Every game in the world is made up of rules. In this chapter, we’ll talk about designing those rules, which I divide into the categories of “verbs” and “objects,” and the relationships between them that create the experience, the dynamics, of the game for its player or players. Those rules are the characters in our story and, like any characters, our stories are most effective when we develop them and their relationships fully.
Rules

Games are made of rules. Surround stones of the opposite color with stones of your own color to capture it. Complete a line of blocks to make it disappear. Reduce the opponent’s health to zero to eliminate her. It’s the interplay of these rules, the interactions between them, that creates an experience for our players.

As game creators, we want to design the rules that will make for the strongest experience. We want to design rules that have relationships to each other. We want to design rules that have the opportunity to develop as the game goes on and avoid rules that we won’t be able to develop. I don’t mean “develop” in the sense of “game development”—a rule develops as the player’s relationship with it grows deeper, more complex, and more refined, as she finds new ways to work with the rule and understands the nuance of how it affects her experience of play.

John Newcomer and Bill Pfutzenreuter’s Joust, a 1982 coin-operated arcade game, has rules that support and strengthen each other. Joust is a game about ostrich-back gladiators who joust with spears in a desolate arena. Here’s one rule of Joust: when two gladiators collide, the one who is highest defeats the other. Another rule: pressing the button makes the player’s ostrich flap its wings and gain a little height. A third rule: the constant pull of gravity causes all the ostriches to fall downward, toward the bottom of the screen (Figure 2.1).

![Figure 2.1 Basic rules of Joust.](image)

So you can see how these rules work together to create an experience that demands skillful play with attention to one of the main themes of Joust: height is important! Flapping to maintain height is critical because gravity keeps lowering your height, and the higher gladiator will always win in a collision. There’s also a shrieking pterodactyl that sometimes flies through the arena, devouring player and enemy alike. To slay the pterodactyl, a player has to strike it directly on the nose. So the pterodactyl develops the rules about height even more: slaying a pterodactyl demands accuracy and control and represents a significant moment in the game.

Having established the rules of the game, we’re then interested in communicating those rules to the player as succinctly as possible and developing those rules through design. Later
chapters in this book will be about that. This chapter is about establishing a basic vocabulary with which to understand and discuss rules and how they function in a game—a grammar.

Writing this book, I’m using an established grammar to structure the words and sentences you’re reading—hopefully to the end of communicating my ideas as clearly as possible. Writing is a creative work, though, so sometimes I ignore certain rules or conventions when I think it means the results will be more expressive. So let that be a caveat: none of the “rules” I’m going to discuss are law, are immutable. The purpose of this text isn’t to constrain your design but to suggest ways to start thinking and talking about it.

Games are made of rules. We’re going to think of rules as the characters in our games that are going to be developed over the course of the game. When we talk about story, that’s what we’re talking about: the development, conflict, climax, and resolution of the rules. Rules are characters. Got it?

Who then are the most important characters—the main characters? You might be tempted to say the protagonist, the hero, the gladiators in Joust. The nouns, right? Because in linear stories we’re used to nouns—people—being the ones who develop. But the protagonists of our games, the nouns, aren’t rules.

Verbs are a kind of rule; they’re the most important rules of a game. By a “verb,” I’m referring to any rule that gives the player liberty to act within the rules of the game. Any rule that lets the player change the game state. Any rule that lets the player do something. Verbs are the rules that allow the player to interact with the other rules in the game: “jump,” “shoot,” “fall,” or “flap” in the case of Joust. Without verbs we have a simulation, not a collaborative story-telling system.

Is it hard to think about verbs as main characters in a story? It’s easy to think of the hero as the main character, because verbs characterize the hero. Maybe climbing the sloping foothills at the beginning of the game is easy, but climbing the precarious precipices near the end is much harder. And maybe that suggests both that the hero’s journey (not to be confused with the story structure my Guildhall teacher insisted we base every one of our games on) is getting more arduous as she approaches her goal and that the hero is being tested and becoming stronger to meet these challenges.

But we can’t design the player or her behavior. We design the rules that shape her experience, her choices, her performance. Rules are how we communicate. Verbs are the rules that allow her to communicate back. The game is a dialogue between game and player, and the rules we design are the vocabulary with which this conversation takes place.

When a game’s creator relies exclusively on animated cutscenes or text dumps to tell a story—when she only uses noninteractive means of telling a story in an interactive game—it’s because she has misunderstood how to think of the story in terms of how the player is allowed to actually act within it.
Chapter 2

Verbs and Objects

Creating Choices

Visualize Venus. An endless green sky, purple mountains piled on the horizons like clouds, a yawning cleft like a mouth, leading into the smoking bowels of the planet. The Robot Mines. Not mines staffed by robots—mines where humans dig for technology abandoned by a much older, much more extinct, spacefaring race. Or where they normally do, except that today at 481900 hours—Venus’ days are much longer than Earth’s—the unearthed robots suddenly, simultaneously, blinked to life. Immediately and as one, operating under orders hard-coded into their circuits millennia ago, they seized control of the mine and took the human workers hostage—those that didn’t manage to escape.

Now it’s up to Janet Jumpjet, space-hero-for-hire, to explore the mines, incapacitate the robots, and rescue the human hostages. She’s armed only with her wits and the Megablaster 3000 Laser Pistol—shooting it is her primary verb.

What does it mean that Janet’s main verb is “shoot”? Probably that there’s going to be a lot of gunfire in this game. Venus is a violent place. That’s the future for you.

Janet’s Megablaster fires a single laser bolt at a time—enough to melt one of those menacing robots. POW! But firing the Megablaster generates a tremendous amount of heat—it takes half a second to cool down between shots. This is a rule that we’ve designed. We probably spent a lot of time playing with exactly how long the duration between shots is, tweaking the number, playing the game, and trying to decide which made for the most interesting choices. We’ll probably tweak it a lot more before we consider the game done.

