced you. You have seen the light, and you’re a True Believer in the gospel of verb thinking. Now how can you learn verb thinking?

First, you lay the foundations by shifting your outlook on the world. Here’s that list of dichotomous words again:

<table>
<thead>
<tr>
<th>Noun</th>
<th>Verb</th>
</tr>
</thead>
<tbody>
<tr>
<td>goods</td>
<td>services</td>
</tr>
<tr>
<td>particle</td>
<td>wave</td>
</tr>
<tr>
<td>assets</td>
<td>operations</td>
</tr>
<tr>
<td>data</td>
<td>processing</td>
</tr>
<tr>
<td>number</td>
<td>algorithm</td>
</tr>
<tr>
<td>fact</td>
<td>idea</td>
</tr>
<tr>
<td>truth</td>
<td>logic</td>
</tr>
<tr>
<td>knowledge</td>
<td>understanding</td>
</tr>
<tr>
<td>parts</td>
<td>labor</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>character</td>
<td>plot</td>
</tr>
<tr>
<td>story</td>
<td>drama</td>
</tr>
</tbody>
</table>

Every time you think about any of these topics, contrast the left-side word with the right-side word. Are you worrying about the facts of an issue when you could be thinking about the ideas behind it? Go down the street looking at shops and stores. Are they selling goods or services? Can you imagine their output solely in terms of the services that went into the merchandise?

Above all, try to think about things in terms of what things do, not what they are. A window is not glass; it's something that blocks air movement while permitting light to pass freely. A car is not an engine, a body, seats, and so forth. It's something that moves; everything else is subsidiary. A computer is not a box with a whirring fan; it's a processing machine. A pill is not a bundle of exotic chemicals; it's something that alters the biochemistry of your living processes.

Recall my reference in Chapter 1, “Story,” to the image taken from the end of the movie The Matrix; I'm going to use it again. The protagonist, Neo, finally realizes his potential to see behind the artificial world of the matrix, to see that it's really just numbers and algorithms. The walls dissolve into numbers, and his antagonists are stacks of numbers; everything is just numbers that he can control at will. This imagery provides the metaphor for how you must see the world. Those aren't walls in the hallway; they're zillions of atoms held together with atomic forces, transmitting weight from roof to floor, interacting with photons of light to bounce those photons in different directions. Things don't just sit there—they happen, and if you can come to understand how and why they happen, you can understand the universe at a deeper level, like Neo. And like Neo, you can freeze bullets in mid-air. That's the power of this kind of thinking.

Everything is connected in intricate webworks of cause and effect, and your goal in life is to understand as much of that webwork as you can. To do so, you must concentrate on causal relationships, on the precise nature of causality. You need a language that allows you to express with clarity and precision the exact nature of each causal relationship you discover. I have good news and bad news for you. The good news is that this language has already been developed; the bad news is that it's mathematics.
I can hear the hissing sound of your deflating balloon. The very thought of using mathematics probably repels and terrifies you, but in my experience, a willingness to embrace mathematics is the single most significant factor in success or failure in software design. Many people refuse to tackle mathematics, like debutantes unwilling to seize a greased pig, and this aversion is natural. Our brains are wired to think in associations and patterns, and mathematical thinking is sequential, step-by-step in style. It requires you to twist your mind around in knots to make it operate in a style it just isn't built for. You have every right to balk at my suggestion that you subject yourself to this kind of mental torture.

But what, I ask you, is wrong with the idea that you have to work to accomplish wonderful things? Great artists have never shied away from unpleasant tasks in pursuit of their artistic goals. The brutal truth is that you must embrace the slobbering monster of mathematics if you are to succeed in interactive storytelling. Here's the true test of your passion: Will you get in bed with this creature to pursue your vision?

And don't kid yourself with the comforting self-deception that there's just got to be another way to do interactive storytelling without mathematics. There's no getting around this one. Lord knows, I've tried. In designing the Erasmatron, I have struggled to cushion the spikiness of mathematics in layers of soft verbal down. I have tried to concoct other forms of expression that skirt mathematics completely. Despite all my creative powers, I have failed to concoct any non-mathematical scheme that adequately empowers storybuilders.

So gather your courage and face the monster; I'll keep him on a chain.

**Causal Relationships**

We normally think of causality in merely binary terms: Socrates is either mortal or not mortal. If X, then Y, and X is either true or false, as is Y. It's more useful to think of causality in mathematical terms, however: Apples cost $0.60 per pound, so if I buy 2 pounds of apples, I pay $1.20, and if I buy 4 pounds of apples, I pay $2.40. There's a direct mathematical relationship between the cause and the effect, expressed in a simple mathematical formula:

\[
\text{Price} = 0.60 \times \text{Weight of apples}
\]

Take this idea a half-step further: Suppose somebody discovers that a man's weight in pounds is equal to four times his waist measurement in inches. This
isn't actually true, but for purposes of argument, suppose that it is true. Then you could make the following mathematical statement:

All men's weight is equal to four times their waist measurement.

You can set this up as a kind of syllogism:

All men are mortal. All men’s weight is equal to four times their waist measurement.

Socrates is a man. Socrates has a 48-inch waist.

Therefore, Socrates is mortal. Therefore, Socrates weighs 192 pounds.

The difference between the two syllogisms is that the left-side syllogism gives mere binary results (yes or no to the question “Is Socrates mortal?”), but the right-side syllogism can have all sorts of different results, depending on the man’s waist size. The left side divides Truth into black and white, yes or no, but the right side permits shades of gray. The left side can address only the most simple-minded questions, but the right side can address far more: anything that can be quantified.

Having established the basic principle that mathematical formulas are simply a kind of syllogism, you can then make those formulas more complex. For example, a more accurate formula for a man's weight might be something like this:

All men's weight is equal to 3 times their waist measurement plus their height.

Or to put it in the spare terminology of mathematics:

\[ \text{Weight} = 3 \times \text{Waist} + \text{Height} \]

This equation is a little more accurate. It's still not really true, but only in the absolute sense of getting exactly the right number for every man. This equation might get within 20 pounds of correctly predicting the weight of 70 percent of all men. That's pretty good. You could probably make it better if you made the formula even more complex:

\[ \text{Weight} = 2 \times \text{Waist}^{-0.5} + 0.75 \times \text{Height} - 12.1 \times \text{ShoeSize} \]
Perhaps this equation would get within 10 pounds of correctly predicting the weight of 80 percent of all men. The perfect equation—the one that correctly predicts with zero error the weight of all men—is impossible to figure out. Perhaps it could be found with enough effort, but it would probably involve thousands of variables put together in the most intricate fashion and be dozens of pages long. For all practical purposes, perfection is impossible.

Perfection is seldom necessary, however. A chair designer needs to know the approximate relationship between the size and the weight of most people, but there's no need for exact numbers; getting within 20 pounds is good enough. An interactive storyteller need not achieve perfection of mathematical description; getting close enough for dramatic fidelity is all that's required.

**Modeling as Metaphor**

One of the oldest figurative metaphors is Homer's "rosy-fingered dawn." It's a powerful metaphor because it compares the rays of predawn sunlight and the fingers of the hand, suggesting an animation of a natural process. Poets are in the business of creating metaphors. So are most other artists, in at least some sense. The same thing applies to the mathematical modeling required for interactive storytelling. Indeed, this mathematical modeling is the essence of its artistic content—NOT the images, NOT the sounds, NOT the dialogue, NOT even the plot development. Sure, you can have artistic images in an interactive storytelling product, and you can stuff in a symphony or two, mix in a dash of poetry, and so forth, but that's not interactive storytelling. You could create a movie consisting of little more than pictures of great paintings, but would it be a good movie? You could create a movie of a great poet reading poems, but would it be a good movie? You could create a movie showing an orchestra playing a Beethoven symphony, but would it be a good movie? Taking good art in one medium and simply transferring it to another medium doesn't make the end result good art. Indeed, a true artist strives to express content most parallel to the grain of the medium.

This point is immensely important, but sadly, one that many people find difficult to accept. Part of their resistance arises from the passionate attachment that any artist feels for his or her art and its medium. Moviomakers love the cinema; poets adore poetry; writers worship literature. My claim that these magnificent art forms have no primary place in interactive storytelling sounds like a principal
telling a parent that her child isn’t good enough to attend his school. The same outraged rejection is the inevitable reaction to my claim.

But, hold, friend. I write these words not to belittle your child but to help you find the ideal place for your efforts. In past times, sons were forced to take up their father’s trade regardless of their own proclivities. This interactive storytelling medium might not be the best place for your talents or inclinations; if so, don’t fight it. Follow your muse.

