CRC PRESS ▪ TAYLOR & FRANCIS

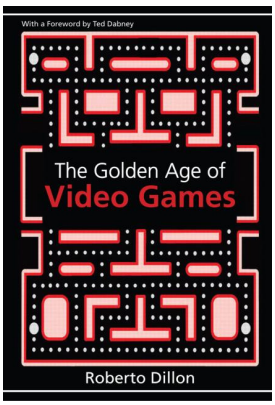
Game History

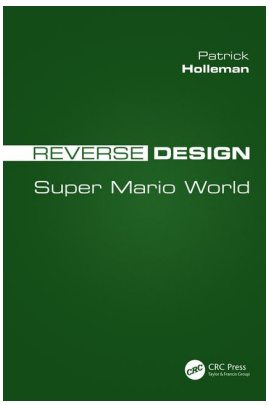
A Chapter Sampler

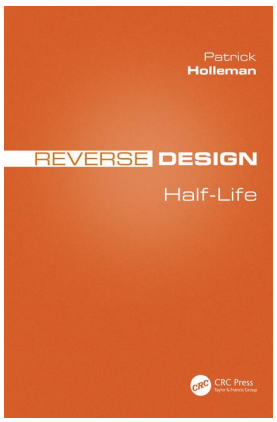
CRC Press
Taylor & Francis Group

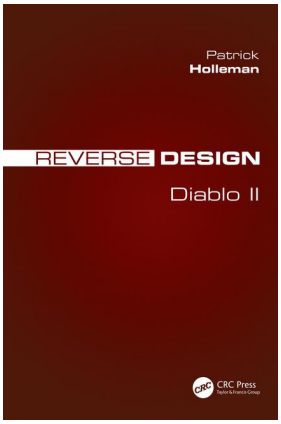
www.crcpress.com/games-animation

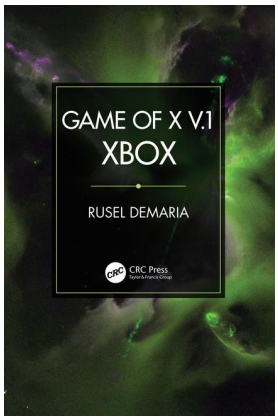
Contents

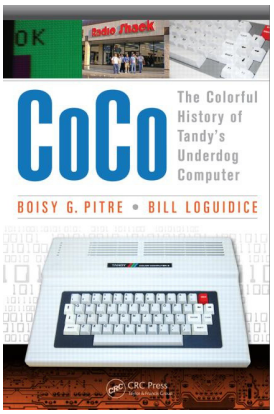








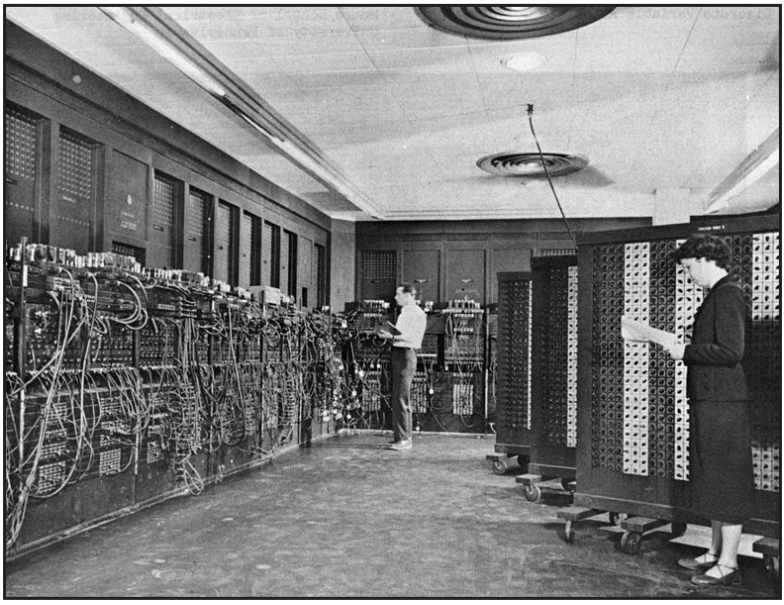




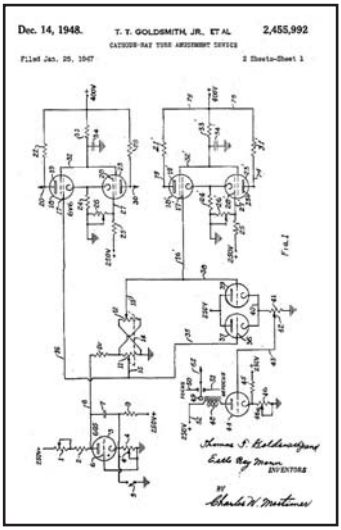
The Beginning

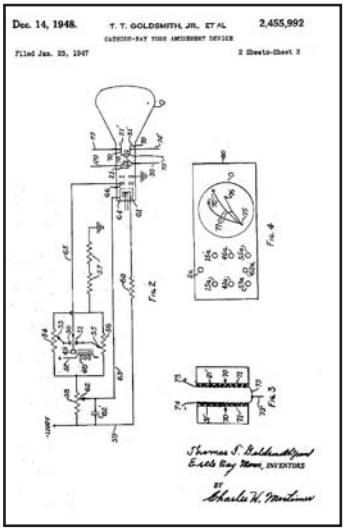
The first general-purpose electronic computer is considered to be the Electronic and Numerical Integrator and Computer (ENIAC), developed secretly in the United States during World War II to compute artillery firing tables and manipulate complex data related to the hydrogen bomb. The huge system was made public in 1946 and, obviously, no one even thought of using it to play games.

Nonetheless, the time of video and computer games was approaching fast and, in 1947, the first known electronic game played on a screen was developed. The cathode ray tube (CRT) was a technology perfected in the thirties, and, by the mid-forties, it was the main building block in all TV systems. It shouldn't be surprising then that the first game using this technology, named the Cathode Ray Tube Amusement Device, was developed at DuMont Laboratories, a well-known television manufacturer.



The ENIAC at work.





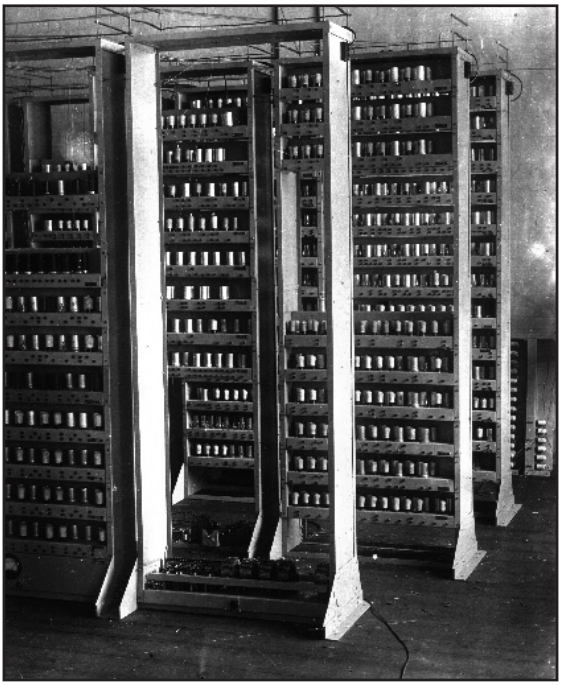
First two pages of the patent filed for the CRT Amusement Device, the first electronic game ever to be played on a screen.

The main author of the project was Dr. Thomas Goldsmith Jr. (1910–2009) who, together with Estle Ray Man, devised a clever system of manipulating the electron beam by controlling a set of variable resistors to simulate a simple missile shooting game. Only a dot was displayed on the screen but different targets were graphically represented by transparent layers applied on the CRT's screen to simulate a radar-like environment. The project was successfully filed in a patent but, regrettably, DuMont didn't pursue this research further.

In 1950, the first computer game was designed by famous mathematician and engineer Claude Shannon (1916–2001) in the seminal paper "Programming a Computer for Playing Chess."[1] Unfortunately, though, no computers at the time were powerful enough to implement a chess game so his design was not transformed into an actual program.

The first actual game programmed on a computer was in 1952 when Ph.D. candidate Alexander Douglas (born 1921) from the University of Cambridge, UK, developed a tic-tac-toe game named *Noughts and Crosses*. His game, also called *OXO*, was a part of his thesis on human and computer interaction. The game ran on the Electronic Delay Storage Automatic Calculator (EDSAC), which was the first machine programmable by means of assembly instructions, as opposed to turning switches or connecting cables as required on the ENIAC or earlier computers.

[1] Published in *Philosophical Magazine*, 7th series, Vol. 41, No. 314 (March 1950): pp. 256–275.

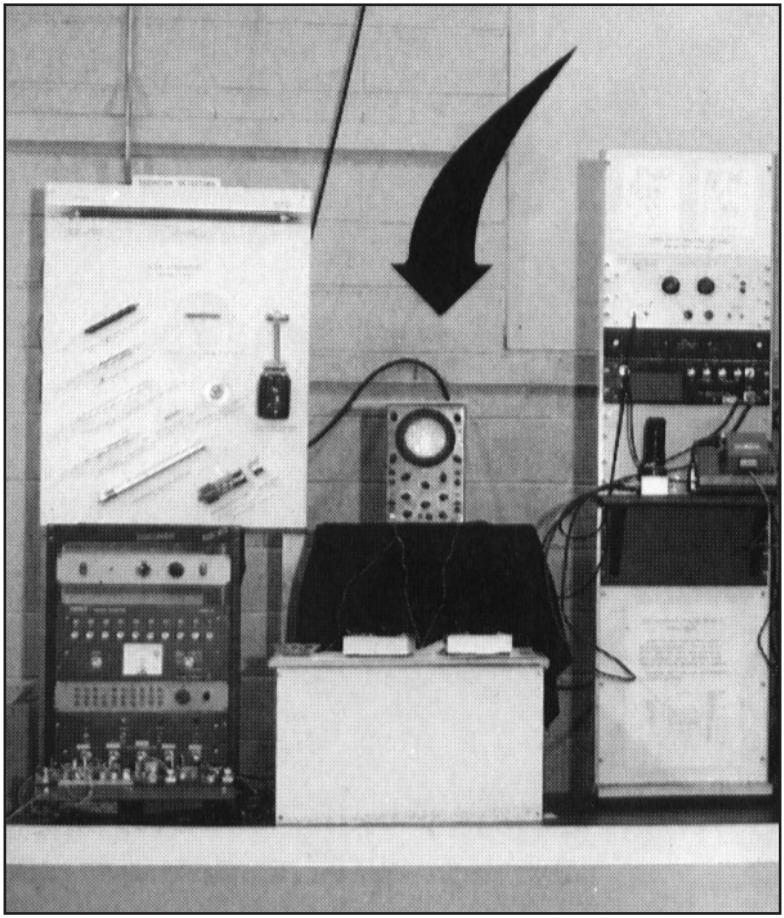


The vacuum tube-based EDSAC, the first computer to ever run a game.