The duration between shots is part of the “shoot” verb. The rule: pressing the button fires the Megablaster ahead of Janet at a rate of one laser bolt every half-second. Why is this important? Because we can use rules to set up choices for players. A choice can be whether to shoot Janet’s Megablaster, or when, or where. If there’s a half-second duration between shots—maybe that doesn’t sound like a very long time, but it’s ages when you’ve got a crazed robot clanking toward you—what choices does that create? Janet’s pinned in a dark corridor, one that looks a lot like Figure 2.2—there are two robots clambering toward her from two different directions. Which one does she shoot first?

Figure 2.2  Rules offer choices: shoot the left robot or the right one?

In fact, Space Invaders, the 1975 game of using a moving gun turret to destroy invading aliens, presents the player with similar choices. In that game, the player can have only one bullet on
the screen at a time—she has to wait for a shot to strike an enemy, or leave the back of the screen, before she can fire another one. How does that affect the player’s choices?

For one thing, a miss means that the player has to wait a while before she can fire again, but a hit allows her to fire again more quickly. So she’s given an incentive to pick her shots more carefully. Shooting invaders who are closer—who are a more immediate threat—afford the player a greater rate of fire than ones who are far away. There’s some balance here: the player has more ammunition to use against enemies who are more dangerous. A miss against a close enemy means a longer interval before the player can shoot again—but nearer enemies are also easier to aim at.

## Verb Relationships

In fact, there are a few different verbs that are interacting in *Space Invaders*. How does the player interact with the game other than shooting? She moves left and right. There’s a relationship between these verbs: when she presses the shoot button, the shot is fired from her current position. Lining up shots with an invader means moving into the range of enemy fire. Dodging enemy fire means moving behind an obstacle, where the player can’t return fire. There’s an ongoing dialogue between moving left and right and firing shots, and the left/right verbs are more developed as a result. You use them to aim your shots and to avoid being hit.

By establishing relationships between verbs, we gives ourselves more opportunities to design choices. The relationship between those verbs is also something that we can develop over the course of the game, the same as with any two characters in a story.

At the heart of *Super Mario Bros.* is a strong relationship between Mario’s ability to move horizontally—to walk left or right—and his ability to move vertically—to jump. But notice that that relationship becomes stronger over time. At the beginning of the game, the player isn’t expected to coordinate these verbs in a very complicated way.

The first jump the game requires is over an enemy that moves of its own volition. Mario is safe if he lands on the enemy—he only has to avoid being touched by the side. This jump doesn’t even require horizontal motion: Mario can jump straight up while the monster walks underneath him. The next jump the game requires is onto a stationary, solid obstacle from the ground next to it. This requires a minimum of horizontal motion. It isn’t until much later in the game that the player is expected to perform really complicated jumps, with careful management and coordination of both horizontal and vertical movement (Figure 2.3). By that point, the player understands how to use horizontal and vertical movement as a pair of verbs that work together in harmony.
Chapter 2  Verbs and Objects

Figure 2.3 Development of the relationship between horizontal and vertical movement in Super Mario Bros.

What we as creators want to avoid are orphaned verbs. An orphaned verb has no relationship to the other verbs, so the other verbs don’t reinforce it, it doesn’t grow, and the player has forgotten about it by the time she reaches the one situation that demands it. Imagine a game where the player finishes each level by using the “open” verb on a door that exits into the next level. If there’s only a single door in each level, this verb is an orphan: it never gets used for anything else and doesn’t have the opportunity to develop in relation to other verbs and varying situations.

How do we avoid having vestigial verbs? Design the game so that the verbs you’ve given the player are sufficient to perform everything you ask of her. Increase their utility by giving them more interactions. If you play a bunch of videogames, you might be surprised how many ways there are to open doors.

We’re back on Venus. Janet Jumpjet is squinting at a door, an ancient metal bulkhead, in the darkness of a subterranean mine, Megablaster smoking. How is our hero going to get to the other side of this door? Is she going to knock politely and wait for someone to let her in? Is she going to put down her Megablaster and turn the door handle? Or is she going to point her Megablaster at the door and pull the trigger?

Give the important verbs in the game as much to do as possible, so you won’t be forced to fill the void with a bunch of secondary verbs that never get developed.

Making Verbs Robust

We want our verbs to be as developed as possible. We want them to be well-rounded characters. That doesn’t just mean that they interact with as many other rules of the game as possible, but also that every interaction the player expects to have a reaction does. Verbs are the rules the player uses to learn the rest of the game’s rules. If she uses a verb some way and is given no feedback, she doesn’t learn anything about the verb or about the rules of the game. We want robust verbs that communicate with the player, even if just to say, “No, you can’t do that.” That seemingly negative statement can be just as important as showing the player what she can do.
Here’s an example from my own work. In 2009 I made a game called *Tombed* about an archaeologist named Danger Jane. Investigating a deep crypt, she’s pursued downward, through layers of fragile earth, by a descending spike wall—the quintessential tomb trap. She’s armed with a shovel, which she can use to dig through the soft clay blocks that she finds underfoot. “Digging” is a critical verb, established as such as early as the title screen, which shows Jane on diggable ground with a shovel in her hand and the instruction “Press Shift to Start.” When the player does so, Jane digs through the floor—every touching, like-colored block (there are three colors) is considered the same piece of clay for purposes of digging—and plummets off the title screen and into the game.

So the player now knows what Jane’s important verb is (“dig”), what key to press to trigger that verb (Shift), and the effect of digging on soft clay blocks, many of which she will encounter in the game. Jane’s other verbs are walking left or right when on stable ground. When she has no ground beneath her feet, she falls until she lands on some.

In addition to the diggable clay blocks, though, there are solid, metal blocks that Jane can’t dig through. These are used to constrain Jane’s movement: to create choices for the player. Maybe she has to run around a metal obstacle instead of digging through it, allowing the spike ceiling to close in on her. Maybe she has to wait for the spike ceiling to drop far enough to destroy the metal blocks for her. That’s another rule: the spikes destroy any blocks they touch—even metal ones. So here the metal blocks work as a pacing mechanism, a solution to Jane getting so far ahead of the spikes that she’s off-screen, where the player can’t see and make decisions for her.