And don’t try to prostitute your muse into a role for which she’s ill suited. In the earliest years of cinema, some gifted thespians attempted to force Melonene and Thalia into this new and alien medium. They simply set up the camera in the middle of the audience seating and acted out a play on the stage. These travesties earned the approbation of the better people, but nobody actually enjoyed them; they were inferior versions of the real thing. They were seen as rather like canned fruit: a feeble substitute for the real thing, but adequate for distribution to remote areas lacking access to the genuine article. They were also financial flops, buried by the success of vulgar, sensationalist collections of what were nothing more than carnival sideshows. As a consequence, thespians came to disdain the cinema as a simian medium. It took D.W. Griffith, a failed actor with no emotional attachments to the theater, to approach the cinema without preconceptions. No worshipper of Melonene and Thalia, he conjured up his own muse and succeeded where so many talented thespians had failed.

Therefore, friend, if you have already set your path on the well-trod way up the mountain peaks of other media. I wish you well but urge you not to drag the medium of interactive storytelling along with you. That is another mountain peak, along a very different path. Worse, the path to that mountain peak is unexplored and unmarked; you’ll need a machete as much as good hiking boots. Progress up this path will be slower than what you can accomplish in other media. You’ll not get there faster by taking the easier path.

If you can jettison your emotional attachments to other media (including games!) and approach interactive storytelling with the naiveté of a child, then you can accomplish something. If you try to force it to fit into your preconceptions, you will fail.
These things said, I shall now explain how to create metaphors using mathematical modeling.

I'll start with a simple example: that overwrought moment in *The Return of the Jedi* when Darth Vader, watching the evil Emperor torture his son, Luke Skywalker, suddenly changes his mind, sides with Luke, and tosses the Emperor down a convenient deep shaft. As the Emperor blasts Luke with his nasty Evil-Electricity bolts, Darth looks at Luke, then at the Emperor, then at Luke, then at the Emperor—this goes on for what seems an eternity. Obviously, he is making a decision. In the film, the decision (and the difficulty of making it) is communicated through the long back-and-forth sequence. But in interactive storytelling, you must determine how this decision is made. What forces are at work in Darth’s mind?

Obviously, this decision represents a battle between two competing forces: his loyalty to the Emperor and his natural love for his son. But there are secondary forces as well: self-interest, which adds to his loyalty to the Emperor, and idealism, which detracts from it. There’s also the empathy that any person feels for those who suffer. These various factors do battle in Darth Vader’s mind to decide whether Darth should turn against the Emperor.

You can translate the ideas in the preceding paragraph into mathematical terms, line by line, sentence by sentence:

*this decision represents a battle between two competing forces*

*IF (competing force 1 > competing force 2)*

This formula is the mathematical representation of a simple yes-or-no decision. It doesn’t specify what happens in each of the two conditions; it merely establishes the decision’s basic form. If the strength of competing force 1 is greater than the strength of competing force 2, then the result is “yes.” We’ll figure out what “yes” means later.

*his loyalty to the Emperor*

*Competing force 1 = Loyalty[Darth, Emperor]*

This formula simply represents the idea of loyalty with a mathematical variable that’s read as “Loyalty of Darth for the Emperor.” You specify the two Actors,
Darth and Emperor, because you could just as well worry about loyalty of Darth for Luke or any other pairing of Actors.

Here’s the next term from my descriptive paragraph:

his natural love for his son

competing force 2 = Love[Darth, Luke]

But there are secondary forces as well

This means you have to adjust the earlier formulas with additional factors, such as:

self-interest, which adds to his loyalty to the Emperor

competing force 1 = Loyalty[Darth, Emperor] + SelfInterest[Darth]

This new term is read as “Self-Interest of Darth” because it refers to Darth’s own self-interest. Again, you might want to use Luke’s Self-Interest somewhere else, so you might as well maintain the variable separately for each Actor.

and idealism, which detracts from it

competing force 1 = Loyalty[Darth, Emperor] + SelfInterest[Darth] - \text{Idealism}[Darth]

This formula subtracts (“detracts”) Darth’s Idealism from his Loyalty to the Emperor. Onward:

There’s also the empathy that any person feels for those who suffer

competing force 2 = Love[Darth, Luke] + Empathy[Darth]

Next:

These various factors do battle in Darth Vader’s mind to decide whether Darth should turn against the Emperor.

if (competing force 1 > competing force 2)
then WatchLukeDie
else TurnAgainstTheEmperor
Here WatchLukeDie is the verb for doing nothing, and TurnAgainstTheEmperor is the verb that leads to Darth shot-putting the Emperor.

Now assemble the whole thing in one fragment of pseudocode (pseudocode is a programmer’s term for code that’s not written in any particular programming language, but is so plain and simple in structure that anyone who knows how to program could understand it):

```
competing force 1 = Loyalty[Darth, Emperor] + SelfInterest[Darth] - Idealism[Darth]

competing force 2 = Love[Darth, Luke] + Empathy[Darth]

if (competing force 1 > competing force 2)
    then WatchLukeDie
else TurnAgainstTheEmperor
```

This little snippet might look like a gross simplification of human behavior, and indeed it is. It’s not about human behavior, but about drama. It is an artistic simplification. One of the most important realizations in mathematical modeling is that there’s no such thing as the “correct” model, only models that emphasize different facets of reality with more or less resolution and accuracy. The trick is to let go of a narrow-minded insistence on “correctness”; embrace the realization that the more you learn, the less you know; and regard reality with the playful attitude that every thought in your mind is merely a puppet-like rendition of the infinite complexity of reality. Humbly accept your work as sketches, not photographs. Don’t bemoan your failure to achieve photographic perfection in your paintings—revel in the cleanliness of line, the clarity of image you can produce.

Now go over the artistic content of the snippet of pseudocode. It says that Darth has just two choices: WatchLukeDie or TurnAgainstTheEmperor. Yes, there could be a number of other choices: ProtestToEmperor, BegForLuke’sLife, LaughAtLuke’sSuffering, TurnAwayInSorrow, JumpBetweenLukeAndEmperor, or even PlayRummyWithAStrongTrooper. Under other dramatic circumstances, these options might be appropriate for Darth, but this moment is the climax of the whole movie (indeed, the climax of the entire six-part series). You don’t want to spin this out into interesting sideways curlicues; it’s time for Darth to put up or shut up, so you give him a stark choice between two options. Remember, you could
give him more choices if, as a storybuilder, you wanted to play the storyworld out further; it's a matter of artistic choice, not an absolute requirement.

Having established the basic nature of the choice to be made, now turn to the criteria by which that choice is made. I have established that the force driving Darth to side with the Emperor is equal to his Loyalty to the Emperor plus his SelfInterest minus his Idealism. This is the simplest possible mathematical combination of those terms. However, I might want to place more emphasis on Darth's SelfInterest, believing that it will play a larger role in the decision than his Loyalty to the Emperor. I can do this with a simple change:

\[
\text{competing force 1} = \text{Loyalty}[\text{Darth, Emperor}] + (2 \times \text{SelfInterest}[\text{Darth}]) - \text{Idealism}[\text{Darth}]
\]

All I've done here is doubled the influence of SelfInterest in the final outcome. I might also want to reduce the influence of Empathy in Darth's decision:

\[
\text{competing force 2} = \text{Love}[\text{Darth, Luke}] + (\text{Empathy}[\text{Darth}] / 2)
\]

Or I could make the influence of Empathy even smaller:

\[
\text{competing force 2} = \text{Love}[\text{Darth, Luke}] + (\text{Empathy}[\text{Darth}] / 4)
\]

My point with these variations is to demonstrate how simple it is to express artistic ideas with arithmetic. There's nothing absolute about these formulas; I don't insist that dividing by 4 in the last formula is the only, or even the best, way to express it. You could have divided by 3 or 5 and still gotten decent results. The important idea is that I, as an artist, decided that Darth's Empathy was a less important factor in his decision than his Love for Luke, so I scaled down the Empathy factor a notch or two.

To summarize, mathematics is just as valid a medium of artistic expression as oil and canvas, stage and actor, or pen and paper. The artist uses the medium to create metaphorical descriptions of the human condition. There's one big difference: Mathematics, more than any other medium, addresses the choices that characters make. That's because mathematics is about processes, not data. Other media can show the results of those choices, the attitudes of the characters, the anguish on their faces, but ultimately, the choice must be treated as a black box, like Darth Vader looking one way and then the other. Mathematics can delve
right down into the fundamental basis of the choice; no other medium can do that.

Aside: Quantifying Humanity

The notion that human traits can be represented by a mathematical variable raises my hackles. Human beings are infinitely complex creatures; reducing a person to a set of numbers is dehumanizing. The function of art is not to dehumanize people, but to explore and glorify the wonder of our existence.

Here's my counterargument: Remember, this is drama, not reality. Yes, real human beings are infinitely complex, but much of that complexity is stripped away in drama. Imagine the nastiest real person you've ever known; can that nastiness hold a candle to the Emperor's nastiness? Have you ever known any real person remotely similar to Darth Vader? Anybody as dashing as Han Solo? The characters in movies are not real, complex persons; they are simplifications. A movie that's 100 minutes long can't delve into the infinite complexities of the human condition, so it discards most of that complexity to show a few glimpses of it in shining clarity. Thus, the mere fact that mathematical representation simplifies isn't a significant argument against using mathematical expressions.

There's still the argument that quantification is intrinsically demeaning.