Playing *OXO* involved inputting moves into a rotating phone dial and the nine-cell board could be viewed through a tiny 35 × 16 pixel display. However, the EDSAC was a one-of-a-kind machine so the game never had the chance to be appreciated outside of Cambridge University.

There were a few more artificial intelligence (AI) research projects within academia that related to board and mathematical games such as checkers and nim, but let's shift our attention again to the US East Coast where the first two-player sport-inspired video game, designed for mere entertainment, was born: *Tennis for Two*.

In 1958, William Higinbotham (1910–1994), a physicist working at the Brookhaven National Laboratories in New York, decided to "spice up" the annual open house event by giving the visitors a game to play. He did this by using an analog system able to compute missile trajectories and applied it to simulate a much friendlier tennis ball. The computer displayed the game by using a small oscilloscope, and the game could be played by using two controllers featuring a button to change the ball direction and a knob to affect the rebounding angle (see the figure on the opposite page). These controllers were provided to visitors who wanted to try the new technological marvel. Hundreds of people queued up to play the game that was showcased again a year later through a bigger oscilloscope.



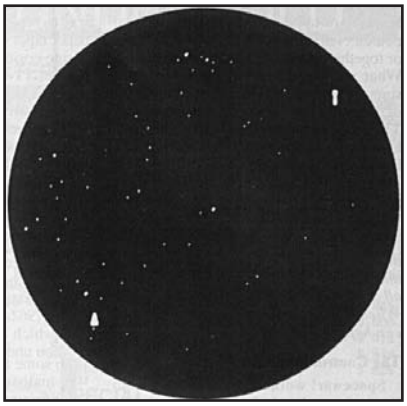
The computer running *Tennis for Two*. The arrow points at the oscilloscope showing the ball being bounced around by the players thanks to the provided controllers (the two white boxes on the table right below the oscilloscope).

Like *OXO* before it, though, *Tennis for Two* was created in a laboratory, restricting its use to that particular environment where it was produced and making it difficult for the game to become widely known and enjoyed.

The first time a video game was finally able to attract people's attention on a daily basis was in 1962 at the prestigious Massachusetts Institute of Technology (MIT) in Cambridge, Massachusetts. This video game was played using a state-of-the-art computer that the electrical engineering department of the university received in 1961. This computer was called the Programmed Data Processor 1, or PDP-1. It was a mainframe sporting 9 KB of memory and a monitor (see the figure on the top of page 6).

The PDP-1 by Digital Equipment Corporation (DEC). It used punched paper tapes as its primary storage medium and instructions had to be input through a tape reader.

Among those allowed to access the machine were a group of students from the Tech Model Railroad Club (TMRC). The TMRC loved experimenting with new technologies, so they were very excited by the newly arrived "toy" and decided to build a test application to push it to the limits. This small group of friends, led by Stephen Russell (born 1931), developed *Spacewar!*, a combat-style game in which two players had to face each other in a space shootout while avoiding the gravity well of a star placed in the middle of the screen.



A close up on the PDP-1 monitor running *Spacewar!*.

A *Spacewar!* match in action.

In the following years, different versions of the game were developed, featuring improvements such as better physics, real star constellations as a backdrop, and a pair of button-based remote controls, made by team members Alan Kotok and Bob Sanders. Thanks to these improvements, *Spacewar!* quickly became so well known that DEC started shipping it with the machines as a demo/test program. It was so popular, in fact, that when a US university purchased a PDP (for about $120,000), it was the first application that students loaded and played when the computer was free from more serious and academically-inclined duties.

Overall, 50 PDP-1 units were produced and the last working unit is still showcased at the Computer History Museum[2] where visitors can admire *Spacewar!* in all of its original glory.

[2] For more information, see http://pdp-1.computerhistory.org/.

2

Measurements and Their Meanings

This chapter will explain how I, along with a colleague, measured *Super Mario World* and what conclusions I drew from those measurements. Everything that can be said about the smallest level of *Super Mario World*'s design comes down to one question: "How difficult is this jump?" Obviously, it's a question that I asked thousands of times. Players don't always consciously ask themselves this question, but it's definitely in the back of their minds, especially on the really tricky jumps. Players have to predict the motion of Mario (or Luigi), the motion of their target, the distance they have to cross, any enemies they need to avoid while in flight, and numerous other factors that affect the difficulty of each individual action. Experienced players can do this in an instant, so it's clear that these elements form a very cohesive whole. The problem for us as designers is figuring out how numerically hard a jump is. Can we unpack the intuitive understanding of this game that so many players have? Can we really break down *Super Mario World* into its constituent parts?

Fortunately, the answer is yes. In fact, *Super Mario World* breaks down quite nicely into its component design elements. From jumps, I was able to deduce challenges, and from challenges, I was able to realize the presence of cadences, and cadences fit together into skill themes. Going back to the beginning, my first task

was to figure out ways to meaningfully measure everything in the game. Once everything was measured, things like cadences and skill themes came together quite quickly. This section tries to assemble the same line of thinking and process of discovery that I experienced while researching. The conclusions about these measurements are in Chapter 3, but that chapter and the definitions in it will not make much sense without reading this one first.

Event Height, Event Width, and D-Distance

The first and most important key to understanding the design of *Super Mario World* is the coin-block.

This block is the basic unit of measure in *Super Mario World*—the "atom" of the game. The hardware of the Super Nintendo required that images in the game conform to certain pixel sizes to optimize the file size of assets. Thus, there are no playable surfaces anywhere in the game that are not made up of an integer block length. There are no heights that are not made up of a whole number of blocks. Some moving platforms pass through heights that are not evenly divisible by blocks, but even these come to rest at block-level. Some objects appear to sit in spaces not evenly divisible by a coin-block, too, but their "hit box" is almost always at an effective integer mark. Therefore, every jump event in the game, even underwater events, can be measured precisely.

Mario's ability to jump to any given height is dependent upon the amount of momentum he has. With no momentum, Mario can jump on top of a platform four blocks above the position of his feet. With one block of lateral momentum he can jump to a platform five blocks above the level of his feet. With eleven blocks of momentum, he can jump as high as the top of the sixth block above his feet.

If he has the cape, 11 blocks of momentum will enable Mario to soar high enough that measuring that height is pointless. You'll notice, of course, that these momentum levels correspond to block-lengths. The 11-block momentum benchmark is important because it frequently prevents Mario from using the soaring cape ability to bypass levels (or sections of levels) that lack platforms of sufficient length. The designers of *Super Mario World* made level design easier on themselves by doing this; as long as they restricted platforms to lengths of less than 10, they could exert some control on how the player would approach a level.

There's also the issue of how far Mario has to go *laterally* when jumping. Most jumps involve a lateral distance, which also breaks down nicely into block-lengths. At no momentum, Mario can jump from block one to block six, a distance of five blocks.

At any level of momentum before a full run, he can jump from block one to block nine (distance of eight), and at a full run he can cover 12 blocks. All of this, of course, assumes a flat area; any vertical distance involved in the jump will change the figures significantly. None of this takes into account the use of the cape powerup. The cape extends the fall time on a typical jump to about three times its normal length. It doesn't significantly affect the speed of the lateral motion (what we'll call the x vector), but by allowing the jump's vertical motion (y vector) to last longer, jumps can go farther before Mario hits the ground again. Of course, the player can use the cape while at a full run to go so far across the level that measuring is pointless; only a wall, ceiling, or enemy will stop him.

There are three important statistics that come out of these measurements: delta height, delta width, and d-distance. (Delta means the change between two or more points of data.) The most obvious place to start measuring jumps is the *danger distance*, or *d-distance* for short. This statistic is probably the most important one in this book. The d-distance is the amount of lateral distance that

Mario has to cover in a single jump event. In the screens below, you can see how I measure this.

You'll notice I don't measure diagonally. Because of the way that vectors, intercepts, and powerups in *Super Mario World* work, diagonals aren't any more meaningful than the mere lateral distance unless there's another element like an intercept present, and then it's the intercept that's really meaningful. D-distance measures the size of deadly obstacles Mario is trying to avoid, whether it's a bottomless pit or some kind of damage floor.

The next most obvious point of study is changes in height during a jump event (delta height). In the jump pictured here, Mario is ascending. A number alone doesn't explain if the jump is difficult, however, so let's break it down further. This jump goes up; jumps that go up are more challenging than jumps that go down. The reason for this is that Mario loses his momentum when he's going up (unless he's flying with the cape or Blue Yoshi), but when going down his momentum tends to stay the same and can be easily controlled with a cape-glide. Anyone who has played the game will instantly know that the right hand jump pictured below is easier—even with a larger d-distance—because it is descending. When Mario begins a descending event, he usually has a lot more time to get into the right x-position. The y-position will take care of itself because of the pull of in-game gravity. Mario is already falling the whole time, so the player doesn't really have to worry about that—especially if he has the cape, which will greatly extend the fall and give the player a ton of time to guide Mario downwards. There are exceptions to the rule, but it's almost always harder to jump upwards than downwards.

The trickiest measurement we have to make is the width metric. First, we should define what we mean by width. The starting width of a jump event is the amount of horizontal space that the player has available to begin a jump event. Starting platform width matters mostly because it determines how much momentum Mario can accumulate before jumping. Starting width is pretty easy to measure, but landing width is often complicated by factors other than the size of the platform.

There's an enemy approaching that will damage Mario if the player doesn't avoid it. Can the player simply drop Mario onto the head of the oncoming Koopa? Maybe, but the problem is that when this jump begins, the Koopa is offscreen, and so by the time the player sees it, Mario might not have enough height to bounce of its head. Therefore, the real target/landing width of this jump is four blocks because that's the width available for landing between the edge and the enemy. Most of the time, the starting and target widths are not so complicated, but it is an issue that the player has to confront from time to time.