But how does the player know this? How does she learn that Jane’s shovel—which she has been taught can destroy obstacles—can’t break through metal? I’ll tell you how. She strikes that first metal obstacle with her shovel, and the game provides her with feedback to show what happens.

Here’s what happens when Jane’s shovel hits an unbreakable metal block: it bounces off with a metal “ting” sound (Figure 2.4). Even when the player is unable to use a verb to break through an obstacle, there’s still an observable effect that gives the player information about the relationship between the verb and the obstacle. The rule “Jane can’t dig through this” is taught or reinforced when the player uses her verb.

Figure 2.4  Jane attempting to dig through clay and through metal.
In fact, the way the game introduces the rule is this: the player begins by having to dig through three different layers of clay blocks—one of each color in the game. Figure 2.5 shows the first scene of the game. Each color, when struck, crumbles every touching block of the same color, but not ones of different colors. These are the most basic rules of the game and the things the game teaches first.

![Figure 2.5 Opening scene of Tombed.](image)

When Jane has dug through all three colors, she’s at the bottom of a well of metal with raised sides. She can strike the metal with her shovel, but it’ll just bounce off with a *ting*. In a few moments the descending spike wall will reach the top of the well’s raised sides, shattering the whole piece of metal and freeing Jane, who falls to the ground below. So now the player has most likely observed the “Jane can’t dig through metal” rule and the “spike ceiling can dig through metal” rule.

Every interaction that the player could reasonably expect to have an effect should have one, even if it’s negative—that’s what I mean by a robust verb. If the player sunk her shovel into the metal and nothing happened, the objects didn’t seem to touch, or they just passed right through one another, the player might still figure out that she can’t dig through these blocks, but we haven’t sold the rule as strongly or effectively. Maybe it takes the player a little longer to figure out, and while she’s doing so, the spike ceiling comes down and crushes her. Now she has to go back and repeat the whole thing. She has wasted time and maybe not even learned anything.

That’s bad design. As creators, we want to teach and reinforce rules wherever and whenever we have the opportunity.
Explaining with Context

Back on Venus, Janet is ready for adventure, Megablaster in hand. Although she’s a fictional character in a story, the world she lives in is a simulation. In the computer that’s running the simulation, Jane exists as a bunch of numbers: her horizontal and vertical position on the screen, which direction she’s facing, and the speed at which she’s moving. The laser bolts she fires from her gun are just signals: they have a speed and a direction. When the computer detects that one of these signals is overlapping an appropriate recipient—a robot, another x and y position with a direction and a speed—the simulation resolves the collision by removing both objects. In the abstraction of the rules, the math of the game, this is all we see.

We explain these rules to the player by giving them a context that she’s familiar with, one that helps her understand them. The context of a game is composed of many pieces: the images that represent Janet and her Megablaster, the words that appear to describe these things, the way the images animate, the sound effects that accompany play, and even the timing that brings them all together. We’re used to thinking of these elements as parts of the narrative of a game—the story of Janet Jumpjet rescuing hostages in the mines of Venus—and they do arise from that story and help tell it. But in a game, contextual elements do something more: they illustrate and help make sense of what’s happening in the other story, where our rules are the main characters.

Janet has a gun, so she can fire laser bolts. This is a robot; laser bolts explode it into pieces. These are metaphors that serve to help the player grasp the rules, and we communicate them to her with images, sound, animation, and feedback. If we tell the player the Megablaster needs to cool down after discharging a mega-hot laser, we’re selling her a justification for the half-second reload time. If the player can see her Megablaster heat up white for a half-second before fading to normal, we’ve made a visual metaphor to reinforce the rule.

The more cohesive a game’s context is—the more things behave according to the metaphors we’ve assigned them—the more easily the player can build expectations and anticipate and understand the rules of the game.

In the sub-Venus darkness, Janet is stepping through the blasted remains of robots, keeping her eyes peeled for human hostages. We, the game’s creator, have decided that to avoid introducing a new, underdeveloped verb, we want the player to rescue hostages using her “shoot” verb—as an extension of a verb she’s already familiar with.

When Janet has a hostage in her sights, staring down the barrel of her Megablaster at a harrowed human captive, will she be able to pull the trigger? Is the player likely to shoot someone she’s been tasked with rescuing, now that she’s observed how shooting robots wrecks them? Most likely, she’ll hesitate, confused.

Maybe once she’s done it once, and she understands that shooting a hostage teleports the hostage to safety instead of splattering her on the cave walls, she’ll be able to shoot hostages. But that initial doubt is a serious hurdle to the player’s process of learning the game. This is the
first time she's encountered this rule, and as creators, we want her to grasp the rule with as little prodding as possible. If we need a commanding Sargent Tutorial to radio Janet and instruct her that shooting hostages tags them for teleportation, we've failed to communicate the rule. Resist Sargent Tutorial.

Shooting a hostage with a deadly weapon to rescue her introduces a dissonance between the player's expectations and the rules of the game that will haunt the rest of her experience and make her doubt her understanding of the game.

What if, instead of shooting the hostage to rescue her, the hostage is in a cage—a robot cage, made of and obviously resembling the same metal as the robots Janet's been blasting. When the player shoots the hostage now, she's blowing open the cage and freeing the hostage. Freed from the robot cage's teleport dampening field, the hostage can now beam to safety. Thank you, Janet! My hero!

This is a lot easier to buy than having to shoot a hostage with a deadly weapon. If the player sees something she can identity visually as a hostage trapped in a cage, she can put two and two together—the other “two” being her knowledge that her Megablast destroys robot-looking things—and figure out that maybe she should shoot the cage. When she does, the game reacts by communicating that she's freed the hostage. Good job!