Is not loyalty a trait you can have more or less of? Are there not degrees of loyalty? If we can agree that loyalty does exist in varying degrees, what's wrong with assigning numbers to those varying degrees? If we can say that Darth Vader's loyalty to the Emperor is greater than Luke's loyalty to the Emperor, is there any fundamental shift in thinking to assign numbers to reflect that?

But loyalty is a multidimensional concept, one that can't be measured by a single number. There are differing kinds of loyalty: the loyalty to superiors and subordinates, the loyalty for family members, and the loyalty for friends. These different kinds of loyalty can't be represented by a single number.
Are these differences significant to the storyworld you intend to create? Will actors in that storyworld behave differently because they have these different dimensions of loyalty? If so, you must break down the single variable loyalty into four separate flavors of loyalty, and quantify them separately. But if these differences aren’t significant to your storyworld’s artistic content, you can dispense with them and use the single factor loyalty.

There’s a huge difference between talking about “greater” or “lesser” loyalty and “57” loyalty or “23” loyalty. The verbal description leaves plenty of room for the approximate nature of appreciation of human qualities, and the quantitative description creates a wholly inappropriate impression of accuracy.

Consider the following list:

- Greatest loyalty
- Greater loyalty
- Lesser loyalty
- Least loyalty
- No loyalty

Clearly, this list constitutes a verbal description of varying degrees of loyalty. Hence, you could easily assign numbers to it:

<table>
<thead>
<tr>
<th>Loyalty</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greatest loyalty</td>
<td>4</td>
</tr>
<tr>
<td>Greater loyalty</td>
<td>3</td>
</tr>
<tr>
<td>Lesser loyalty</td>
<td>2</td>
</tr>
<tr>
<td>Least loyalty</td>
<td>1</td>
</tr>
<tr>
<td>No loyalty</td>
<td>0</td>
</tr>
</tbody>
</table>

You could readily extend this list by adding more terms:

<table>
<thead>
<tr>
<th>Loyalty</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greatest loyalty</td>
<td>7</td>
</tr>
<tr>
<td>Strong loyalty</td>
<td>6</td>
</tr>
<tr>
<td>Greater loyalty</td>
<td>5</td>
</tr>
<tr>
<td>Moderate loyalty</td>
<td>4</td>
</tr>
</tbody>
</table>
Lesser loyalty 3
Insignificant loyalty 2
Least loyalty 1
No loyalty 0

You could extend this list and achieve better precision if you didn’t stumble over imprecise—and, therefore, overlapping—terms such as “insignificant,” “strong,” and “great.” But these are limitations of the vocabulary; how do you know they’re intrinsic to the idea of loyalty? I agree that specifying somebody’s loyalty as 2,364,198.7349 is ridiculous; you couldn’t possibly insist that 2,364,198.7349 is correct and 2,364,198.7348 is wrong. Perhaps specifying loyalty with an integer scale of 0 to 100 is too ambitious; perhaps you should specify it on an integer scale of 0 to 10. The principle of quantification is not at issue—only the degree of precision is debatable.

Quantification is not dehumanizing, nor does it trivialize the complexity of the human condition. Finding the right value for a variable is analogous to finding the right word for a sentence. It’s just a matter of zeroing in on the precise content you want. The only difference is that the process is numerical rather than verbal. Quantification uses a different vocabulary, but it says the same things. The benefit is that this vocabulary can be used with a computer.

Despite all these arguments, some readers will still choke at the notion of quantifying human qualities. If all my reasoning fails to move you, is it perhaps because of a profound emotional aversion to all things mathematical? If so, I ask you to confront your feelings and determine whether you can set them aside in pursuit of the grand and glorious goal of realizing interactive storytelling. If you cannot, then this field of endeavor is not for you; you will always be unhappy struggling without the proper tools. Go, follow your muse elsewhere.

The Artist’s Mathematical Palette

Now explore the palette of mathematical colors available to the artist. What do you have to work with?

The four fundamental arithmetic operators (addition, subtraction, multiplication, and division) provide the starting point. The easiest way to understand
them is as extensions of logical operations you're already acquainted with. Start with addition. When you add two variables together, the sum will be large if the first number is large or the second number is large. Another way to think of it is to imagine the sum as a criterion for action. It sits inside a formula like this:

\[
\text{Heroism} = \text{Courage} + \text{Selflessness}
\]

\[
\text{IF (Heroism} > 50) \text{ THEN}
\]

\[
\text{DoTheRightThing}
\]

\[
\text{ELSE}
\]

\[
\text{SitThereSnivelng}
\]

In this case, the Actor in question will pass the test if EITHER his Courage OR his Selflessness is high. High Courage can compensate for low Selflessness, and vice versa.

Subtraction is really just addition backward; many situations that require subtraction could be handled by inverting the meaning on one term and using addition, as in this example:

\[
\text{Heroism} = \text{Courage} - \text{Selfishness}
\]

\[
\text{IF (Heroism} > 50) \text{ THEN}
\]

\[
\text{DoTheRightThing}
\]

\[
\text{ELSE}
\]

\[
\text{SitThereSnivelng}
\]

If you have set up a personality model to record Selfishness instead of Selflessness, you would use subtraction here. If you think about it, almost any kind of variable can be presented in a left-handed version and a right-handed version. Here are some examples:

<table>
<thead>
<tr>
<th>Selflessness</th>
<th>Selfishness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Courage</td>
<td>Cowardliness</td>
</tr>
<tr>
<td>Magnanimity</td>
<td>Pettiness</td>
</tr>
<tr>
<td>Empathy</td>
<td>Hard-heartedness</td>
</tr>
</tbody>
</table>
And so on. In some situations, you need to use subtraction instead of addition, but from a larger viewpoint, the two variables are just mirror images of each other.

The use of multiplication is just a bit more difficult to understand. Multiplication operates like an extension of the logical AND operator; you multiply two numbers together when both the first number AND the second number are crucial to the outcome. For example, suppose an actor has just been wronged by another actor and is contemplating taking revenge. Suppose further that the two relevant variables in this case are Anger and Malevolence. The actor’s inclination to take revenge is the product, not the sum, of his Anger and Malevolence because they don’t compensate for each other. All the anger in the world doesn’t cause a fundamentally benevolent person to seek revenge, and all the malevolence in the world doesn’t cause a happy person to seek revenge. That’s why multiplication, not addition, is called for in this case.

As with subtraction, division is just multiplication backward. In cases in which you would multiply by Selflessness, you would divide by Selfishness. However, division can blow up in your face. This happens when you divide by a value that turns out to be zero. This situation is bad; no computer in the world can handle division by zero, so your algorithm is bound to fail. You need to refrain from using division unless you are absolutely certain the denominator can never be zero. Sometimes you can guarantee this by simply adding a small constant to the denominator, like so:

\[
\text{Courage} / (\text{Selflessness} + 5)
\]

You might ask why I chose the value 5. That brings me to a whole new subject: weighting factors.
**Weighting Factors**

Rare is the case when the values of variables can be used directly; most of the time you must adjust them so that one variable plays a more important role. This adjustment requires the use of *weighting factors*.

**Additive Weighting Factors**

The simplest kind of weighting factor is the additive weighting factor. For example, suppose that a female Actor must choose between two men, one of whom she already has a relationship with. Her inclination to leave Boyfriend #1 for Boyfriend #2 would be based on the difference in the degree to which she loves each:

\[
\text{Inclination}[\text{BreakUp}] = \text{Love[Girl, Boyfriend#2]} - \text{Love[Girl, Boyfriend#1]}
\]

For example, if the girl feels 46 Love for Boyfriend#1 and 52 Love for Boyfriend#2, the formula becomes:

\[
\text{Inclination}[\text{BreakUp}] = 52 - 46
\]

This formula yields a +6 inclination to break up. Tough luck, Boyfriend#1!

But this formula has a flaw: We all have a natural resistance to change and tend to stick with what we already have. So the equation should be altered by making it a little harder to break up. You could accomplish this with the following change:

\[
\text{Inclination}[\text{BreakUp}] = \text{Love[Girl, Boyfriend#2]} - \text{Love[Girl, Boyfriend#1]} - 25
\]

So the new formula becomes:

\[
\text{Inclination}[\text{BreakUp}] = 52 - 46 - 25
\]

This yields a -19 inclination to break up. Lucky for Boyfriend#1!

Subtracting 25 makes it a little harder for her to break up with Boyfriend #1. Now, if you want to model the decision to break up more precisely, you could throw in factors for her Loyalty, but if you want to keep it simple, this formula works.
Multiplicative Weighting Factors

Suppose an Actor must decide whether to respond angrily to an insult (a Retort), and you have already decided on the two factors that will control this decision: the affection the Actor feels for the Insulter, and the anger the Actor feels as a result of the insult. Therefore, the inclination equation looks like this:

\[ \text{Inclination}[\text{Retort}] = \text{Anger}[\text{Actor}] - \text{Affection}[\text{Actor}, \text{Insulter}] \]

If the Actor’s Anger is 18 and his Affection for the Insulter is 28, the formula gives this result:

\[ \text{Inclination}[\text{Retort}] = 18 - 28 \]

This yields -10; in other words, the Actor is not going to issue a retort.