Just as it is more difficult to perform a jump that ends higher than it began, it is more difficult to perform a jump that ends on a platform smaller than the one Mario jumps off. The width-specific difficulty of a jump is usually determined by the size of the landing platform. It's not especially hard to start a jump on any platform wider than one block, but it's definitely harder to land on a narrow target because of Mario's momentum. The general idea is that the wider the target is, the easier the jump will be. That said, the hardest jumps are the ones that both begin *and* end on very small platforms. We'll get to talking about that momentum in just a moment, but what I found overall is that across the course of a level, the starting platform for a jump event doesn't need to change to make that jump harder. It's the landing platform's width that reveals if there are any really meaningful changes in the sizes of things. That said, starting platform width matters in a few levels that require lots of momentum and don't provide space for it, like Outrageous or Forest of Illusion 4.

There is one last point of concern for the measurement of space, and that is the use of soft sizes. An object with a "soft" size is any object for which the disparity between the graphics and collision box favors the player. That's a little wordy, so let's use an example. The big bullet pictured below is an obvious case: although the animated object is quite large, the part of it that can hurt Mario is smaller than it seems like it ought to be.

When it comes to platforms, however, the sizes are almost always *larger* than they appear to be, as you can see above. The size of the object is almost always more favorable to the player than it looks like it ought to be. Shigeru Miyamoto was a pioneer of this technique, going back as far as the barrels in *Donkey Kong*. Play a few other arcade games that don't employ soft sizes, and you will quickly see what a difference they can make to a player's feelings of euphoria or frustration.

Intercepts

An intercept is an enemy timed and placed so that it interferes with a jump that Mario needs to make. The prototypical intercept is a Wing-Koopa that patrols a vertical path above a bottomless pit. If Mario jumps at the wrong time, the Koopa will intercept him along his path, sending him to his death. The general rule for intercepts is that the more of them there are, the harder the jump and/or challenge will be.

This rule of quantity is only true up to a point. Eventually the screen can become so saturated with Wing Koopas that Mario can't help but land on one after another, earning him points and possible extra lives, but this is rare. It's also true that different kinds and speeds of intercept are more difficult than intercepts that

are uniform in quality. The last important thing to know is what intercepts are not. This enemy is *not* an intercept:

This enemy is causing Mario to jump. This enemy begins the jump event, and therefore cannot also be an intercept that alters the jump event. The Wing Koopa in the first example is not the cause of the jump event, but rather an obstacle that modifies the jump event. The gap between the platforms is the cause of that jump; the Wing Koopa merely makes it harder.

Penalty

It is also important to measure the results of failure, which I call "penalty." Penalty specifically refers to the result of a failed jump event. That is, if Mario needs to jump from platform A to platform B and fails, the penalty is the result of that failure. Only guaranteed penalties are measured for this book. That is to say, if there are enemies in a pit below, there's a chance Mario could land in that pit without taking damage. This is merely a *risk* that the player must deal with, rather than a guaranteed result. The most iconic penalty comes from the bottomless pit, which will definitely cause the loss of a life. Not all jumps penalize the player with death, however. Some jumps merely do damage and some have no penalty except having to do the jump over.

I rate these penalties with numbers for the sake of consistency and comparability. If the penalty for failing a jump is only having to try that jump again, and Mario doesn't lose a life or take any damage, the penalty rating is a zero. Jumps in which the penalty is damage but not necessarily death are rated one. Instant death pits are rated two. Throughout this book, we'll see how penalty, but especially *changes* in penalty, are a meaningful part of level design. Deviations from that standard penalty are what tell us, as players, that the designer is pressuring us to learn and perform in new ways.

Vectors

Although vectors only need to be measured once, they're nevertheless very important to the game—and they're also important because in *Super Mario World* they've changed significantly from earlier Mario titles. For our purposes, we can define vectors as the forces of momentum a moving object has in the world of *Super Mario World*. Real world vectors are significantly different from *Super Mario World* vectors, so disregard your knowledge of physics because it will not help you understand this game. Mario's vectors in *Super Mario World* have two properties: magnitude and direction. Acceleration and deceleration are totally unrealistic from a real-world point of view, but they make complete sense from the point of view of game design. As we saw above, Mario doesn't gain running speed in a linear, asymptotic, or gradual fashion. Rather, Mario has three speeds and he switches from one to the next after passing a certain number of uninterrupted blocks of distance. This is our x-vector, the side-to-side motion of Mario in the game. As for the other direction, it's important to note that Mario's jump speed is completely static. His jumps simply rise and fall without perceptible acceleration or deceleration physics. He jumps at one speed and falls at almost exactly the same speed. (Falling takes just a few frames longer than jumping, to give the player a

little more time to land, but it is imperceptible to most people.) There is also a very slight pause (about three frames long) in between ascent and descent. The only things that can modify his jump velocity are the presence of water, a springboard, or the use of the cape powerup in a glide.

The x-vector and y-vector are of enormous importance to the design of the game because one of them always has a fixed magnitude. In most levels, the y-vector has a static magnitude; just as noted above, Mario's upward jump motion only has two speeds. Even when soaring upwards with the cape or falling from a great height, Mario only ever moves along the y-axis at either jump speed or at cape-glide speed—that's all. Even soaring up onto the highest platforms is a manipulation of the x-vector; Mario can only gain that extra height by increasing his x-vector magnitude (run speed) to its maximum. This is the reason why platform width is such an important datum: Mario needs that width to gain height. Once Mario is in the air, however, there are only two speeds at which he can move up and down: fall speed or cape speed. (Some master-level players can "blink" the cape glide in certain situations involving longer falls to fake a third fall speed, but this is very hard to do. There is a built-in cooldown on button presses that will start a glide, and so most players cannot switch back and forth between falling and gliding with meaningful results.) This means that in most levels, much of the problem-solving work done by the player consists of these skills:

1. Getting Mario to the right momentum
2. Standing at/jumping from the right place
3. Jumping at the right time
4. Avoiding the sudden appearance of enemies

Those four skills correspond to the skill themes in the game, respectively: the preservation of momentum theme, the periodic enemies theme, the moving targets theme, and the intercepts theme. We'll cover those in more detail Chapter 3.

Of course, there are also the water levels, and as you might expect these levels place a much greater emphasis on solving problems via the y-axis, but this doesn't mean the necessary skills have changed. What's really happened is that the game has literally turned sideways. In a water level, the vector scheme is reversed; now the x-vector magnitude (speed) is static and the y-vector speed can actually change. In a water level, the player has control over Mario's ascent and descent speeds, and can zoom along the y-axis quite quickly and with a great amount of control. The player is unable, however, to change Mario's x-axis swim speed in a meaningful way. There are only two speeds that Mario can move laterally in these levels: walk or swim. (Skilled players can cancel Mario's swim animation for a slight boost in speed, but only when directly underneath a platform. This is not often possible.)

Many of the challenges in a water level have to do with quickly going up or down in the water to avoid enemies or other obstacles. You can see how enemies in water levels are staggered so that Mario can ascend or descend between them.

Shooting these gaps in a dry level would require some serious jumping skills, but in a water level, it's not hard. In water levels, it's not that the designers have removed the platforms, but that *everything* has become a platform. Mario can jump at any time; all empty space is a platform. Naturally, this strips away the possibility of the designers using the platforming-declension themes (all water levels are either intercepts, periodic enemies or outside of the themes altogether). The moving targets and preservation of momentum themes are built upon the fact that Mario has to commit to a jump in the right way. The water allows Mario to "jump" (in a loose sense) at any time. The only problem with these challenges is that with fewer inherent design possibilities, the levels can become a bit repetitive. Each water level is different from the others, but they can feel slow and similar throughout because the designers have fewer options.

Historical Changes in Vectors

Despite all the detail laid out about vectors, the momentum mechanics present in *Super Mario World* are actually a simplification from earlier titles in the series. In *Super Mario Bros* and *Super Mario Bros. 3*, when Mario was moving at full speed, he was unable to come to a complete and sudden stop on an open plane. Instead, he would "slide" about half a block in the direction his momentum had been carrying him. This made for some frequent problems when landing on a narrow platform, since Mario's momentum could easily carry him straight over the edge into pits and enemies. To compensate for this, many players would do a momentum-breaking backwards-jump in order to land on a platform.

As if the momentum weren't enough, there was a very short (but potentially deadly) input lag/cooldown on Mario's ability to jump. These mechanics were eliminated in *Super Mario World*. The player's ability to stop Mario's horizontal momentum is now about as close to perfectly precise as human and technological limitations would allow at the time.

The second major change in game-physics is in the arc of Mario's jump. In *Super Mario Bros*. and *Super Mario Bros. 3*, Mario's jump arc was clearly divisible into sections. It's easiest to understand if you see the diagram below.

The first part of Mario's jump is a rapid ascent; the second part slows down before beginning the descent of the arc. The descent starts off slowly, and then returns to the initial speed of the jump. The reasoning behind this two-speed arc is that by giving Mario more time at the peak of his jump, it's easier to land on a target like the head of a Koopa. (There may also be technological reasons for this, but we can only account for the design effects here.) The problem is that a two-speed

jump makes the hardest jumps harder. Jumps that feature things like intercepts or moving targets are complicated by the fact that a player doesn't just have to predict Mario's landing point, but also how the slower, second stage of his arc might collide with the obstacles along the way. Eventually the player will learn how to handle those physics, but wouldn't it be more straightforward just to have a uniform jump motion everywhere? *Super Mario World* changes things so that the motion of Mario's jump is almost uniform throughout. In *Super Mario World*, when Mario jumps, he ascends almost exactly at the same speed he descends (the descent is imperceptibly longer), with only a tiny pause (about 3-4 frames) at the apex of the jump.

The meaning we find in a study of the vectors in *Super Mario World* is less obvious than d-distance or delta-height or any of those measurements. The refined and intuitive vector system in this game is in place to make this game not just accessible to a broad audience, but also to shift the primary focus of the player from systems mastery (physics) to content mastery (learning patterns in level design). While players who grew up on the earlier games might complain that this makes the jump mechanics too easy, the decision ultimately allows the designers to come up with more elaborate and inventive challenges in the architecture of the levels themselves. It's much more rewarding for the player to fail a jump because they're hit by a fireball they're aware of than it is for them to simply slip off a cliff edge they weren't even paying attention to.