The easier we make the rules of our game to understand, the more easily and effectively the player can internalize those rules and begin anticipating them.

## Objects

I'm not referring to “objects” in the simple sense—nouns, detritus, inanimate objects. I mean objects for our verbs: the objects that complete their sentences. “Jane digs through a block of green clay.” “Jane tries to dig through a metal block but is repelled.” We will draw from our palette of objects to set up choices for our player and to tell a story.

The right selection of objects goes a long way. The downward path that Jane navigates in *Tombed* is made up of those four objects: clay blocks in three different colors that connect to blocks of the same color, and metal blocks. As with verbs, we get a lot of utility over choosing the right ones and should try to avoid introducing ones that will be underdeveloped because their uses overlap too much with that of other objects.

One color of clay blocks would be too little. I could use them as pacing, but not to set up interesting choices about where to dig. Two would let me set up choices, but not ones that link into each other very well. Because there are only two colors, two diverging paths can’t touch. Four colors—at least in the limited space that I've given the game, for the sake of it being fast—doesn’t give me very many options that I can’t achieve with the three colors I already have. See Figure 2.6 for an example.
Figure 2.6  Possibilities in Tombed with one, two, three or four colors. Note how three colors adds way more choices than four.

Did you ever do that exercise where you try to color in a map of a bunch of different regions without having any regions of the same color touch, using as few colors as possible? Four, it turns out, is the maximum amount that we would need. (This is known as “four-color theorem.”) But if you look at any completed map—for example, a map of the United States, using four colors—you’ll notice the fourth color appears infrequently, only in very hairy combinatorial situations. A fourth color in this game would be underused. It would mostly make the visual pattern of the game more complex and potentially confusing, forcing the player to spread her understanding of the game rules that much thinner.

Tombed was made for a two-day game design competition—Ludum Dare 14—so I released its source along with the finished game. (It was made in Game Maker 8.) Using that source, my friend Leon Arnott created a spin-off/sequel called Tombed II: Twombed Off. In his sequel, he adds only a single new diggable object to the four that are already there: a red “bomb” that, when struck with Jane’s shovel, destroys every piece touching it, regardless of its color, even if it’s an undiggable metal block.

This is a good addition because it adds a lot to the game. It gives the player the ability to destroy metal blocks, but only when given the opportunity. And it has a relationship with the other blocks: because it destroys everything connected to it, sometimes the player wants to dig through other blocks to sever connections with a bomb block. This is strengthened by another new rule Leon added: now the bottom of the screen is also lined with spikes, so the player wants to pace her descent carefully. Bomb blocks potentially wipe out a lot of terrain at once, which sometimes means Jane falls into the spikes before ground appears to catch her (see Figure 2.7).
The bomb block adds the opportunity for a lot of new choices—choices that aren’t possible with the existing blocks because they only interact with similar-colored blocks, not different-colored ones. But it still fits with the existing objects because it’s about digging and caring about which blocks are adjacent to the block Jane’s digging.

It’s easy to add game objects that have tenuous relationships to existing objects because they’re quick fixes, sloppy patches. For instance, if I wanted Jane to run over here before having to quickly run back over there, the most dumb-simple way to do that would be to put a switch here that, when Jane strikes it, opens a door there. (See the left side of Figure 2.8, with the switch and door in red.) But that doesn’t have anything to do with digging—with any of the rules we’ve been teaching the player. And the fact is that we can accomplish the same thing using the objects we already have and design, instead of having to introduce a new, underdeveloped rule. The better way to create a “switch and door,” using the same objects and verbs that Tombed is already based on, is shown on the right side of Figure 2.8. (Keep in mind that Jane is two blocks wide, so she can step over gaps that are only one block wide.)
The Physical Layer

Digital games, even if we watch their rules play out on a TV screen, have a physical layer. Jane may be digging through blocks of clay, but what the player is doing is tapping the Shift key. The verbs the player is using to interact with the other rules of the game—to make Jane do stuff—are connected to physical actions the player is taking, though they’re small ones.

Our goal as creators is to communicate the rules of the game as clearly and succinctly as possible, yes? This is why it’s important to make the connection between the verb the player is acting upon and the physical action the player is taking to do it as close as possible. I don’t mean that digging in Tombed should be physically grueling for the player, because that’s a game of quick thinking, not endurance.

Grueling controls might suit a different game. Take the original arcade version of Track & Field (1983). Running in that game involves alternately mashing two different buttons. This helps characterize the action of running because you can imagine two feet stepping in sequence: right leg, left leg, right, left. It’s also physically arduous, which increases the player’s connection to the athlete on the screen and the actions he’s taking. In a game where running is less important, it’s safe for the mechanics of running to be abstracted. You can walk from one side of the room to the other without having to focus on the motion of your legs. But for a game about competitive distance running, this connection is important.

Of course, if you’ve played many videogames, you know that we don’t need to make a verb’s physical performance resemble the verb. There’s not much about pressing a button that resembles jumping, and that’s part of the interesting quality of videogames—we can explore the distance between the physical act of using the interface and what it represents inside the game. Sometimes we can close the distance. There are platforms that can sense a broader range of human motions and allow us to force the player to perform a scooping motion to dig through a floor. But even if we explore a wider distance between physical interaction and game representation—if we’ve restricted the input to a single button, there’s a range of physical action we can connect to our game verb.

The player makes Jane dig in Tombed by tapping a key. How does Jane dig? A single, fast tap, a poke of the earth with her shovel. A quick action that mirrors the player’s key-press. What if the player tapped the button and Jane slowly dug her shovel into the earth, pressing it down with her foot, and then threw the dirt over her shoulder? There’d be a dissonance there that would make the player feel less of a connection to the verb she was performing.