But suppose you decide this formula isn’t quite right: the Affection component isn’t as important as the Anger component. You want to ensure that the Anger component plays the primary role in the decision, and the Affection component plays only a secondary role. To do this, insert a multiplicative weighting factor, like so:

\[ \text{Inclination}[\text{Retort}] = 2 \times \text{Anger}[\text{Actor}] - \text{Affection}[\text{Actor}, \text{Insulter}] \]

Here’s how the numbers work out now:

\[ \text{Inclination}[\text{Retort}] = 36 - 28 \]

This formula gives you an answer of +8, so the actor using this formula will issue a retort.

This weighting factor doubles the weight that Anger plays in the decision. Of course, you could accomplish almost exactly the same thing by halving the weight of the Affection component, like so:

\[ \text{Inclination}[\text{Retort}] = \frac{\text{Anger}[\text{Actor}] - \text{Affection}[\text{Actor}, \text{Insulter}]}{2} \]

See how the numbers work out to the same result:

\[ \text{Inclination}[\text{Retort}] = 18 - 14 \]

The result is +4, still positive, so the Actor still issues a retort.
You could also make the Anger component even stronger by using a larger multiplicative weighting factor, like so:

\[ \text{Inclination[Retort]} = 5 \times \text{Anger[Actor]} - \text{Affection[Actor, Insulter]} \]

\[ \text{Inclination[Retort]} = 90 - 28 \]

This means the Actor would retort even if Affection were much greater.

The general rule is that if two factors are being added or subtracted, and one deserves more weight than the other, multiply the more significant factor by a number.

**Exponential Weighting Factors**

What if two factors are being multiplied or divided? How do you weight one in relation to the other? Using a multiplicative weighting factor won't work because it multiplies both factors equally. For example, suppose that, for some crazy reason, you multiplied the two factors in the previous example. Here's what a multiplicative weighting would yield:

\[ \text{Inclination[Retort]} = 5 \times \text{Anger[Actor]} \times \text{Affection[Actor, Insulter]} \]

Plugging in the numbers, you get

\[ \text{Inclination[Retort]} = 5 \times 18 \times 28 \]

The result equals +2520.

Suppose the situation is reversed, however. Suppose the Actor's Affection value isn't 28 but 18, and the Actor's Anger is 28. Then the formula would look like this:

\[ \text{Inclination[Retort]} = 5 \times 28 \times 18 \]

The result equals +2520, exactly the same as the previous example. Multiplying by 5 doesn't do anything to alter the weights of either factor; it just makes the end result five times larger. Therefore, this formula does nothing to distinguish the two factors; you need something else. That something else is an exponentiation, like so:

\[ \text{Inclination[Retort]} = (\text{Anger[Actor]}^5) \times \text{Affection[Actor, Insulter]} \]
Here's the formula with the numbers plugged in:

\[ \text{Inclination[Retort]} = 18^2 \times 28 \]

The result is 9072. This time, however, if you reverse the values of the two variables (making Anger 28 and Affection 18), the formula gives this result:

\[ \text{Inclination[Retort]} = 28^2 \times 18 \]

This formula yields 14,112—not the same as in the previous example. By squaring the Anger, you have doubled its overall weight in the formula. As with multiplicative factors, you can achieve the same effect by applying its opposite to the other factor:

\[ \text{Inclination[Retort]} = \text{Anger[Actor]} \times \sqrt[3]{\text{Affection[Actor, Insulter]}} \]

Now run through the same reverse-the-variables test. The first case (18 Anger and 28 Affection) looks like this:

\[ \text{Inclination[Retort]} = 18 \times \sqrt[2]{28} \]

The result is 95.25.

The second case (28 Anger and 18 Affection) looks like this:

\[ \text{Inclination[Retort]} = 28 \times \sqrt[2]{18} \]

The result is 118.79.

Here's what happened: When Affection went down and Anger went up, the answer went up, not down; it followed the Anger, not the Affection. In other words, taking the square root deemphasizes a variable.

You can achieve more or less weighting by using larger or smaller exponents. However, be careful here: Exponentiation can yield astoundingly large numbers!

**How to Set Values**

How do you decide how big a weighting factor should be? There's no simple answer to this question; it's fundamentally an artistic decision. How prominently should a painter highlight the nose in a portrait? How many lines of dialogue should a playwright give a character? How close to the camera should a character stand? There are no mathematical answers here; you simply make a first guess.
and see whether the formula behaves the way you want it to. Again, you don't have to find the perfect formula, just one that's close enough to satisfy your own artistic goals.

There's one major difference between other artists and storybuilders: Other artists can experiment and immediately see the results of their efforts. A film director can position actors and look through the camera to see the frame that results; if it doesn't look right, the director can reposition them. A painter can fiddle with the nose, step back, and evaluate the result; if it doesn't look quite right, the painter can fiddle some more. A storybuilder working on an inclination equation, however, can't step back and look at the formula and determine whether it looks right; the only way to try it out is to run a rehearsal and see how the decision comes out.

The difficulty of fine-tuning inclination formulas is another example of the degree of abstraction required to build storyworlds. It simply doesn't do to write down a formula and leave it at that; the storybuilder must instead think in terms of huge ensembles of rehearsals. If you tell the computer to run through a storyworld a thousand times, what percentage of the time will an Actor reaching this decision choose the Retort option, for example? It's easy if the Actors almost always go one way or the other; clearly, the inclination equation is out of balance and requires adjustment. But what if 25% of the Actors facing this situation choose the Retort option? This could be a good balance if all the Actors choosing the Retort option are fairly hot-headed people and all the Actors rejecting the Retort option are more level-headed. There's no way to know from the single statistic, however. Therefore, the single statistic emerging from multiple rehearsals gives a rough gauge of the inclination equation's performance, but for fine-tuning a storyworld, more detail is required.

You can obtain this detail in several ways. One way is using a statistical analyzer that permits detailed statistical analysis of a database of rehearsals. This technique permits the storybuilder to examine the circumstances under which each decision was made. For example, a storybuilder working on the problem described previously could examine all cases in which the inclination equation was used, the identity of each Actor who used it, and what that Actor's Anger and Affection values were at the moment the decision was made. This detail would give the storybuilder the information needed to perform finer adjustments in the inclination equation.
There's another way to adjust inclination equations, much more tedious but perhaps workable. This technique requires the storytelling engine to run through rehearsals in which the storybuilder can intervene after every decision and provide some correction. In this way, the storybuilder experiences the storyworld directly, and every time an Actor makes a decision, the storybuilder can look at the list of options, check the inclination values the actor produced, and then indicate which inclination values are too low and which ones are too high. The engine would then adjust the weighting factors correspondingly.

The difficulty with this technique is that thousands of rehearsals would be required to get the weighting factors balanced properly. Moreover, because this process would extend over many months, decisions the storybuilder makes late in the process might contradict decisions made early in the process.

I will discuss some of these issues in more detail in Chapter 17, "Development Environments."

Wrapping Up

- The two most fundamental components of all languages are nouns and verbs. Nouns are about existence; verbs are about action.

- Choice lies at the heart of interactivity. You choose between verbs, not nouns. Therefore, good interactivity design requires verb thinking (thinking about things in terms of what they do, not what they are).

- First rule of software design: Ask "What does the user DO?"

- Mathematical formulas are a kind of extended syllogism. The mathematical modeling used in interactive storytelling is the essence of its artistic content.

- Quantification does not trivialize the complexity of the human condition. Computers use a different vocabulary than people do (numerical rather than verbal).

- Mathematics, more than any other medium, addresses the choices characters make because it delves right down into the process of the choice, not its results.
"IF YOU CAN'T SAY IT, you don't know it." That's what one of my English teachers used to say, and over the years I have come to realize how profoundly right he was. It's not just that transcribing an idea from thought into language is the only proof of thought; the thought and the language are deeply intertwined.

Linguists have tussled over this idea for 75 years; it's formally known as the Sapir-Whorf hypothesis. The strong version of this hypothesis declares that language determines the nature of thought.
requires something similar. Fortunately, you don't have to create an entire language as expressively powerful as, say, English or Chinese; you can get away with much smaller sublanguages.

Lesson #20

Interactive storytelling requires a sublanguage that both computers and humans can use.