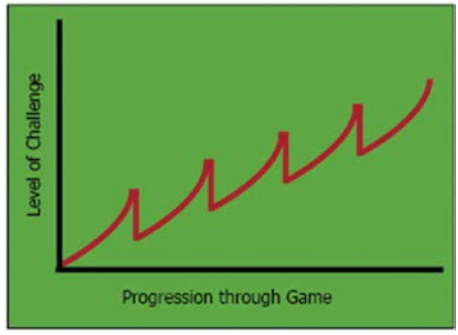
1

Half-Life and the History of Videogame Design

Through the *Reverse Design* series and other documents, we have already set forward the overall history of videogame design several times. Therefore, for this summary I will try to be as brief and as specific to *Half-Life* as possible about the overall history of games. We will go into great depth about the history of the FPS, however, because of how relevant it is to *Half-Life's* design. For a more in-depth look, I suggest looking at the first few sections of *Reverse Design: Super Mario World*, which covers the composite era in greater depth. That said, it isn't necessary to read it to understand this book; a synopsis follows just below. At any rate, this book focuses primarily on the transition from the composite era into the set piece era, and how that was an inevitable consequence of the collision of Western development techniques with Japanese game design styles. *Half-Life* straddles the composite and set piece eras in a significant way, and so it makes for a great example of the third great inflection point in videogame design history.

Nishikado Motion and the Arcade Era

The history of videogame design, as we understand the field today, began in 1978 with the game *Space Invaders*. Obviously, videogames had been invented before this, but *Space Invaders* was the first game to demonstrate the core principle of videogame design. The designer of *Space Invaders*, Tomohiro Nishikado, was also the lead engineer responsible for building the physical components of the arcade machine. Because of an unexpected property of the processor he used for the game, the ranks of enemies (the "space invaders" themselves) moved progressively faster as the player cleared the level of them. This meant that every level would get progressively harder toward the end, and then the difficulty would drop off considerably when the next level started. Although this was accidental, Nishikado kept this feature in the game and then embellished it by making the beginning of each level successively (but only slightly) more difficult. I have visualized the difficulty of the game just below.

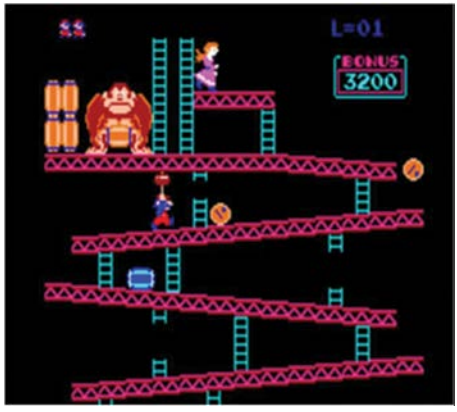


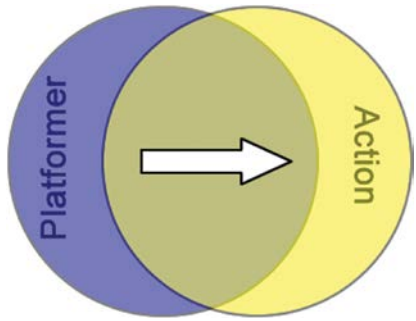
The regular up-and-down motion of the game's difficulty is what I call *Nishikado motion*. Other designers and writers have sometimes referred to this as a "series of ascending arcs," which is a fine term as well, but I'll be using the former term throughout the rest of this chapter (in this book).

In the first few years after the discovery of Nishikado motion, there were two primary ways that designers implemented it in their games. First, designers could change the numbers, strength, and timing of their enemies or obstacles. In *Space Invaders*, Nishikado accidentally changed the timing elements of the enemies; they moved faster when there were fewer of them. As the player gets through a level of *Asteroids* (1979), the asteroids themselves become more numerous and move faster. In *Missile Command* (1980), the missiles fall more rapidly and there are more of them as the player gets deeper into the game. The invention of the powerup added a second means of causing the difficulty to rise and fall, however. In games like *Phoenix*, *Galaga*, or *Pac-Man*, the player periodically gained powerups that made the game easier. For example, in *Galaga*, the multi-ship powerup doubled the player's firepower.

These powerups really just accomplished the same thing as changes in the number, strength, and timing of enemies or obstacles. For instance, in *Galaga* it's obvious that the designers were simply increasing the ship's shooting ability numerically by doubling it. It's not that different than if the developers were to simply cut the number of enemies by a large fraction. In *Pac-Man*, it's less obvious, but still essentially the same design idea. Over time, the duration of the energizer powerup gradually diminishes while the enemies only grow in difficulty. Again, this is just a change in difficulty variables from another source.

Another more important way of using powerups arose in the work of Shigeru Miyamoto. While most of the games of the early 80s used powerups as a back door into controlling difficulty, Miyamoto's first game—*Donkey Kong*—did something very different. When *Jump Man* (Mario's first incarnation) gains the Hammer powerup in *Donkey Kong*, he loses the ability to jump, but gains the ability to attack enemies. The game temporarily stops being a platformer and starts being an action game.

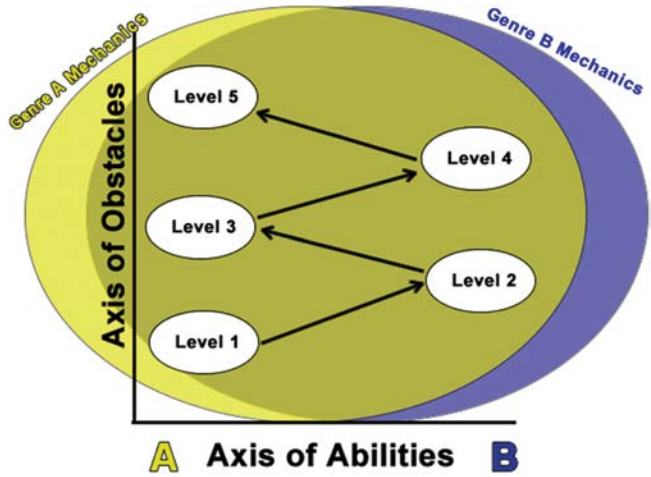




Instead of treating the powerups as another quantitative modifier of game difficulty, *Donkey Kong's* powerup makes the gameplay change genres. In *Donkey Kong*, this was a very rudimentary idea and probably the product of serendipity rather than a clear plan, but Miyamoto and his team must have gotten the sense that moving between genres within a game would be the design style of the future. The great strength of a game that moved between genres (even if only in a small way) is that the game could present new challenges to the player without always getting quantitatively more difficult. The great weakness of arcade games was that they forced the player up an endless hill of quantitative increases in difficulty. As a result, those games lost many players who became frustrated before they could really get into the game.

In 1985, Miyamoto and his team took this idea to its logical conclusion and created the first real composite game, *Super Mario Bros*. A *composite game* is a game in which a player can use the mechanics of one genre to solve the problems of another genre. In *Super Mario Bros*, the player can use platforming mechanics (jumping with momentum) to solve action game problems (defeating or avoiding enemies). The secret of a composite game, though, is not just combining two genres, but rather moving between those two genres without ever abandoning either one. Each level in *Super Mario Bros* "declines" (literally, leans toward) one of the two composited genres while never ceasing to be a combination of both. In the screenshots below, you can probably guess whether the levels in question are in the platformer (lots of jumping) or action (more combat) declensions.

The back-and-forth motion between genres in the composite creates "composite flow." This is a phenomenon similar to ordinary psychological flow in that the player becomes immersed in the task and forgets everything else. The unique feature of composite flow is that it is achieved by moving from one genre declension to the other just before the player gets bored or frustrated. All the while, however, the game is also getting more difficult. If you were to make a graph of it, it would look something like the figure you see below.



Nishikado motion is still the foundation of the composite game; it's just incorporated into larger framework. In the figure above, the ebb and flow in difficulty typical of Nishikado motion is still visualized on the y-axis, or what I call the *axis of obstacles*. Now, we also add another axis that measures changes in genre, what I call the *axis of abilities*. Immediately after *Super Mario Bros* came out, videogame designers all over the world latched onto the idea of the composite game and started making their own combinations.

Genre Creation in the Composite Period

Composite design displaced the arcade style of design, and so we call the period from 1985 until about 1998 the *composite period*, after which point another game design style became equally popular. During this time, dozens of different composites flourished, and the practice of composite design advanced considerably. Designers created some truly great composite games through innovative combinations. *Mega Man* and *Metroid* both added shooting to platformers to great effect. *Sonic the Hedgehog* took the Action/Platformer composite of *Super Mario Bros* and added racing game mechanics. *Castlevania* added RPG elements to the *Mario* formula. *ActRaiser* created an

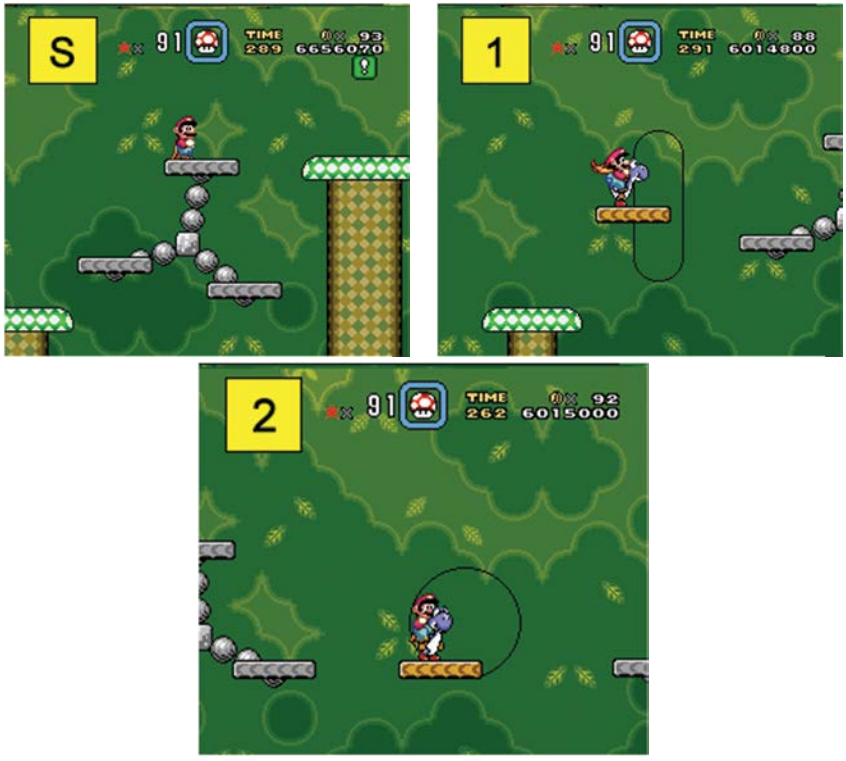
odd but extremely likeable composite out of the Simulation, RPG, and Action/Platformer combination. Even after the heyday of composite games, we still see new composites like *Portal*, which allows the player to solve platforming problems with shooter mechanics, or *Katamari Damacy*, which is really a racing game that operates by an accelerated RPG level-up system. *Half-Life* is partly a composite game, involving both the FPS genre and a considerable amount of platforming. The relationship between *Half-Life's* composite parts is unusually complicated, because *Half-Life* is both a composite game and a set piece game, and the kind of composite game that *Half-Life* draws from has some special properties, too.