That’s not to say you would never make a game where digging was slow and time-consuming: digging is a slow task. But if you designed a “dig” verb that functioned like that, you would also want to design a physical layer to the verb that would better describe it. Maybe the player has to hold the button down while Jane is pressing the shovel down into the earth. Maybe she throws the dirt over her shoulder when the player releases the button (see Figure 2.9).
In fact, the latter is especially compelling because it’s dense: we’ve mapped two parts of a verb to a single button. We’ve mapped the different steps of a sequence—jam the shovel in, and then lift it over your shoulder—to two different physical actions that can be performed with the same button: holding it down and letting it go. By compressing as many verbs as possible into as small a physical layer as possible—in this case, as few buttons as possible—we’re potentially avoiding a lot of player confusion and fumbliness.

When I play a game that uses three or four different, adjacent keyboard keys, my fingers have a hard time remembering which one is which, resulting in a lot of mispresses—which usually means spending crucial resources by accident or misperforming an important choice. It means a player who feels cheated. The only exception to this multikey problem is with a keyboard’s arrow keys, which, like the multidirectional joypads of game controllers, have been designed and arranged with a physical metaphor in mind already! It’s no wonder so many games use the arrow keys as controls—it’s a handy shortcut.

Some “fighting games,” like the Street Fighter series, map many overlapping verbs onto many buttons—traditionally, six buttons and an eight-directional joystick. Here, learning to physically perform verbs, and being able to perform them successfully when the time is right, is an important part of the game. The discipline needed to learn and perform verbs is here intended as a parallel to the discipline that learning and performing a martial art requires.

Reinforce a verb’s physical layer wherever possible. If a game has a title screen and menu, try to connect the important verbs and input to it. In Tombed, I’ve mentioned, the player starts by making Jane dig through the floor of the title screen using the same button she’ll use to dig in the rest of the game. At the very least, allow the player to navigate the menu with the same input that she uses for the rest of the game. There are a number of Flash games where the player’s verbs are tied to keyboard keys—say, the arrow keys and the spacebar—but the menus ask the player to point and click with the mouse.

Imagine if, after every level of the game, the player has to move her hands off the important keys, take her mouse, point at and click the Next Level button, and then return her fingers to
the arrow keys and spacebar. That’s tedious, and what’s more, it weakens the player’s focus on the game’s verbs and the keys that correspond to them. Games that successfully use both the keyboard and the mouse usually assume that the player has only one hand on a limited set of keyboard keys and the other hand on the mouse, to avoid this kind of fumbling and switching of the hand positions.

I recently discovered an iPad game called *Mini Mix Mayhem* (2012), the premise of which is that the player or players have to manage up to four games, with different rules and goals, that are sharing the screen at the same time. There are around twenty games in all that can show up in the four you have to juggle at once, each controlled in a different way that suits the game’s visual metaphor. Unwinding a nut from a screw involves physically flicking the nut to the left; you think of actually using your finger to spin a nut loose. Solving a which-cup-is-it-under puzzle requires just tapping the right cup; you think of pointing at the cup for the barker, who then reveals what’s under it. Collecting drops of rain in a cup involves holding your finger on the cup and dragging it left and right; you think of holding a cup and moving it around.

Many of us will never design for a touchscreen, but regardless of what physical actions we’re giving the player to perform the verbs in our games, we can design actions that call to mind the properties of our verbs—that communicate and reinforce the rules of our game.

**Degrees of Control**

Different kinds of input allow for different paces of input. Mouse input, for example, is highly nuanced. It allows for slow, gentle movements, or fast, sweeping movements, along two axes simultaneously. (A player can move a mouse *up* and to the *left*, for example, at rates that are independent of each other.) Often, games use mouse motion for verbs like looking around: your eyes pan slowly along the sprawl of the mysterious city in the distance. What secrets does it hold? Suddenly in the corner of your vision you spot a bird made of skulls and swing your eyes quickly after it.

The mouse offers many more degrees of movement than, say, a keyboard key that is simply in a *pressed* or *not pressed* position. But binary keys have their moments, too. Take a game like *Tetris* (1986). The goal of *Tetris* is to position shapes, each made of four square blocks, such that they fill the horizontal space of the playing area with no holes. As such, movement in this game—the player moves one piece at a time—is in the length of a single block. Movement happens on a grid. If it didn’t, it would be hard to fit new pieces into the holes between existing pieces. Binary keys make sense here: one tap of a key can correspond to one block of movement.

You could design a way to control *Tetris* with a mouse—maybe the shape would always snap to the closest grid position to the player’s horizontal mouse position. But what about the mouse’s vertical axis? In this example, it has nothing to do (see Figure 2.10). *Tetris* wasn’t designed with a mouse in mind.
A game that was designed for a mouse instead of grafted to one might look very different from a grid-based game like *Tetris*. As an example, take Christophe Andreani’s 1987 Atari ST game *Bubble Ghost*. Andreani made the protagonist of his game a ghost—insubstantial, it can travel anywhere a mouse cursor can without worry of colliding on walls or obstacles. However, the ghost is trying to maneuver a fragile bubble (by blowing on it) through a mansion full of things that, while harmless to the ghost, will pop the bubble on contact. So you can see that this game was designed with the mouse foremost in mind.

(And interestingly, *Bubble Ghost* was itself grafted back onto a four-way d-pad in a 1990 version for the Game Boy. Unsurprisingly, it loses something: allowing the player to move at a fixed speed means that lining up and aiming is both easier and less nuanced.)

As a creator, you should always consider the properties of any method of input you’re designing for, even if, for whatever reason, the choice isn’t yours. I’ve played a million attempts at *Super Mario Bros.*-style games on the iPad touchscreen that try to reproduce the same controls as the game they’re imitating: left and right movement “buttons” on the left, and two “action” buttons on the right. There’s no textural distinction between the two action “buttons,” as there would be if they were two keys, so it’s hard to hit “the left action button” or “the right action button” with any degree of consistency. And any motion that moves the player’s fingers too far out of the corners obscures the screen—the information the player is relying on.