Vocabulary

To be computable, the vocabulary of a dramatic sublanguage must be small—smaller than the average person's working vocabulary (about 5,000 words). Fortunately, a vocabulary of only a few hundred words should be adequate. Even more fortunately, there are numerous lists of the most common words in a variety of languages. Here are the 100 most common English words:

1. the
2. be
3. of
4. and
5. a
6. to
7. in
8. he
9. have
10. it
11. that
12. for
13. they
14. I
15. with
16. as
17. not
18. on
19. she
20. at
21. by
22. this
23. we
24. you
25. do
26. but
27. from
28. or
29. which
30. one
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>would</td>
</tr>
<tr>
<td>32</td>
<td>all</td>
</tr>
<tr>
<td>33</td>
<td>will</td>
</tr>
<tr>
<td>34</td>
<td>there</td>
</tr>
<tr>
<td>35</td>
<td>say</td>
</tr>
<tr>
<td>36</td>
<td>who</td>
</tr>
<tr>
<td>37</td>
<td>make</td>
</tr>
<tr>
<td>38</td>
<td>when</td>
</tr>
<tr>
<td>39</td>
<td>can</td>
</tr>
<tr>
<td>40</td>
<td>more</td>
</tr>
<tr>
<td>41</td>
<td>if</td>
</tr>
<tr>
<td>42</td>
<td>no</td>
</tr>
<tr>
<td>43</td>
<td>man</td>
</tr>
<tr>
<td>44</td>
<td>out</td>
</tr>
<tr>
<td>45</td>
<td>other</td>
</tr>
<tr>
<td>46</td>
<td>so</td>
</tr>
<tr>
<td>47</td>
<td>what</td>
</tr>
<tr>
<td>48</td>
<td>time</td>
</tr>
<tr>
<td>49</td>
<td>up</td>
</tr>
<tr>
<td>50</td>
<td>go</td>
</tr>
<tr>
<td>51</td>
<td>about</td>
</tr>
<tr>
<td>52</td>
<td>than</td>
</tr>
<tr>
<td>53</td>
<td>into</td>
</tr>
<tr>
<td>54</td>
<td>could</td>
</tr>
<tr>
<td>55</td>
<td>state</td>
</tr>
<tr>
<td>56</td>
<td>only</td>
</tr>
<tr>
<td>57</td>
<td>new</td>
</tr>
<tr>
<td>58</td>
<td>year</td>
</tr>
<tr>
<td>59</td>
<td>some</td>
</tr>
<tr>
<td>60</td>
<td>take</td>
</tr>
<tr>
<td>61</td>
<td>some</td>
</tr>
<tr>
<td>62</td>
<td>these</td>
</tr>
<tr>
<td>63</td>
<td>know</td>
</tr>
<tr>
<td>64</td>
<td>see</td>
</tr>
<tr>
<td>65</td>
<td>use</td>
</tr>
<tr>
<td>66</td>
<td>get</td>
</tr>
<tr>
<td>67</td>
<td>like</td>
</tr>
<tr>
<td>68</td>
<td>then</td>
</tr>
<tr>
<td>69</td>
<td>first</td>
</tr>
<tr>
<td>70</td>
<td>any</td>
</tr>
<tr>
<td>71</td>
<td>work</td>
</tr>
<tr>
<td>72</td>
<td>now</td>
</tr>
<tr>
<td>73</td>
<td>may</td>
</tr>
<tr>
<td>74</td>
<td>such</td>
</tr>
<tr>
<td>75</td>
<td>give</td>
</tr>
<tr>
<td>76</td>
<td>over</td>
</tr>
<tr>
<td>77</td>
<td>think</td>
</tr>
<tr>
<td>78</td>
<td>most</td>
</tr>
<tr>
<td>79</td>
<td>even</td>
</tr>
<tr>
<td>80</td>
<td>find</td>
</tr>
</tbody>
</table>
81. day  
82. also  
83. after  
84. way  
85. many  
86. must  
87. look  
88. before  
89. great  
90. back  
91. through  
92. long  
93. where  
94. much  
95. should  
96. well  
97. people  
98. down  
99. own  
100. just  

Here are the English words in the frequency range 2,100–2,200:

<table>
<thead>
<tr>
<th>2,100</th>
<th>sometime</th>
<th>2,114</th>
<th>pearl</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,101</td>
<td>applaud</td>
<td>2,115</td>
<td>ray</td>
</tr>
<tr>
<td>2,102</td>
<td>underneath</td>
<td>2,116</td>
<td>lazy</td>
</tr>
<tr>
<td>2,103</td>
<td>hello</td>
<td>2,117</td>
<td>limb</td>
</tr>
<tr>
<td>2,104</td>
<td>pretense</td>
<td>2,118</td>
<td>grammatical</td>
</tr>
<tr>
<td>2,105</td>
<td>descent</td>
<td>2,119</td>
<td>beast</td>
</tr>
<tr>
<td>2,106</td>
<td>conquer</td>
<td>2,120</td>
<td>monkey</td>
</tr>
<tr>
<td>2,107</td>
<td>framework</td>
<td>2,121</td>
<td>jewel</td>
</tr>
<tr>
<td>2,108</td>
<td>confidential</td>
<td>2,122</td>
<td>persuasion</td>
</tr>
<tr>
<td>2,109</td>
<td>adoption</td>
<td>2,123</td>
<td>obedience</td>
</tr>
<tr>
<td>2,110</td>
<td>disgust</td>
<td>2,124</td>
<td>sock</td>
</tr>
<tr>
<td>2,111</td>
<td>waist</td>
<td>2,125</td>
<td>vowel</td>
</tr>
<tr>
<td>2,112</td>
<td>momentary</td>
<td>2,126</td>
<td>hammer</td>
</tr>
<tr>
<td>2,113</td>
<td>receipt</td>
<td>2,127</td>
<td>inn</td>
</tr>
</tbody>
</table>
2,128 chimney 2,153 complication
2,129 dissatisfaction 2,154 right
2,130 annoyance 2,155 indoor
2,131 ornament 2,156 lower
2,132 honesty 2,157 actress
2,133 outward 2,158 congratulation
2,134 sharpen 2,159 ounce
2,135 handkerchief 2,160 fry
2,136 greed 2,161 everlasting
2,137 heavenly 2,162 goat
2,138 thirst 2,163 ink
2,139 niece 2,164 disappearance
2,140 spill 2,165 reproduction
2,141 loaf 2,166 thicken
2,142 wheat 2,167 avoidance
2,143 worm 2,168 spoon
2,144 secrecy 2,169 strap
2,145 rude 2,170 deceive
2,146 heighten 2,171 lengthen
2,147 flatten 2,172 revenge
2,148 loosen 2,173 correction
2,149 cheese 2,174 descendant
2,150 rivalry 2,175 hesitation
2,151 royalty 2,176 spade
2,152 discontent 2,177 basin
Note two observations about these lists. First, many words in the first list are function words, such as articles and prepositions, which can often be discarded in a sublanguage. Second, many words in the second list aren't truly necessary for interactive storytelling. You could build a perfectly good sublanguage for interactive storytelling without using such words as "handkerchief," "photography," or "ray." This means your vocabulary can be smaller than 2,000 words—a large but achievable goal.

Moreover, your vocabulary can be extended by a variety of means. All languages sport a set of prefixes and suffixes that extend the meaning of a root word in many directions. Here are just a few of the common prefixes and suffixes commonly used in English:

- baker
- fondness
- nationhood
- kingdom
- resharpen
- terrorize
- fortify
- mandatory
- unfortunate
- antidote
- contradict
- interactive
- nonsensical
- prediction
- producing
- transform
- sublanguage

A small set of these extensions to your sublanguage could increase its vocabulary without taxing players.

**Grammars**

Language equals vocabulary plus grammar. That's a bit of a simplification, but for the purposes of sublanguage construction, it's adequate. A grammar specifies the rules for putting words together into a sentence. Natural-language grammars are horridly complicated: The conjugations, declensions, voices, moods, and gender overwhelm the beginner, and then there are all the exceptions! English has, for example, sink-sank-sunk and drink-drunk-drunker but wink-winked-winked. Clearly, you want to design a sublanguage with a clean grammar.

But what's the best grammar to use? A cleaned-up version of English grammar? Spanish has a particularly clean grammar; perhaps it would be a better starting place. Fortunately, a clean, simple grammar already exists, and it appears to be wired into our brains. Throughout history, a number of revealing linguistic experiments have been carried out as the unintended consequence of immigration. If a large body of immigrants occupies a new territory, not as conquerers but as laborers, the first generation of their children finds itself handicapped by the incompatibility of the language spoken at home and the language of their playmates. In this social context, where neither language can assert itself as dominant, the children invent a new language called a creole. This new language borrows vocabulary from both parent languages, but its grammar is always the same,
no matter where or when it's created. The core structure of creole grammars is simple: There are no conjugations, declensions, voices, moods, or genders. All these functions are handled with auxiliary words. For example, a hypothetical creole based on English might include statements such as:

<table>
<thead>
<tr>
<th>I go store.</th>
<th>I am going to the store.</th>
</tr>
</thead>
<tbody>
<tr>
<td>I been go store.</td>
<td>I went to the store.</td>
</tr>
<tr>
<td>I gwanna go store.</td>
<td>I will go to the store.</td>
</tr>
</tbody>
</table>

Auxiliary words are simply tacked onto the existing words to modify their meaning. As Mr. Spock would say, "crude, but effective."