One of the most surprising developments of the composite era was the creation of new genres out of old ones. Plenty of games combined two genres in a way that left those two genres apparently intact. For example, everyone can see the way that *Mega Man* or *Metroid* alternates between shooter and platform content and sometimes mixes the two. Similarly, the RPG and action elements of *The Legend of Zelda* are still distinct. In the middle of the composite era, however, composites began to appear where the mix was blurrier. The real-time strategy genre is a good example of this. There are plenty of examples of strategy videogames, but players of the "pure strat" game tend to disdain the RTS as not really being "strategy." In a certain sense, this is correct because the RTS is quite a bit more than just strategy. *Dune 2*, the first real RTS, mixed several other genres into the formula. By adding not just action game combat, but also *Sim-City*-style economic simulation, the RTS became its own distinct genre. That new genre didn't really retain the audiences of any of its composited parts; instead, it created a new RTS-specific audience.

The FPS genre is largely the same. Obviously the FPS is a shooter and shooters go all the way back to the 1970s, but consider how little overlap there is between the hardcore enthusiasts of the FPS and the 'shmup, for example. They're both shooters, but the audience is different, and that difference stems from the genre composite. The first FPS, *Wolfenstein 3D,* brought together the aiming and dodging elements of the shooter, but it adopted first-person mechanics, exploration, and level design of the CRPG.

On the left is the layout of the UI from a PC RPG. On the right is a screenshot from *Wolfenstein 3D*. *Wolfenstein 3D* retains the elaborate dungeon, the stat-oriented HUD, the secret treasure rooms, and progressively more powerful equips of an RPG. It has the fast-paced, live-action feel of a shooter. Despite this obvious mixture, the FPS does not have the same audience as the CRPG or the 'shmup. The FPS has its own audience, one that has come to be a dominant force in gaming today.

Before we get to the set piece era, I want to highlight a design practice from the composite era that carried over into *Half-Life*. Earlier, I mentioned the challenge, cadence, skill theme (CCST) structure that appeared first in *Super Mario World*, but then spread to other games. At a basic level, this structure is a way of tying together the challenges in a level so that the level seems consistent, but also so that it becomes progressively more challenging. The consistency arises from a precise form of iteration. Designers start with a simple idea at the beginning of a level. I'll use some examples from *Super Mario World* to illustrate the point first because it's much easier to see the iterations in that game, then we'll see some examples from *Half-Life*. A Mario level starts off with a simple challenge made up of just a few jumps. This challenge then gets more complicated.



You can see here how in the first iteration, the number of available platforms has decreased, and the type of platform has also changed. In the second iteration, the minimum distance between platforms and the total number of jumps have gone up. The first iteration is an example of mostly qualitative changes, which I call an evolution. The second is an example of a quantitative change (the distance between platforms grows numerically), which I call an expansion. These terms are used throughout this book and are necessary to understanding some of the concepts that appear later.

Evolutions and expansions on their own are meaningful, but the relation of sequential iterations is where we begin to understand the real structure of a game's design. When we see several iterations upon one design idea, I call the relationship between those iterations the cadence of a level. The relationship between many of the set pieces in *Half-Life* can be described this way as well. *Half-Life* doesn't structure entire levels the way a composite game would; it doesn't design whole levels around any single design philosophy. *Half-Life* is a transitional game and is inconsistent in the design styles it uses, but there are definitely segments of the game in which the designers take the same basic design idea (or core of a set piece) and iterate it several times to form a cadence structure. A good example of this is the very similar arena-style challenges in Chapter 14.

You can see how the same arena-style room with a cut-out floor in the center is the basis for all three of these set pieces, and yet, each one is different. One set piece adds barnacles; one set piece puts three grunts on the floor instead of dividing them between floor and catwalks. One set piece has multiple teleport-spawns. The cadence of Chapter 14 tells us something—that the designers are shifting away from the cover-based content they began with and are now focusing on a series of evolving arena encounters. Of course, the cadence of level 14 isn't composed of challenges, it's composed of set pieces, and that difference is meaningful.

Although *Half-Life* uses many composite design techniques, it is also one of the first examples of the next era of game design: the set piece era. The set piece style of game design grew out of the practical necessities of increasingly large software projects. As videogame production grew in size, duration, and profitability, it was inevitable that the business and production constraints of making games would impact design. When this happened, game design entered the set piece era. Until the mid-90s, videogames had been created by relatively small teams who would work together for multiple games. Their small size and familiarity with one another made communication easy, and so it was easy for those teams to adhere to a shared vision for a game. Once teams grew larger and had more replaceable parts (with people moving onto or off of a project frequently), it became harder for the team to maintain a single design style. Thus, software production methods that were employed in non-game projects began to have more prevalence in the production of games. The impact on game design that these new methods had was visible as modularity. If members of the project couldn't always communicate easily with one another, and if staff members were moving on and off the project frequently, it made sense for projects to modularize their content to a greater degree; modular content offers convenience. Modular content also requires less intensive collaboration because solo designers can create numerous small modules from pre-existing tools and assets, without having to draw in other team members very often. That module can be moved around in a game to find wherever it fits. As long as that module fits the style of the game, everything else about it is essentially fungible.

The module that began to appear in *Half-Life* was the set piece. There are three important characteristics of a set piece. A set piece is an extended, self-contained and quantitatively-focused unit of content.

1. By *extended*, I mean that the content of a set piece lasts much longer than the content of a challenge, which was the primary unit of content in a composite game. A level in a composite game would have many challenges, usually more than 10, and each challenge would last only a few seconds. Set pieces tend to last for around 60 seconds or more, and can last as long as five minutes each. Accordingly, there are fewer set pieces in a game like *Half-Life* than there are challenges in a typical composite game.
2. *Self-containment* means two things. The first is that a set piece is a unified action; once the player begins a set piece, they have to finish it in order

to move on. The second aspect of self-containment is the way that every set piece tends to contain within it everything that the player needs for that set piece. The best example of this is the phenomenon of regenerating health/shields. If the player character can heal fully between every set piece, then the designers never have to worry about whether the player is entering set piece seven with enough health to survive after completing set pieces one through six. The same is largely true of ammo; if every enemy in a set piece drops a little bit of ammo and maybe a grenade or two, the designers never have to worry about the player suddenly running out. The set piece contains everything the player needs to beat it.

3. The final defining aspect of a set piece is an emphasis on *quantitative changes* in content between modules. While composite games focused on qualitative and quantitative changes, it was easier for set piece games to emphasize quantitative changes because they are so easy to communicate. If a series of replaceable low-level designers are simply trying to push out as many set pieces as possible while using the same set of tools, they can simply make sure that each successive set piece has more enemy marines than the previous one did so that the difficulty of the game keeps rising to challenge the player.

All of what I said in the list above is a simplification, of course; no game is entirely built on a string of set pieces that only change numerically. *Half-Life's* focus on qualitative variation is striking, in fact. Although Half-Life begins the trend of set pieces, it doesn't develop the idea all the way into its modern form. The biggest discrepancy between *Half-Life* and the "modern" set piece game is the absence of regenerating health. *Half-Life* takes a step in that direction; a majority of the set pieces in the game feature some kind of health-regeneration apparatus before or after the battle, but very few set pieces feature health regenerating items in the middle of the fight.

The regularity with which these regeneration stations come between Chapters 3 and 13 was especially astonishing for the time. The great FPS games had previously

featured a lot of level-long attrition. It's clear that the designers of *Half-Life* were trying to move to a more modularized kind of content, but it hadn't yet reached the point where the player could hide behind a piece of cover and regenerate fully. That extreme would come a few years later. Additionally, *Half-Life* lacked the quantitative emphasis that its descendants would have. There are sections in the game that are more quantitative than others—Chapters 8, 13, and much of Xen spring to mind—but overall, the designers of *Half-Life* still employed a much more cadence-like sequence of qualitative evolutions. These two differences are part of *Half-Life's* identity as a transitional game; even while beginning the set piece era, it shows its composite heritage.

The Vocabulary of Cover

The story of *Half-Life's* design is largely the story of shooter cover. The largest single portion of the game's content is a series of set pieces that involve meaningful cover. Before I begin to talk about cover, I want to come up with a functional definition for the purposes of this book. Technically, anything that a player uses to hide from enemy fire is cover, but that definition is unhelpfully broad. Cover in *Half-Life* is at its most meaningful when it allows the player to move in and out of safety while firing on the enemies. To illustrate, I'll pull from a different game from the *Half-Life* era: *Time Crisis*. In *Time Crisis*, the player is able to use a pedal (in the arcade) or button (at home) to move their character in and out of the sheltering safety of a wall, box, or other object.

While in cover, the player character is unhittable and can reload, but the player cannot see the enemy or fire. The player can press the pedal or button to pop out of the cover and take shots. Good players will know exactly when to pop in and out of cover so as to avoid damage and still shoot enemies. It is this pop-in/pop-out process that defines what I mean by cover. If the player has more than a couple of seconds where he or she can use cover in this way, then the game is deploying a cover-based set piece.

The Earliest Cover

As far back as *Space Invaders*, cover has been a part of shooter designs. We already saw in the previous section how *Space Invaders* was a very forward-looking game for its basic design structure, but its shooter-specific elements were also prescient. Not only did *Space Invaders* have cover when many subsequent shooters did not, but that cover was destructible.