The most effective games of this kind pare down the player’s verb set until only two binary inputs are needed: either the player’s left thumb is on the screen or it isn’t, and either the
player’s right thumb is on the screen or it isn’t. In 1-Bit Ninja (2012), the protagonist moves forward when the left side of the screen is touched and jumps when the right side of the screen is touched (see Figure 2.11). The player doesn’t use the touchscreen to choose the protagonist’s direction of horizontal motion: rather, there are in-game objects that flip the protagonist around if she touches them. Design for a game’s means of input, not against it.

![Figure 2.11](image)

Two different models of touchscreen input, one designed a little more consciously of being on a touchscreen.

Be aware of the properties of a form of input before designing a physical layer for your verb. A computer’s arrow keys and the plus-shaped “directional pad” of a Nintendo controller might seem the same, but there’s a fundamental difference. The keys are four independent binary states, whereas the “d-pad” is a single piece of plastic—it can be depressed left or right and up or down (two axes of movement), but the player can’t depress left and right at the same time. There’s an enforced exclusion in the d-pad that makes sense in games about spatial navigation in two dimensions—after all, what does left and right simultaneously mean in terms of that movement? They cancel each other out.

But other contexts have a use for them. Dance Dance Revolution (DDR, 1998), a Simon Says game that players play with their feet on giant d-pads, occasionally gives players a Left and Right command or an Up and Down command simultaneously. These require the player to jump into a position where their left and right feet are opposite each other. It’s important here that DDR’s giant d-pad is made up of binary buttons rather than a single piece of molded plastic (which is designed for a single thumb, not two feet).

This may seem obvious—it seems impractical that someone would build a physical input device that size out of a single piece of plastic—but the hidden pitfall here is that, used to the properties of a single type of input, you fail to recognize and design for the less obvious properties of another mode of input—like the developers who tried to re-create Nintendo pads on touchscreens.

**Character Development**

In the Robot Mines of Venus, Janet Jumpjet stalks the catacombs, her Megablaster 3000 held in front of her, trained at the unseen hordes of deranged robots no doubt lurking somewhere
in the darkness ahead of her. Then she sees it—the glint of metal up ahead, the unmistakable gleam of a robot! She stops dead in her tracks, finger tensing on the trigger. But the metal form doesn’t move. She takes a tentative step forward and sees that it’s not a robot—it’s another robot cage! Has she found another prisoner? Peering through the viewscreen in the cage, Janet sees the face not of a human prisoner, but a robot in storage! This is a cage Janet wisely decided to leave unopened.

Suddenly a robot, taking advantage of Janet’s momentary distraction, lunges out of the darkness. Taken by surprise, she fires wildly—blasting open the cage and freeing its robot inhabitant, who immediately blinks to life and begins clambering toward Janet, alongside its compatriot. Cursing, Janet blasts the robots to smithereens. She wonders if she’ll be seeing more of these robot containers and reminds herself that she’ll have to be way more careful with her shots.

Not long after, Janet emerges into a room patrolled by two robots. Janet raises her Megablaster, but the hairs on the back of her neck are trying to tell her something. Taking a breath, she looks around. The walls of the room are covered in caged robots, waiting to be released (see Figure 2.12).

She’d better aim carefully, all right.

![Figure 2.12](image.png)  
*Figure 2.12* Janet had better think before she shoots in this room.
Janet is the main character of the narrative that unfolds in this game, but don’t forget: the main character of the game itself is still the verb “shoot”—a verb the player has to understand how to use wisely. If verbs are the main characters of our game stories, we develop them as we would characters in any other form: we challenge them, we give them new responsibilities or burdens, we let them show new sides of themselves, we let them grow—or force them to.

Our rules and our objects are the vocabulary we use to develop our verbs. Again, we want to avoid introducing so many rules or objects that some of them won’t be allowed to develop fully—that they won’t see enough use. We also want to be thrifty because all these new rules are things that the player has to learn and keep in her head. Every new rule takes a little of the player’s attention from every existing rule. The more rules, the more conditions, the thinner the player’s understanding of the game is spread. The thinner her understanding, the more she’ll forget things, become confused, and fail to predict what some new object will do or how it will combine with existing objects.

We can get a lot of development of our verbs out of the objects we have if we use them well. On Venus, we introduced an object that functions like the earlier “robot cages” but releases a new robot when shot. We can get a lot of interesting situations out of this object alone. There’s the room where the walls are covered in caged robots: the player fights mobile robots, but a missed shot means more robots can potentially escape. Or imagine a tunnel, the exit of which is blocked by a caged robot. To get through, the player has to free, and then destroy, the robot. Maybe there are robots chasing from the other side of the tunnel!

The “caged robot” object is keeping with our existing understanding of the rules: robots chase Janet, and cages release what’s inside them when shot. And the preceding scenarios develop Janet’s aim and shoot verbs. The cage room forces the player to shoot more accurately; the tunnel gives her a better understanding of the Megablaster’s rate of fire: a half-second, remember? She chooses when to let out the robot at one end of the tunnel; she has to time it so she won’t be overwhelmed by enemies from both sides.

These sorts of arrangements are what we call “level design,” and we’ll talk more about that in Chapter 3, “Scenes.” It’s important to develop the skill of recognizing objects that will support a verb rather than merely add clutter to a game. It’s also good to choose ones that have a lot of utility—ones that can be developed themselves, rather than jumping on the stage with a shout and then standing there awkwardly, having nothing more to say.

Elegance

We’ve been discussing an approach to elegant design through a careful cultivation of verbs, objects, and rules: the idea being that rules that are too specific or narrow to add much to the game will nonetheless occupy a portion of the player’s headspace and thus dilute her focus and understanding of the game. We’ve been discussing the value of being concise.
There are some traditions of games, though, that emphasize having as vast a population of verbs as possible. These kinds of games have their root in text adventure games—games where the player engages with the rules of the game by typing in commands at a text prompt. These games are in turn inspired by role-playing games between human players, where a player in a “Game Master” role narrates an adventure for the other players and responds to the choices they make.