A completely different approach relies on creating a square matrix with vocabulary words along the top and the left side. Each cell in the matrix represents one pairing of a column-word with a row-word: if that cell is marked True, the column-word is permitted to follow the row-word; otherwise, it is not. In Figure 10.1, for example, the row-word "Go to" can be followed only by the column-word "the house." This approach to grammar is even cruder than a creole grammar, but it's simpler to understand and implement.

A word connection matrix's worst problem is its inadequacy to address contextual factors that might change the word sequence. You can partially address this problem by including special-case rules for these contextual factors. A clean sublanguage might have only a few such special-case rules, but as the language grows, the number of special-case rules explodes.

**Meaning**
Interactive storytelling demands a sublanguage that both computers and humans can use. The program must be able to read any valid expression in the language and understand what it means. But what does "understand" mean in this context? Surely a computer can't understand language in the way that people do. What it can do in response to a sentence is change a storyworld's components. Each sentence must somehow generate changes in the values of variables in the storyworld; presumably these changes will then trigger responses from other Actors.
Thus, “meaning” in the narrow sense I’m using here is nothing more than the changes engendered in the storyworld triggered by a verb. The verb “make” means a new prop will be entered into the database of props; its properties can be partly derived from other components of the sentence containing the verb. For example, “Joe makes a box” means that a new box is placed into the Prop database owned by Joe, and that the components used to make the box are removed from the Prop database.

This example, however, is simple because it’s merely mechanical. Verbs with more dramatic import demand more complex treatment. For example, consider the verb “to love.” It takes an Actor as its direct object. Carrying out this verb requires a change in the relationship between subject and direct object. It could also affect the relationships of each Actor with other Actors.

But it gets worse: What about verbs whose significance depends wholly on the context? The verb “invite home” provides an example. If this event takes place on the schoolgrounds where little Molly invites little Tommy over to her house, it means one thing; if Molly and Tommy are unattached 20-somethings, it could mean something entirely different. You can solve this problem by carefully
choosing the vocabulary to ensure that no ambiguities of this kind result. You can also tighten your storyworld to make sure it’s not populated by both schoolchildren and young singles.

**Inverse Parsing**

People automatically speak natural language in complete sentences. You frame the sentence in your mind, and then speak it as a unit. So when computers came along, programmers just naturally assumed people should talk to computers the same way they talk to each other—except they then designed abstruse languages too difficult for people to remember. There’s no reason that sentences in dramatic sublanguages must be created in their entirety; a sentence can just as easily be assembled word by word.

*Why does it matter whether you construct sentences word by word or as entire units?*

The difference lies in players’ interactions with the computer. If they compose sentences as complete units, they must be certain that every single word in the sentence is perfect. Suppose that a player is only 90% certain of each word in a seven-word sentence; the odds are even lower that the sentence as a whole will be correct. Requiring players to submit sentences in their entirety compounds the uncertainties.

The biggest benefit of word-by-word composition is that the computer can present players with a menu of all appropriate words that can be entered at that point in the sentence. Instead of guessing from memory, players need only select from a menu—a much simpler task.

*But menus are so slow to use!*

Pull-down menus are slow because they require three actions for each selection: click on the menu title, drag down to the menu item, and release. Menus for this kind of interface (which I call an *inverse parser*) can be kept front and center. The player looks directly at the menu choices, identifies the best choice, and clicks on it—just one action instead of three. The inverse parser then recalculates the sentence and, using the sublanguage’s grammar, determines all the words that might
fit into the next open slot. It displays those words in a new menu that replaces the previous menu.

What if there are so many words that they don’t fit into the available space?

Then you make the menu a scrolling window. This occurrence will be rare; with a vocabulary of less than 2,000 words, it’s unlikely that more than a dozen will fit into most slots. Indeed, the context of the situation often narrows down the choice to but a single word, which the inverse parser can fill in automatically for players.

Suppose you’re playing as Tom, occupying a Stage with just one other person: Diane. You enter “Tom” as the first word in the sentence (after all, Tom is the only Actor you control), and then you choose the verb “asks.” The inverse parser can figure out that Diane is the only Actor the question can be directed to, so it fills in “Diane” as the direct object of the sentence. This kind of one-word choice happens all the time with inverse parsers.

Every sublanguage must have grammar, and every sentence must be contextually appropriate. If you use a regular parser, you must write code that analyzes the grammatical and contextual appropriateness of players’ input—but you’ll have to disappoint players with error messages explaining that their sentence input makes no sense. With an inverse parser, you use the same code that analyzes grammatical and contextual appropriateness, but you execute that code before players make their decisions, not after they’ve made their mistakes. With an inverse parser, grammatical and contextual errors are impossible.

**Semantic Networks**

Realizing the problems inherent in language comprehension, computer scientists have been assembling a variety of semantic networks, which are data structures that connect words by their various meanings. The most advanced semantic network is WordNet, also known as a *lexical database*: a huge database of more than 150,000 English words with all their logical connections. Each word is connected to each of its synonyms, antonyms, hypernyms, hyponyms, meronyms, and holonyms. You already know what synonyms and antonyms are. A *hyponym* is a specific instance of the word in question; thus, “spaniel” is a hyponym of
"dog." A hypernym has the reverse relationship, so "mammal" is a hypernym of "dog." A holonym is an object that includes the word in question as one of its parts; thus, "body" is a holonym for "toe." The reverse case is that "toe" is a meronym of "body."

WordNet includes additional information, such as classes of words (nouns, verbs, adjectives, and so forth) and some additional relationships among words. The project continues to develop as researchers expand the database to include ever more useful information.

Another example of a semantic network is the Visual Thesaurus, it presents the synonyms of a word, and their synonyms, as a three-dimensional structure. It's visually striking and screams with potential.

Many researchers are exploring the potential of semantic networks. Elizabeth Figa and Paul Tarau at the University of North Texas, for example, have explored the possibilities of using semantic networks for interactive storytelling.

See Chapter 19, "Research," for more on Figa and Tarau's work on using semantic networks for interactive storytelling.

Wouldn't you need a superpowered machine to parse all that data?

Nope. WordNet is distributed in a form that runs on a modern PC. Its operation isn't optimized for speed, but it still runs plenty fast for our purposes. Remember, inverse parsing takes place one word at a time, between player inputs. The code can fiddle around with the parsing problem for, oh, half a second before a player would notice the delay. Half a second represents a billion machine cycles on a modern PC. A computer can parse of a lot of information in a billion steps.

**Graphical Languages**

A sublanguage for interactive storytelling has one gigantic advantage over natural language: It's visual rather than auditory. Auditory expressions are one-dimensional, but visual expressions have multiple dimensions (horizontal, vertical, color, texture, and animation). The visual expression of natural language, writing, remains one-dimensional in structure even on a two-dimensional page.
Failure to take advantage of the computer’s multidimensional capability would be a mortal sin against the Muse of Design. I first used a graphical language in 1986 with my game Siboot. My sublanguage used icons that were linked into sentences, as shown in Figure 10.2.

![Figure 10.2: Linking icons in Siboot.](image)

In this screenshot, the lower-right pane shows a short sentence: “Wiki greets Vetvel pleasantly.” Wiki is the catlike icon, and Vetvel is the horned icon. The verb “greet” is in the center, and the adverb “pleasantly” modifies the verb.

Player input is handled in the upper-right pane; it shows a complex sentence the player has almost completed. (This is an example of an inverse parser at work.) The almost complete sentence says “Vetvel offers to tell Wiki what Gardbore’s shial-value is if in return Wiki tells Vetvel…” The notion of the deal is communicated by the central handshake icon. The verb “tell” shows up in two places. Vetvel’s “tell” is modified by the icon for Gardbore (the long-eared creature), which in turn is modified by the value for “shial” (a critical quantity in the game). On the right side of the sentence, Wiki’s “tell” hasn’t yet received its modifier and so has a hanging connector. A list of icons shown at the far left could be used as modifiers; each icon represents a character.

An important concept used in this sublanguage is the notion of autofilling. The sublanguage’s software is intimately connected with the game’s database (as it must be if it’s to be useful). The software knows what words fit into which situations, so it can prune the menu of available words to display only permissible words. In the preceding example, the software has restricted the set of choices.
to include only those secrets that Wiki has available to trade. It can even exclude
secrets Wiki has to trade that Vetvel already knows, so wouldn’t be interested in.

The big benefit of sublanguages is that they focus players on what’s possible and
make it impossible to conceive of utterances that aren’t permitted. The sublan-
guage contains the rules as well as the substance of the storyworld—a powerful
feature.

But wouldn’t it be difficult for people to learn a new language just to play
a storyworld?

Don’t underestimate the facility with which people pick up languages. The aver-
age tourist can start picking up the rudiments of the local language after only a
week. If people can begin learning complex natural languages so readily, imagine
how easy it would be to learn a narrowly defined sublanguage.