Cover would disappear as a meaningful part of shooter design until the composite era. Side-scrollers in the mode of *Galaga* became the dominant type of shooter during the arcade and early composite era. Having little or no cover, the design of scrolling shooters (would later be called 'shmups) instead emphasized constant movement and did so in a way that diverged from the path eventually taken by *Half-Life*. The first problem in a scrolling shooter is not being shot; actually hitting targets is secondary to that because the player has so little in the way of HP and extra lives. As the genre evolved, this emphasis on dodge-first, fire-second mechanics became increasingly apparent. This is why 'shmup powerups tend to greatly magnify the player's field of fire (by adding extra shots, beams, or explosions). Those powerups are designed to aid players whose first concern is surviving, not aiming. Usually, those weapons also feature infinite ammunition because precision firing is not the point of the game. Obviously, as a player masters the game they will be able to concentrate on both dodging and aiming, but moving and dodging are the central skills.

During the composite era, the shooter genre was an important ingredient in many classic games like *Metroid*, *Mega Man*, and *Contra*. These shooters all made some use of what we can call cover, although its implementation was different than that of a pure shooter. All of the games listed above are composites of the platformer and shooter genres, and so most instances of cover take the form of platforms.

The player isn't hiding behind an object as much as waiting to make a jump. Although many platforms or other pieces of level architecture serve as cover, their true purpose is to force the player to make platformer-style jumps. Often, those jumps are combined with shooting tasks, but this means that a lot of the "cover" in those games makes the tasks in question more difficult, rather than less difficult. While there are definitely some similarities between this and what we'll eventually see in the FPS genre, that one difference is very important. *Half-Life*

has plenty of examples of platforming content, but only two examples where that platforming content blends with shooting in a meaningful way.

The FPS Beginnings

Cover in the first person shooter began, like so many great videogame design ideas, as an accident. We've already discussed how the level design of the early first person shooters was derived from western CRPGs and adventure games. The game to show this most clearly is the first real FPS, *Wolfenstein 3D (W3D)*. The very dungeon-like structure of early *Wolfenstein* levels is obvious just from even a cursory play-through. These levels don't resemble anything in the modern FPS.

Wolfenstein features nothing immediately identifiable as dedicated cover. All those walls and doors look alike, regardless of whether there are enemies beyond them or not. That said, the walls and doors of the dungeon can be used as cover because of the way that enemy AI works. In *Wolfenstein*, enemies move in herds. There are three principal characteristics of an enemy herd:

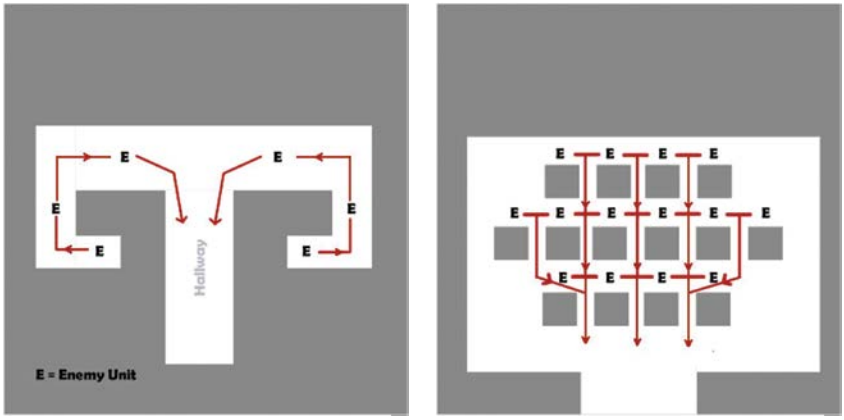
1. Herds of enemies share aggro.
2. Herds of enemies feature no distinct tactical roles, although they may have heterogeneous firing ranges and projectile types.
3. Herds of enemies, no matter what shape they start in, always collapse on the player.

Not every encounter with enemies is with a herd; there are plenty of solo enemies in *W3D*, too. When a player does encounter a herd, though, what tends to happen is that a room full of soldiers will start moving towards the player character as he stands at the point of encounter. That point of encounter is most often a doorway or aperture into a new room, even though that door isn't designed to operate as cover, merely as a way to inflict a sudden surprise on the player. When the herd collapses on the player who is moving and shooting, the player can move backwards through the aperture, cutting off the enemy's angle of fire and stacking

them up neatly in the aperture space or around the edge of a corner. By reducing the number of angles from which enemies can shoot and stacking enemies up so they can't fire around each other, the player is in a very basic form of cover.

The architecture and AI in *W3D* do not demonstrate any intent on the part of the designers to allow for what we perceive as cover mechanics. Rather, the cover-like strategies that players employ are an emergent property that the designers probably didn't plan for in the original game. Every doorway is more or less the same shape, regardless of its proximity to a group of enemies, meaning that no door is tactically more useful than any other. The doors in the game close automatically after just a few seconds, meaning that sustained firefights through a doorway are interrupted without much warning. Furthermore, the controls in *W3D* do not allow for strafing. The left or right movement controls rotate the player character, and mouse controls (which were added later) are not independent of key controls either. Ducking in and out of cover is difficult and only possible in a few limited circumstances.

The development team must have noticed the emergent strategy of using rudimentary cover because the sequel to *W3D*, *Spear of Destiny (SoD)*, takes frequent advantage of it as a part of level design. What we see in *SoD* are huge, oddly-structured herds that are designed to force the player into the "cover" situations described above. Although the whole game is more or less based around this one idea, it can take a variety of forms in terms of practical execution.



Both of the encounters above operate by the same principle: a large herd collapses on the player and the unusual level architecture causes the herd's movement to approach the player character in a variety of strange ways. The ram's-horns structure on the left sends successive waves of enemies at the player. The game's AI doesn't support delayed movement, but it does support herds of any shape, and so by staggering the enemies geographically, the designers were able to stagger

their emergence from the corridors as well. The player can manipulate the enemy action by backing down the hallway and stretching that herd out further, to pick them off one or two at a time from behind a corner. In the "forest" of columns on the right, above, the herd will suddenly appear from behind those columns when the player obtains aggro on any one of them. The player has to move around the outside perimeter of the columns to isolate small groups of enemies before the whole herd comes out into the open. By changing position, the player keeps the herd in the columns where they can't all fire at once.

While it's clear that the designers of *SoD* are able to communicate to the player the importance of minimizing exposure to enemy fire by using the terrain, it still doesn't look like cover as we understand it. For one thing, in both of the aforementioned examples, the player is mostly standing or moving around in open space; that's the case throughout *W3D* and *SoD*. It's really the enemies that are in what we might call the "cover" rather than the player because they get tangled up on doorframes. Later FPS games preserve this to some degree; doorways are always a great place to stack enemies up. Some FPS games also preserve the constant movement and large open spaces of *W3D* and *SoD*, but *Half-Life*, as we'll see, diverges from that idea in a profound way.

The next evolution in the meaning of cover comes, of course, from the early *Doom* series. There are four big advances in *Doom* and *Doom 2* that change the way the player interacts with enemies. The biggest change was the introduction of the ability to strafe. Modern players trying *Doom* for the first time may find that strafing ability to be much clumsier than what they're used to, but they expect to find it and they do. Players who started with *W3D*, on the other hand, would have found the ability to strafe to be an incredible boon, making it possible to dodge enemy fire in a way hitherto impossible. Really, *Doom* was just reclaiming an ability lost when the shooter took on the FPS form. Strafing is essential in *Space Invaders*, and every scrolling shooter that followed it because those games were about dodging as much as they were about shooting. With the ability to strafe came the ability to use level architecture as real cover by popping in and out of cover.

Thus, the second big advance was in a greater variety of level design forms. These forms are fairly chaotic and mostly defy any form of organized classification, but they're a lot closer to what we understand as being "cover" in the modern sense of the word than anything in *Wolfenstein* was. You can see in the images above how these new architectural developments were only usable because of the player's ability to strafe.

The other two big changes were the introduction of the iconic (but short-lived) monster closet, and the arena-style disposition of health and ammo. The monster closet is one of the most highly-recognizable features of the FPS genre, and it definitely had an impact on level design in its heyday. For those who aren't familiar with the term, a monster closet is simply a hidden cache of enemies which opens when a player passes a certain point or takes a certain action. Triggers for monster closet openings can include hitting a switch that also controls doorways, or even picking up a new weapon or item. Much of what a monster closet did was simply to refill areas already cleared of enemies by the player at an earlier time.

The area the player is staring back at in these screenshots is the passage that actually leads to the dead end where the player stands. A large number of enemies emerge from monster closets in previously-cleared hallways and move, as a herd, toward the player. One rule that tends to be true for monster closets in general is that closets that open at the same time tend to feature enemies who are in a single herd, no matter how far apart those closets might be. The other purpose of a monster closet was to fuel a frantic arena battle. Many monster closets in later battles put enemies on every single side of the player character, meaning that the player has to be moving constantly to avoid close-range fire.

The use of monster closets to turn empty rooms into frenetic arena encounters is where the last big *Doom* innovation, the disposition of items, comes into play. In *W3D* and *SoD*, healing items and treasure were located in hidden rooms and alcoves, much as they would be in an RPG or adventure game. In *Doom*, many of those items are strewn around the perimeter of an intense arena encounter (in this case, they're actually on the closet doors) to encourage movement—especially circle strafing.

The obvious purpose of these items is to replenish the player character's health when numerous demons suddenly leap out of the walls and start firing. All of the frantic running helps to dodge enemy fire, but it also allows the player to pick up items from around the floor of the arena. The ammo, health, and armor make it possible to sustain an intense, roving battle without real cover. Later in the history of the shooter, these items are going to appear less often in the middle of battle (although they'll never cease to appear there), and will be moved to the battle's beginning and end. Sometimes, rather than placing items directly in the line of fire, the designers placed health and ammo in the very closets which have opened to let out monsters. The player has to charge through and around the enemies in order to access it. It makes sense for *Doom* to put its health in dangerous places because that placement achieves what the *Doom* designers wanted: reflex-oriented shooting sprees. The game that they were designing is much more frenetic than most of the shooters that we play today; *Doom* is a kind of game that was much closer to its 'shmup ancestors than it is to its cover-based descendants.