Most digital adventure games don’t have a human overseer—the computer responds to the player’s input based on a list of permissible verbs defined earlier by the human creator. In practice, this means that a vocabulary of frequently available verbs develop—“north,” “south,” “get,” “examine.” The challenge to the player comes in recognizing those moments in which an unusual verb is called for, and figuring out which of those verbs is appropriate.

Many graphical adventure games, inspired by those text adventures, did away with the text prompt but still challenge the player by hiding the solution to any given obstacle in a forest of possible verbs. 1990’s Secret of Monkey Island, for example, gives the player twelve verbs at outset—“open,” “close,” “push”—and then gives her steadily more as the game goes on. (See Figure 2.13 for an example.) The protagonist accumulates objects—a mug, a banana, an inkwell—that unlock even more verbs. You can “drink” from the mug, “eat” the banana, “use” the inkwell, adding to the list of options available to the player. This doesn’t really make the puzzles more challenging as brain-teasers or riddles—it just makes the solution more tedious to find as the player sifts through all the possible options and combinations, trying to intuit which of them might be correct.

Figure 2.13 Too many verbs makes for a lot of guesswork.

The appeal of this system is its potential for invention. In Zork, a 1980 text adventure, for example, there’s a room that’s described as being “very noisy,” which contains a platinum bar. If the player attempts to take the platinum bar like she would any other valuable object in the game—by typing “GET BAR,” maybe—the game just repeats her command: “GET BAR.” Anything the player types is echoed by the acoustics of the room. The solution is to type “ECHO.” That will break the spell, allowing the player to collect the bar.
In a graphical adventure game, some near-useless item the player collects early in the game and allows to pass from her consciousness can have an important role at a critical moment late in the game—a kind of Chekhov’s gun. These moments provide a kind of narrative closure and feel clever—when the player gets the joke.

The problem in these cases is that when the player doesn’t immediately get the joke, the game breaks down, because experimentation is slow. How does the player gain information about the rules of any game? She uses her verbs. In a game that gives the player a hundred verbs, experimentation is a slow and tedious process. The player spends an hour trying each of her verbs on every object in her reach, learning nothing until she finds the correct combination.

Are moments of invention possible in games with a concise vocabulary of verbs? Consider Portal (2007) to be in this same tradition: puzzle-solving games with overt, expository storytelling. The game establishes a concise set of verbs very early—the player can join any two points in space by placing a blue portal and an orange portal. These doorways are considered connected no matter where there are. There’s also a pretty concise cast of objects: walls that accept portals, walls (a darker color) that don’t accept portals, blocks that can travel through portals like the protagonist, switches that are opened by the weight of a player or block, gun turrets that fire at the protagonist when they see her but have the same properties as a block.

Realizing one can position a portal under a turret to remove it from a problematic spot is a moment of inventiveness. So is realizing one can make a portal above a turret—it’s on a piece of dark, portal-resistant floor in this example—through which to drop a block onto it and knock it over. In this case, the player is inventing from her existing knowledge of her verbs and the objects they act on. Here she can experiment, observe, and make connections. But the solutions nevertheless have the potential to be eurekas—real moments of breakthrough development for a verb, like an unexpected “plot twist” where the player has found out something exciting and new about the potential of our main character.

You don’t have to avoid moments where solutions come from unexpected directions—but you shouldn’t build a game out of them. The player’s verbs are the means by which she comes to understand the rules of the game—give her too many verbs that don’t interact with each other, and you give her a weak understanding of the rules of the game.

**Real Talk**

Concurrently with writing this book, I am working on a game with Loren Schmidt. In this game, the player guides a succession of expendable slave miners harvesting precious crystals from the caverns of strange planets for unseen alien overlords. We’re only now beginning to design levels—the caverns that the player will explore. Most of the time we’ve spent on the game thus far has gone toward designing our verbs and choosing objects that reinforce those verbs and each other.
The player digs up the crystals by using radium bombs. She starts every cavern with a limited stock of them, and more can usually be gathered from within the cavern. Pressing the spacebar causes the miner to drop a bomb. After pulsing for a few moments, the bomb goes off, exploding in a small circle that eats through adjacent ground and kills any creature within the radius of the explosion—including the miner. So there's already a tension between planting bombs and giving oneself enough space to avoid them, one that becomes exacerbated by the presence of hostile creatures and other threats.

The player can also dig with her hands. When the player steers the miner into a wall, she starts to slowly carve a path for herself through it. This is slow and inefficient but serves a couple purposes. First, because the walls the player's blowing up are made of tiny grains, and the player plants her bombs wherever she chooses, it's possible to leave thin shells and obstacles between otherwise-open spaces. These really aren't worth eating up an extra stock of the player's bombs, so the player can “clear out” that debris simply by digging through it by hand.

The other purpose is tactical bomb conservation. In some circumstances, with few bombs remaining, she might be willing to trade time for the ability to hang on to one of her precious bombs. The player has a limited amount of oxygen, represented by a meter at the top of the screen, that slowly drains, so trading time for bombs can be a critical choice. There are also situations in which the player runs out of bombs and is forced to rely on digging. If digging by hand wasn't an option, the player might be left with a half-completed level she has no means to finish, which is a situation we want to avoid. Figure 2.14 illustrates bombing versus digging.

When to plant a bomb and when not to is an important choice. “Bomb” is that player’s primary verb and means of interacting with the game world. So we wanted a cast of creatures that interact with that verb—that are affected by, and complicate, the player’s decision to drop a bomb. We also wanted them to interact with each other, so that they form relationships.