The sublanguage in Siboot has a vocabulary of about 80 words—too skimpy
for interactive storytelling. Larger vocabularies require more complex software.
Fortunately, computers’ graphical powers have improved dramatically in the 17
years since I designed Siboot’s sublanguage. It’s no problem to use word icons
that are 64 pixels wide rather than the 32 pixels I had to use for Siboot. Doubling
the icon’s size quadruples its theoretical expressive power. In addition, I had to
use black and white for my icons, but now you can use 24-bit color; that alone
increases icons’ theoretical expressive power by a factor of 24. In other words, 64-
pixel icons on modern computers are about 100 times more expressive than the
icons I used for Siboot.

Basic English

Basic English was created about 80 years ago in an effort to provide a simple,
easy-to-learn variant of English. It uses a small, carefully selected vocabulary
design to cover all situations and contains about 1,000 words, with only 18 verbs:
come, get, give, go, keep, let, make, put, seem, take, be, do, have, say, see, send,
may, and will.

With this system, many more verbs can be assembled by combining verbs with
nouns. Thus, the verb “attack” becomes “make an attack”; “cry” becomes “have a
cry.” By reducing verbs to the absolute minimum, the designers of Basic English
were able to simplify the language. Unfortunately, the system makes heavy use of
prepositions to extend the meanings of verbs in ways that make sense to people but aren’t so easy for a computer to understand. “Dedicated” is translated as “given up to”; “die” is “go to death.” I suspect that a reworked version of Basic English could be developed into something useful for interactive storytelling, but first someone must build a better system for handling verbs. With that done, the next task would be creating a large semantic database defining what all the words mean. This is a huge task, but certainly not an impossible one—and after the system was built, it could be used in a wide variety of products.

**Pictorial Languages**
A lush garden of resources on pictorial languages flourishes on the Web. Some of the more interesting are discussed in the following sections.

**Bliss**
Bliss is a symbolic language developed over several decades in the mid-twentieth century. It was applied as a language for handicapped children in an experiment in Canada, with impressive results. It has since been expanded, revised, and refined. The language is built from 120 “key symbols,” which then combine into more than 2,000 words. Figure 10.3 shows a sample.

![Bliss symbols](image)

**Figure 10.3:** In Bliss, “much good feeling person” equals “friend.”

**The Elephant’s Memory**
Timothee Ingen-Housz has created a beautiful graphical language called The Elephant’s Memory. It uses a brilliant set of icons that work well together.

There’s a catch: Although the language is a tour de force in graphic design, Mr. Ingen-Housz’s genius is not all-encompassing. The language has 156 words, and there seems little chance of it growing beyond that. Words that are easily drawn, such as “elephant” or “rabbit,” are present, but the language lacks many crucial words.
Conlangs
The delightful term conlangs is a contraction of “constructed languages” (as opposed to languages that evolved naturally). Believe it or not, plenty of people love creating their own languages. Few of these languages have any relevance to the problems you face in creating a sublanguage for interactive storytelling, but these efforts will certainly broaden your thinking. To find out more, just search the web for “conlang” or visit http://www.langmaker.com/. Another resource for pictorial languages is at http://vlado.fmf.uni-lj.si/seminar/28jul99/PicLang.htm.

Wrapping Up
One way or another, your interactive storytelling engine must communicate with your players. Every computer program creates a tiny sublanguage that it uses for communication with its users. You could cobble something together with menus, dialog boxes, pushbuttons, and so forth; after all, this is what people have been doing for decades. But for interactive storytelling, the range of expressions your players will require is so great that it’s worth your while to go whole hog and create a formal sublanguage.

CHAPTER 13
Verbs and Events

ONE OF THE IMPLICATIONS OF applying verb-based thinking to interactive storytelling is that the central data structure should be the Verb. The answer to the classic question "What does the user DO?" is "The user does Verbs." Because verbs define the nature of any software, it's only natural that defining the Verb data structure is the central problem in all software development.

A Verb, in this context, is not quite the same thing as a verb in a sentence; it's functionally similar but a bit more abstract. A Verb is anything a user can tell the computer to do. The most obvious Verb is clicking a button, but contextual changes can create new verbs.
Thus, if I push the mouse button while the cursor is on the Delete button, I’m specifying an entirely different Verb than pushing the mouse button on the Save button.

**Verb Counts**

Verb counts in most software have always been low, but with the passage of time, those counts have risen inexorably. MacWrite, the first true consumer WYSIWYG word processor, didn’t have a great many verbs. Aside from the obvious verbs for text entry, there were verbs for setting font types, sizes, and styles; verbs for justifying text and indenting paragraphs; verbs for inserting images; and verbs for setting and clearing tabs. All in all, I’d guess there were fewer than 100 verbs in MacWrite. But nowadays, Microsoft Word has so many verbs that when I assign students the task of counting them, they lose count somewhere around 300. What with style sheets, footnotes, index entries, tables of contents, and so on, there are just too many verbs to keep track of.

The same goes for games. The verb counts for games of the early 1980s were generally lower than 10, perhaps because of the simple nature of the joysticks used for input. The only inputs possible were left, right, up, down, and the “fire” button: five verbs. Since then, games have become more complex and verb counts have grown, but in general, the verb counts of today’s games seldom exceed 30 and usually average about 15.

No matter how you design it, interactive storytelling requires hundreds or even thousands of verbs. Consider the verb count in this bit of fiction:

> Alas, poor Yorick! I knew him, Horatio: a fellow of infinite jest, of most excellent fancy; he hath borne me upon his back a thousand times. And now how abhorr’d in my imagination it is! My gorge rises at it. Here hung those lips, that I have kiss’d I know not how oft. Where be your gibes now, your gambols, your songs, your flashes of merriment that were wont to set the table on a roar? Not one now to mock your own grinning—quite chap-fall’n? Now get you to my lady’s chamber, and tell her, let her paint an inch thick, to this favor she must come; make her laugh at that.

This single short soliloquy, just eight lines of text, has 12 different verbs. It constitutes less than half a percent of the entire play; clearly the play as a whole
boasts hundreds to thousands of different verbs. Interactive storytelling engines must be capable of handling such large verb counts.

**Lesson #34**

*Interactive storytelling requires thousands of verbs.*

**Specific Versus Generalized Verb Handling**

With low verb counts, designers can craft each verb individually, writing custom code for each one. This method worked fine when you could count your verbs on your fingers, but when you start needing your toes to count, things became more difficult. A program such as Microsoft Word is an intricate maze of code, requiring the efforts of hundreds of programmers, yet its complexity causes it to break with depressing regularity. The difficulty of managing hundreds of custom-coded verbs has restrained software designers. Storybuilders, however, cannot evade the monster; their work requires managing thousands of verbs. You simply must dispense with the notion of custom-designing every verb. You need a more generalized approach, something more abstract that allows you to create, design, and program thousands of verbs.

The first step in creating a generalized system of verb handling is to design a data structure for Verbs. You need a general-purpose table of properties that define each Verb. Here are some variables that might belong in such a table:

**Note:** Henceforth, I'll be contracting "Direct Object" to "DirObject."

- **Name:** A simple text label for the Verb.
- **Import:** A number indicating how newsworthy this Verb is.
- **TimeToPrepare:** The amount of time required to elapse from the time the Actor decides to execute the Verb until execution begins. Although most Verbs require zero TimeToPrepare, a few might require some. For example, if an Actor decides to poison someone, some time might be required to get the poison. An Actor wanting to hop into bed with another Actor requires a moment to remove clothing.
- **TimeToExecute**: The amount of time required to carry out the Verb. Again, most Verbs require just a moment to execute, but a few have extended execution periods—for example, GoJogging, BakeCake, or BuildBoat.

- **Audience**: Every Verb has certain requirements for the presence or absence of other Actors. Most of the time, the DirObject must be located with the Subject, but in a few cases the reverse is true. For example, should the Subject be hatching some dark plan aimed at the DirObject, the Subject most definitely wants to carry it out without the presence of the DirObject. MixPoison, DigGrave, or DonWeddingDress are three Verbs that require the absence of a DirObject.

Some Verbs require the privacy of Subject and DirObject and, therefore, the absence of others. Verbs used in romantic situations are often like this. A few Verbs require the presence of a third party, either as a witness or as party to the action. And of course, there are also specifications for whether other Actors present can witness a Verb being carried out. All told, I have identified 10 specific situations that require particular combinations of people present or absent:

- **MentalState**: This situation represents an action inside the Subject's mind; others can be present but won't be able to witness it.

- **AnyAudience**: The most common situation; the Actor executing the Verb doesn't care about the presence or absence of any other person.

- **RequireWitness**: The Subject will not execute the Verb unless some third party is present to witness the Event.

- **SubjectOnly**: The Subject wants to carry out the Verb in secret and must be alone.

- **SubjectAndDirObjectOnly**: The Subject and DirObject both must be present, but must have privacy.

- **AllAudience**: This is a special case for an Event so sensational—the 9/11 attacks, for example—that it is, in effect, instantly made known to all Actors.

- **ThreeParty**: An Event requiring the presence of three parties: Subject, DirObject, and a third Actor.

- **FourParty**: An Event requiring the presence of four parties: Subject, DirObject, and two other Actors.
- **CheekByJowl**: This is a special case in which two Actors are together in a public place, whispering together. Their actions can't be witnessed by anybody except a person deliberately spying on them.