Big Advances: Quake and Quake 2

Half-Life originally started as a mod of *Quake 2*, so it's no surprise that it owes much to its immediate ancestor in terms of design as well as in terms of technology. Indeed, the first two *Quake* games are probably the most influential shooters of all time from a mechanical perspective. *Quake 2*, especially, solidified the fundamental mechanics that still make up the foundation of contemporary FPS games. *Quake* made the transition to true 3D. In *Wolfenstein* and *Doom* (and the lesser-known *Marathon*), the environments were 3D, but the characters were all 2D sprites that were (effectively) infinitely tall. The player never has to aim upwards (not that it was possible to do so) because both the character's shots and the enemy's sprite have no upper limits on the y-axis. If the player is aiming at the correct coordinates on the x and z axes, the bullets will hit. In *Quake*, the player has to actually aim at the 3D model of the enemy to hit them. Aiming is actually possible in *Quake* in a meaningful way, where it wasn't before. Before *Quake* the player could only shoot at the center point of the screen; the player had to move their character to move the target icon.

Quake separated the viewing/aiming controls from the movement controls so that the player could fire in any direction—including away from the direction the character faces.

The introduction of true-3D aiming mechanics opened up many new design ideas above the mechanical level. The most widespread of these ideas was the introduction of zone-specific damage (i.e. headshots). Both *GoldenEye* and the original *Team Fortress* (a *Quake 2* mod) used this as a natural outgrowth of the ability to target enemy models precisely. Another thing both games did was to put a greater emphasis on verticality in their level designs.

Half-Life would take both of these things significantly further. One thing I want to point out here about these games is how the mechanical and supra-mechanical changes like verticality led to the use of modern cover. Once players were finally able to really strafe and aim precisely, popping in and out of doorways or just over the lip of a catwalk to take a quick shot at distant enemies became a viable strategy. In *Doom* and *Wolfenstein*, the player's ability to do this was much more limited even when there was a type of "cover" available. In *Quake* and *Goldeneye*, it's a lot easier to perform this maneuver, but even then cover play is really only an emergent property of new mechanics rather than a conscious effort by the designers to create cover-based encounters.

Although *Quake* had incidental cover and the mechanics to use it, the series went in a very different direction than *Half-Life* eventually would. *Half-Life* took the seed of the cover idea from *Quake* and developed it into an entire game because *Quake* did not. *Quake* was designed to be fast-paced and chaotic in the same way that *Doom* was. The large apertures and empty rooms of *Quake* certainly don't make it easy to avoid enemy fire.

Encounters with some of the large, durable, and aggressive enemies in *Quake* make pop-in/pop-out tactics unsustainable for game-wide usage. For better results, the player needs to perform maneuvers like kiting, circle-strafing, and jumping up and down through vertical portions of the levels to avoid the enemies. This kind of combat would come to be known (after *Half-Life* and several more *Quake* sequels) as the true arena shooter style. The arena style of combat isn't limited to single player either; many of the most popular multiplayer maps from the early *Quake* games are actually complete single-player levels with the computer-controlled enemies removed.

Half-Life takes the emergent cover mechanics of *Quake* and builds the majority of its content from that idea. *Half-Life* also features some of the arena combat that we traditionally associate with *Quake* and its descendants, as well as a lot of platforming content which comes from the composite design tradition. These are the three ingredients that make up *Half-Life*; for the purposes of analysis, we'll call each of these ingredients a "design theme." The single largest design theme is the cover theme, which makes up about 55% of all the set pieces in the game. This theme also sees the greatest amount of development (increasing complexity) from beginning to end. *Half-Life* does so much with the use of cover that most of the lessons we can learn from the game are about cover as it relates to level architecture, pacing, and the use of AI and weapons.

The Structure of *Half-Life*

Even though *Half-Life's* cover theme represents the largest portion of the game's content, that theme doesn't start until Chapter 4, reaches its climax in Chapter 12,

and then drops off significantly after Chapter 13. This means that the cover theme is densely packed into those 10 chapters. It also means that the game has a peculiar structure because there are 18 chapters and only three design themes to fill them with. There are certainly pieces of content in the game that don't fall into any of the themes; usually these take the form of boss fights or small pieces of through content. Most of the game's content comes as set pieces, though, and most of the set pieces fall into one of the three themes. Of the three themes, the cover theme is simply the most frequently seen.

One of the most-discussed problems of *Half-Life* is that the last four chapters (in Xen) are stylistically different from the rest of the game, and also of a lower level of quality. Some of that is the sudden and radical change in the gravity and accompanying emphasis on long-range jumping puzzles. It's a bad idea to introduce sudden, permanent changes to the fundamental physics of a game when the player is more than 80% of the way through it, but I think the real reason that the Xen levels feel so much weaker is that the game design goes backwards in time. Xen, with its sudden lack of meaningful cover, emphasis on arena-style encounters, large hallways/apertures and frequent, small engagements is a lot more like *Quake* than it is like Chapters 4 through 13 of *Half-Life*. If we see the cover theme as the core of *Half-Life's* design, then the climax of the game is in Chapter 12 (Surface Tension), with Chapter 13 serving as a kind of denouement to that. What if the portal that opened in Chapter 14 had gone directly to the fight with Nihilanth? I think the overall flow of the game might have seemed a little more even, but that isn't what the designers made.

Instead, the arena theme comes to dominate the later chapters of the game with large doses of platforming mixed in. In the rest of the game, the incidence of platforming and arena content follows the rhythms of composite flow, although not in the traditional 1:1 ratio that a true composite game might employ. From Chapter 4 until Chapter 13, what we usually see are two to four set pieces in the cover theme followed by one or two out of the platform theme, the arena theme or a unique piece of through content. There are exceptions to this rule, especially in Chapters 8 and 10. Chapter 8 has a lot of long, slow through content thanks to the monorail cart Freeman has to ride, and so it eschews the kinds of arena and platforming challenges that might make the chapter much longer. Chapter 10, meanwhile, is entirely made up of platforming—and some fairly orthodox platforming, at that! Chapter 10 could almost be taken from a straight 3D platformer in the way it operates. Starting in Chapter 11, set pieces in the arena theme start to appear more often, and then they mostly displace cover-based set pieces in Chapter 14 and after, with a couple of interesting exceptions.

A Note on Mechanics

This work primarily concerns itself with level design, and little has been said so far of the shooter mechanics at their most basic levels. I have not deliberately ignored them, but I feel that while *Half-Life's* shooter mechanics are very solid,

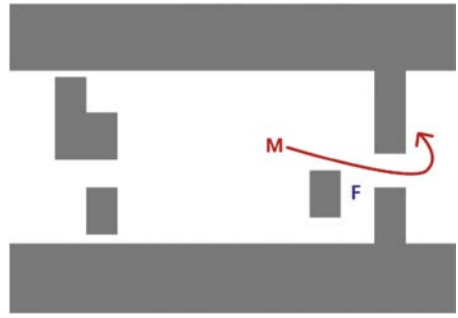
they are generally unremarkable. The auto-aim (or aim-assist) function, for example, is mostly like other aim-assist mechanics before and after *Half-Life*. The only big difference in *Half-Life* is that the always-centered reticle will come unstuck from the center for a second or two if an enemy enters the magnetic area of the auto-aim function. Aside from very minor differences like that (and couple of other things we'll get to, below), there are not huge mechanical innovations in *Half-Life*. Weapons do not do variable damage (i.e. there are no randomly-chosen critical hits) except when hitting a specific area on the enemy target (i.e. headshots). There are not really any specialized weapons for specific enemies. Only the rocket launcher has a truly novel mechanic in its laser guidance system.

This screenshot is from set piece 12–6, during which Freeman has to fight a helicopter from the mouth of a cave. Forcing the player to track the target with a laser pinpoint defeats that player's ability to hide behind cover. That is, the player can't simply fire a rocket and then duck behind the ledges and walls that would otherwise be available here. This means that players have to learn to shoot rockets at relatively close range in order to shorten the length of time that Freeman is exposed to fire while steering that rocket. That's a novel and important mechanic, but most of the other weapons are simply versions of things that appeared in games before *Half-Life* or that don't receive that much development. (For example, the enhanced recoil from the tau cannon is a new idea, but it's never particularly relevant to any single set piece or platforming puzzle.) What makes the weapons interesting is the way the levels are designed. This becomes a lot more apparent when it comes to multiplayer. If you buy, or have bought, the eBook version of this document, there is a section detailing how the most successful multiplayer maps drew most of their appeal from their use of cover. Many of the more arena-oriented *Half-Life* maps were forgotten because they were too much like *Quake*,

and did away with all the level-design innovations that really made *Half-Life* what it was.

The one big exception to everything I wrote above is *Half-life's* AI. The AI of any game is actually composed of many mechanics, although only two of those mechanics are really continuous throughout the game. Many reviews have credited *Half-Life* with having some of the best AI of all time, but this impression is the result of a series of clever illusions. While the AI in *Half-Life* was certainly ahead of its time, many of the most memorable moments in *Half-Life* are not a result of continuous AI, but rather of discrete, one-use scripts and specially-created tools. (There is an entire section about these moments in the later part of this document.) Whether it's an unusual emphasis on grenade-tossing or a sudden and preternatural ability to detect Freeman no matter what he does, these behaviors are examples of custom scripts that exist beyond the normal scope of the AI. Clever level designers can create the illusion of greater enemy intelligence by using hidden scripts, but once players figure out where those traps are laid, they can avoid them. For example, if enemies run only to specific locations, that's a one-use, custom script and not a part of the general AI. The system of decisions that enemy units can make independent of custom scripts is what really counts as a systematic advance in artificial intelligence.