Here are some of the inhabitants of our alien caverns. First, there is a simple, bouncing creature that moves at 45-degree angles (diagonally), slightly slower than the player, dumbly bouncing off any wall that it touches. Because its movements are predictable, it’s easy to avoid, except when it’s in great number. We can create situations in which the player wants to be careful

Figure 2.14 Choosing between bombing and digging.
where she bombs, for fear of letting a bouncing creature out. There’s also a rocklike creature that doesn’t move, remaining dormant, until the ground underneath it is removed. Then it falls, killing anything it touches. So the player can set off this rocklike creature, sometimes to her detriment, sometimes to her benefit. Set off at the right time, the player can use this creature as a means of clearing out a tunnel full of bouncing enemies, for example.

Another inhabitant is a gun turret that, when the player is nearby, awakens and shoots bullets toward the player. These bullets are lethal to the player, but also to other creatures, like the bouncing things. They also chip away at the ground where they hit, so a resourceful player can sometimes use them to dig paths in lieu of bombs. Or a player can set off one of the falling rock monsters by removing the dirt that’s keeping it in place. See Figure 2.15 for an example of all these objects interacting.

![Figure 2.15](image)

We wanted to ensure that even nonintelligent objects in the game react to the player’s bombs. Wild radium—radium that the player can collect to increase her stock of bombs—will itself ignite and explode if touched by one of the player’s explosions. Sometimes the player will want to set off a chain reaction; other times the player will want to be careful to dig out wild radium without setting it off. The crystals the player is trying to collect will crack and shatter after being hit by explosions. The first explosion causes the crystal to crack. This usually happens while using a bomb to unearth the crystal and is a warning not to do it again. The second explosion shatters the crystal. Your explosions have consequences.

All these things reinforce the player’s primary verb: her ability to cause explosions and to dig through the ground with those explosions. And we’ve tried to design objects that allow us to develop that verb in interesting ways: like giving a player the opportunity to rely on an enemy to dig a path for her rather than spending a bomb, for example. We’ve tried to avoid objects that would be cul-de-sacs—that would have a single use and then fail to develop and provide meaningful choices after that.
Review

- Games are made of rules. Verbs—“jump,” “climb,” “shoot,” “place a piece,” “rotate a block”—are the rules that give the player liberty to interact with the other rules of the game.

- We can use verbs to set up choices for the player. Often, to make for interesting choices, we want to give the player several verbs—“move horizontally” and “shoot vertically,” for example. We want these verbs to have a relationship to each other.

- We want to be careful about choosing verbs and relationships that we can develop. If we add too many verbs, not only will the player’s understanding of the game be less focused, but many of those verbs are likely to be left underdeveloped.

- We “develop” a verb the same way we develop a character in a story. We give them more responsibilities, we ask them to perform together more closely, we give them difficult choices to make.

- Verbs have objects—things they act upon—to reinforce them, develop them, and give them choices. As with the verbs themselves, we want to avoid introducing objects that will be underdeveloped, and we want to design objects with relationships to each other.

- We want objects to have relationships with the player’s important verbs. Verbs are what the player uses to gain information about the other rules of the game, so whenever the player uses a verb in a way that she expects a reaction from, she should get one. We want our verbs to be robust.

- Every rule has a context that helps the player relate to it, to understand it. This context can be reinforced by the way the game looks and sounds: something that can hurt the protagonist is covered in spikes, something that we want to direct the player toward is valuable-looking.

- A verb has a physical layer: a button the player presses, a gesture with the mouse, a swipe of a touchscreen, the rolling of a trackball. The more closely the physical layer can suggest the verb—in pace and in performance—the stronger the relationship between these two will be.

- We want to reinforce the player’s verbs and their relationships whenever possible. The verb’s physical layer does this, the context we give to it does this, the objects’ reactions to it do this. Think of games in terms of verbs. Think of what relationships those verbs have. Think of what objects can reinforce and complicate those relationships, and how. Verbs are the characters in our game’s story, and when we develop them, we tell our story.

Discussion Activities

1. Pick a game. If possible, do this at random: if you have access to games on physical media, jumble them in a pile and grab one without looking. Play this game for no more than five minutes, not counting publisher logos and title menu screens and too-long opening movies. When your five minutes are up, stop.
Get a sheet of paper. Start writing down the player’s (or players’) verbs—as many as you noticed. Diagram them, especially the ones that play the most significant roles in the game. Draw connections between the ones that interact and describe their relationships. (For example, the relationship between “move left and right” and “shoot” is “aiming.”)

2. Design an object for Janet Jumpjet’s adventure in Venus’s robot mines. It should develop or complicate Janet’s verbs in an interesting way and should relate to at least one of the existing objects in the game: the robots, the caged human hostages, or the caged robots.

You’re encouraged to imagine Janet’s game any way you like for this exercise. I’ve left some details deliberately ambiguous, like the perspective of the game and the physical layer to Janet’s verbs.

3. Choose one of the following verbs and design a physical layer for it, using one or—at most—two binary keys. (Like keys on a keyboard, the game reads them either as depressed or unpressed.)

- Scooping water from a sinking boat, with a bucket
- Playing catch with a dog
- Hammering a nail into a wall
- Unwrapping a gift-wrapped present
- Firing a slingshot

Design a physical layer that expresses the verb as neatly as possible. You may wish to break a verb into several verbs or actions. Remember to make the physical layer as dense as possible. Design physical layers that allow for appropriate degrees of input where it makes sense.

Group Activity

Discuss the verbs you’ve used when playing your favorite games or the games you’ve played recently. Are these verbs that you find in a lot of games, like “jump” or “shoot”? Can you think of games that have used these verbs in unusual or interesting ways?

Now think of a verb that you often use in regular life or a verb that you’re using right now. Maybe you’re “sitting” in a chair or “writing” notes on a piece of paper. What kind of game would result if you decided to use an ordinary verb as the basis for a system?

Using one of the verbs that you just discussed, come up with an idea for a game that develops this verb. This could involve special objects that help develop the verb, such as an object that the player can jump on to change the direction of gravity or the entire view of the game world, or multiple verbs in conjunction with each other, such as a gun that changes objects into jumping platforms. Talk about what kind of gameplay might result from these combinations of verbs and objects.