- **AnybodyBut**: Another rare case, it requires Subject and DirObject, but a specified third Actor may not be present. This situation would arise when, say, Subject is insulting the third Actor to DirObject.

These are the variables most likely to be of value in any verb-based interactive storytelling engine. My own engine uses many more variables, but they are specific to that engine and not worth exploring in this general discussion.

**Events**

When a Verb is executed, it's part of an Event. My use of the term “Event” is very close to its normal use, except that, for purposes of interactive storytelling, Events must be reducible to computer expression. The real world is full of complex and tricky events that defy computational expression, so the requirement of making Events fit inside a computer expression does seem to restrict your creative freedom. But if Michelangelo had to fit his work on the ceiling of the Sistine Chapel, you can work within the constraints of the computer, right? The data structure for Events looks something like this:

```
  something or other, something or other VERB something or other,
  something or other
```

What are all those “something or others”? It should be pretty obvious from the way I've written it that they are things like subjects, direct objects, adjectives, adverbs, and prepositional phrases. For the moment, I'll keep it as simple as possible:

```
  Subject Verb
```

This data structure is clean and simple, but it's too constrained for most use. Yes, in a few situations this structure works adequately, as in these examples:

- Sammy slept.
- Mary fretted.
- The fruit ripened.
Face it, however: This kind of sentence just won't handle most situations, but it does bring out an important point. The first two sentences look fine, but that third sentence has "fruit" for a subject. That's a tad peculiar. Yes, we use sentences like this all the time, but for the purposes of drama, requiring that the subject of every sentence be an Actor is more appropriate. In other words, inanimate objects can't do things.

*But how would you say "The fruit ripened" if fruit can't be the subject? Doesn't this make it impossible to record events that are outside the control of any Actor? What about natural events, acts of God, or accidents?*

The trick is to use an all-purpose character I call "Fate" to handle these events. Thus, the sentence becomes "Fate ripened the fruit." Actually, I go even further in my engine: I require that every Event take an Actor for Subject and an Actor for DirObject. This forces the peculiar sentence: "Fate ripened Fate the fruit." In other words, Fate does this verb to itself, and the fruit is an indirect object. The principle is simple: Subjects and DirObjects must be Actors. It's occasionally clumsy, as in this example, but then again, how much drama is there in ripening fruit?

*I have more to say about Fate in Chapter 18, "The Erasmatron."*

Getting back to the sentence structure, expand it one step:

**Subject Verb DirObject**

This looks more like a regular sentence. It enables you to handle a broad range of events, such as:

- Joe greeted Tom.
- Tammy missed Jeanette.
- Mordred laughed at Lancelot.

*Just a minute, here! In the third sentence, Lancelot is not the direct object; he's the object of the preposition "at."*
Grammatically, you’re correct. But is there any functional difference between “Mordred laughed at Lancelot” and “Mordred derided Lancelot”? Aside from some slight differences in nuance between the two verbs, the two sentences say the same thing. Mordred did something to Lancelot. For storyworld purposes, the Verb of the sentence is “laughed at.”

*But that’s not the way English grammar works!*

True, and if you hew to the hard lines set down by the grammarians, you’ll never get anywhere. The fact is, language is immensely complex, and you can’t model all that complexity. What’s needed is a gross but functional simplification of the structure of a sentence. You can’t afford to compute with real sentences, so you need Neanderthal sentences that primitive algorithms can handle. As part of this, you’ll be simplifying away much of the richness of language. However, reducing “laughed at” to a single verb is fairly straightforward; it doesn’t wreak havoc with language. Besides, how do you know that a hundred years from now, people won’t have contracted it down to “laftat”?

*But what about “John gave Mary the book”??*

You could use an indirect object to handle this problem, but lots of other bits and pieces can get tacked onto the basic Subject-Verb-DirObject sentence structure. Here are some examples:

- Fred met Jane in the parking lot.
- Veronica kissed Toby in front of Anthony.
- George traded his decoder ring to Meredith for her marble.

Or how about this monster:

- Jeanette warned Mika not to be around when Paul arrived.

And there are tons of other even messier sentences you can imagine. You can dismiss many of the complex constructions on the grounds that you couldn’t compute with them anyway. At the same time, however, you have to admit that the simple Subject-Verb-DirObject structure is inadequate. So how do you expand it neatly?
Chomskian, or Recursive, Data Structures

The common solution to the problem of not having the complex constructions you need is to structure sentences in the manner always used in computational linguistics. Each sentence consists of a noun phrase and a verb phrase. A phrase is a compound structure containing a main word and any modifying phrases. A noun phrase is composed of a noun plus phrases that modify it; a verb phrase is composed of a verb plus phrases that modify it. In many cases, there are no additional modifying phrases, but each modifying phrase can contain its own phrases. Thus, the Chomskian sentence is a recursive data structure, containing phrases nested inside phrases.

This data structure makes any programmer’s mouth water. Recursive data structures are elegant, powerful, and frigorous. All computational linguistics use such data structures, but are they useful for interactive storytelling? I think not. They’re too messy. Every one of those nested phrases has to be interpreted within the context of the sentence. Now, programmers in computational linguistics have succeeded in building programs that accomplish this task, so it isn’t technically impossible; indeed, it really isn’t that difficult. The problem lies in applying all those nested meanings in a dramatically significant fashion. Sure, you can write a program that can figure out something like “The man with the broken arm gave the dog from around the corner a bone that he had purchased from the Armenian butcher that morning.” But figuring out the semantic relationships between all those words is considerably less difficult than figuring out the impact each word has on the event’s dramatic significance. Someday, storybuilders will be able to handle this kind of sentence. For now, you have to crawl before you can walk, and I advise you to stick with simpler sentence data structures.
Flat Data Structures
I prefer to use flat data structures for sentences. This sentence structure handles
the majority of all sentences you'll need to use in interactive storytelling:

Subject verb DirObject IndirectObject

Unfortunately, handling the majority of sentences you'll need to use isn't good
enough; a few special sentence structures always need to be included. They're
rare but essential. Here are a few of them:

Subject Trades X (to) DirObject (in return for) Y.

Subject Tells DirObject (that) (he/she) likes ThirdPerson.

Things really go to hell when you nest clauses:

Subject Asks DirObject (to) give (him/her) X.

Subject Tells DirObject (to) go (to) Y.

Subject Tells DirObject (that) ThirdPerson hates FourthPerson.

Subject Tells DirObject (that) ThirdPerson told (him/her) (that)
FourthPerson (did) Verb (to) FifthPerson.

The sad truth is that language is infinitely extensible, and without recursion, you
can never hope to handle every reasonable sentence structure. The only recourse
is to constrain the storytelling engine to handle a limited subset of all such sen-
tences. This is one of the most painful concessions you must make to technologi-
cal limitations. Indeed, it's so painful that time and again I have returned to the
problem of recursive sentence structures, hoping to find a way to make them
work. Perhaps you'll be able to see what I can't and solve this wretched problem.
In any case, my own solution is to add two secondary objects (SecObjects) to the
sentence structure:

Subject Verb DirObject SecObject1 SecObject2

This structure handles even more of the situations I have found necessary in
my engine, but it's not quite as simple as it appears. The Verb defines the pre-
cise meaning of each SecObject, so the sentence "Subject trades SecObject1 to
DirObject for SecObject2" would be represented as:

Subject Trades DirObject SecObject1 SecObject2.
The algorithms for the `VerbT`r`ades` must specify that SecObject1 indicates the item given by Subject, and SecObject2 indicates the item given by DirObject.

The same structure is used very differently in the sentence “Subject asks DirObject to give him SecObject2”:

```
Subject Asks DirObject SecObject1 SecObject2.
```

Here SecObject1 supplies the Verb that Subject is asking DirObject to perform, and SecObject2 supplies the item to be given. Again, the algorithms specific to the Verb Asks must indicate the meaning of SecObject1 and SecObject2 in this context.

Having the Verb provide the context under which SecObjects are interpreted is entirely reasonable; this is certainly the case in normal language. However, it does impose additional expectations on the designer as well as the storybuilder; somehow those interpretations have to be built into the algorithms for the Verb. Moreover, the storybuilder must keep those contextual requirements in mind while using the Verb. Misunderstandings between designer and storybuilder here can be the source of many difficulties.

There's no reason that you couldn't use three or more SecObjects; the only problem is that when building a storyworld, you can quickly get lost keeping track of all of them.

**What about all those other elements of a sentence, such as time and place?**

I attach a number of housekeeping variables to my sentence structures. For example, a variable called `when` records the exact time the Event took place, a variable called `where` records the Stage on which the Event took place, and several other variables keep track of information required for my Gossip system, which I'll explain in Chapter 14, “HistoryBooks and Gossip.”

**Wrapping Up**

The first step in reducing stories to computable form is to reduce options to Verbs and actions to Events. The ideal structure for an Event is a sentence, although some liberties must be taken to make sentence structure easily computable.