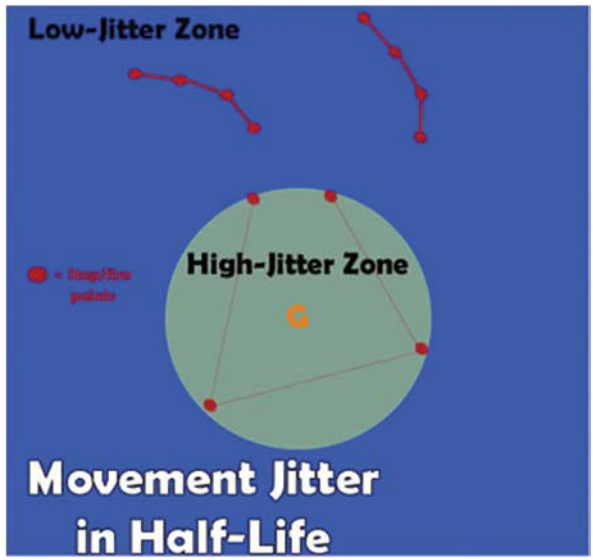
The fundamental AI which almost all marines are equipped with features two new ideas: cover-seeking and what I call "movement jitter." Cover-seeking is just what it sounds like; under certain conditions, the HECU marines will seek cover. The circumstances vary, but generally come down to two criteria. The first criterion is the presence of other marines; if they have companions, marines will often fall back towards those companions and take cover temporarily so as to draw Freeman into a trap. Marines in groups of one or two don't do this very often. The second criterion is health; marines low on health will often seek cover of any kind just to get out of Freeman's line of fire. In either case, but more often the second case, marines will display a flaw in the design of the cover-seeking behavior.



(Marine in red, Freeman in blue, cover structure in grey, the white space is empty.)

Marines are not very picky about their choice of cover when the behavior is triggered. Sometimes, they'll seek a piece of cover that is on the other side of Freeman, meaning they have to pass right by him while he fires. This bit of design has been fixed by *Half-Life's* inheritors, who have used methods like threat gradients and invisible beacons which call enemies to pieces of cover specifically. Indeed, this might be *Half-Life's* most antiquated design idea. We can still learn from watching enemies take cover because that tells us something about the cover design of the set pieces, but the underlying design idea is only remarkable for its place in the history of FPS design.

Movement jitter, on the other hand, means that enemies in *Half-Life* (mostly marines) have irregularities deliberately programmed into their movement patterns. When cover-seeking behaviors are not active, the first priority of a marine is to acquire line of sight on Freeman. After that happens, the enemy's behavior then gains the movement jitter attribute. Movement jitter usually means that an enemy will take stuttering steps while firing upon Freeman. At range, these steps are short and somewhat frequent. In close quarters, enemies can move quite a bit in between shots. Below, I have compared *Half-Life's* movement jitter to its predecessor behaviors in the *Wolfenstein*, *Doom*, and *Quake* games.



Because I do not have access to the decompiled code of *Half-Life*, I can't determine how the high-jitter, point-blank zone works. (The *Half-Life* SDK has many tools, but it doesn't instantly recreate the AI as it exists in *Half-Life*.) It may be that the principal line-of-sight-seeking behavior causes enemies to prefer a longer range, secondarily causing them to perform a run that appears highly jittery. It may be that any enemy who reaches point-blank range is subject to cover-seeking behaviors. Nevertheless, the practical effect of point-blank range on the AI makes enemies run around in unpredictable ways, even when they do not appear to seek cover. As far as the player's experience goes, the extreme movement jitter of point-blank range seems only like an exaggeration of the same behavior that is occurring at long range. Ultimately, it's easy to see how the development of movement jitter was an important mechanical evolution in the history of FPS campaigns. While many aspects of single-player campaigns before *Half-Life* prepared the player for the level-design aspects of multiplayer combat, it was rare for single-player enemies to approximate real players in their behavior. No game-based AI can ever match the resourcefulness or intuition of a human player, but by giving the player a more difficult target, *Half-Life* made its single-player campaign more interesting, and its multiplayer experience more accessible (if only slightly so).

1

The Different Kinds of RPG, and How *Diablo II* Borrows from Them

The Original Problem of Digitial RPGs

From an academic perspective, the best thing about RPG history is that it has a clear beginning. The first RPG was definitely one of the early forms of *Dungeons & Dragons* (hereafter *D&D* except at the beginning of a sentence), which were developed from 1968 through 1970. The development of *D&D* explains why there are so many different kinds of RPGs today, and why the audience for one type might not like another. In the previous book in this series, *Reverse Design: Final Fantasy VII*, I laid out a fairly comprehensive history of the path from *D&D* to modern RPGs. If you're looking for a longer explanation of RPG design history, I recommend that book and the many great books it cites, especially the work of Jon Peterson and Shannon Appelcline. For this book, however, we only need to look at two events that happened relatively early in the history of RPGs: the birth of the specialization style and the birth of the action RPG. These two events occurred in reaction to *D&D* and created the majority of the design material that would eventually make up *Diablo II*.

Dungeons & Dragons presents a huge problem for RPG designers: it does almost everything that an RPG can do. By the early 1980s, *D&D* already included

hundreds (if not thousands) of important RPG design ideas, and these were not small ideas, either. Most of the structural systems which we would recognize in our modern RPGs had already seen a few iterations by the time the books comprising *Advanced D&D* (*AD&D*) came out. Ideas like random enemy spawns, randomized loot, lighting levels in dungeons, ability cooldowns, difficulty settings, and item creation all existed in *D&D* long before any videogame designer attempted to use them.[1–3] The staggering completeness of *D&D* meant that designers had a hard time making an RPG that felt mechanically original. Many of the first RPGs to emulate *D&D* only changed the setting and aesthetics. There are many games that are more or less *D&D* in space, or in steampunk settings, etc. Some of those games are classics, but they didn't really change the scope or methods of the RPG genre. One game took the opposite strategy, however, and focused only on a small subset of *D&D's* mechanics. That game was *Rogue*, which spawned the roguelike genre.

Rogue abandons all the parts of an RPG that do not directly relate to dungeon crawling. There are no towns, there is essentially no plot, and there are no mini-games or dialogue trees. There is only dungeon crawling. The goal of *Rogue* is to get a player into a dungeon as fast as possible, and keep him or her there indefinitely—always crawling more dungeon. *Rogue* doesn't have any features that were not, in some form, originally invented in *D&D*. Yet, *Rogue* seems very different from its source material because it focuses so intently on doing one thing. *Rogue* can kill the player character more often than the average *D&D* campaign because the player is back in the dungeon after a few button presses. Moreover, *Rogue* can force the player to learn all the tricks of its dungeons in a short period of time because there aren't that many tricks. The player doesn't have to worry about party composition and job classes. *Rogue* is also able to have a greater degree of volatility in its random operations because the game is too short for any single game event to have far-reaching effects. The player can never really lose more than a few hours of progress. These tradeoffs are the secret to the success of *Rogue*, and all games that use the same design strategy. *Rogue* gets rid of certain RPG design ideas to embellish others. This design strategy is what I call *specialization*. A specialization-type RPG is any RPG that focuses on just a tiny part of the *D&D* source material. In spawning the roguelike genre, *Rogue* made a template for a certain kind of specialization that the *Diablo* designers would eventually use.

There are two design ideas that are embellished in historically important ways in *Rogue*. The first idea is procedural content generation, and the second idea is permadeath. Procedural content generation means that some parts of a game's content, like level layouts, monster stats, and magical item properties are created by an algorithm rather than by the direct work of a designer. Rogue relies very heavily on procedural content generation. Procedural content allows for an effectively infinite variety of dungeons to explore, which is necessary for *Rogue* since there's nothing else to do in the game. *Dungeons & Dragons* actually pioneered procedural content generation, although it relied on dice and the human imagination to implement it. Room layouts, item drops, and enemy encounters in *D&D* are often augmented by (paper) procedures. *Rogue* simply

1. The Different Kinds of RPG, and How *Diablo II* Borrows from Them

goes further than *D&D* does by making nearly every aspect of the game design procedural. Permadeath, on the other hand, requires that with every death of the player character, the game starts over from scratch, with nothing preserved. This makes *Rogue* longer by forcing the player to play fresh dungeons every time he or she dies. For arcade games of the 1980s, this is a common enough strategy for extending the life of a game's content. For RPGs, with their focus on long-term character progression, this is not especially common. Parties in *D&D* campaigns are, of course, wiped out all the time, but *Rogue* embellishes this idea by making it more common, and by eliminating most of the player's defenses against it. Few dungeon masters kill their players' parties multiple times an hour, but in *Rogue*, this would not be surprising at all.

As an important historical aside, I want to point out that while the key embellishments in *Rogue* were a product of an artistic reaction against *D&D*, the game was also shaped by technological concerns. No commercial computer in 1981 could have simulated all of the major systems in *D&D*. *Rogue* saves time by abandoning a large amount of the standard *D&D* systems. *Rogue* goes further, however, in that it also abandons content in favor of algorithms. The tabletop RPG lives on content. That is, a designer (such as your group's dungeon master) creates and/or deploys a series of pre-constructed encounters which add up to a campaign. *Rogue's* designers turned this content generation almost entirely over to a set of algorithms that construct dungeons, place monsters, and dispense items. This lack of content saves a lot of disc space, which, in 1981, was a precious resource. It also saves the design team a lot of time and money spent on making content. The next section looks at how, as technology and time constraints loosened, later roguelikes and derivative subgenres preserved the primacy of procedural content even when they didn't have to. Eventually, a technological constraint became an artistic constraint, and the roguelike became an ingredient in other kinds of RPGs.

Roguelikeness as a Commodity

The oldest theme in the history of videogame design is that technological constraints become artistic strategies. Tomohiro Nishikado (whom we will discuss again later) turned an unexpected property of a microprocessor into the foundation of all mainstream videogame design. In a similar fashion, the designers of *Diablo II* took procedural generation and permadeath (two ideas instituted partly because of technical constraints) as they appear in *Rogue* and roguelikes and turned them into artistic resources. One of the most important things to know about *Diablo II* is that it plays with its source material and implements it in an idiosyncratic way. In essence, the designers of *Diablo II* saw that roguelikeness was not a binary thing; games can be somewhat roguelike. The most obvious example of this is the fact that the *Diablo* games take place in real time, whereas most of its roguelike predecessors are turn-based. There are plenty of other examples, however. Player death in *Diablo II* is clearly based on its roguelike antecedents, but with some adaptations for a wider audience. When a player-character dies

(on softcore setting), that character loses gold and experience points. Some of that can be retrieved if the player can reach his or her own corpse, but some of it is irretrievable. Thus, death is penalized—sometimes quite heavily—but in a way that's more appropriate for a mainstream audience. Then again, *Diablo II* also has a hardcore setting which reinstitutes permadeath if the player wants the extra challenge. Procedural generation is also something the designers play with rather than employ entirely. Traditional roguelike dungeons are almost entirely algorithmic, whereas many locations in *Diablo II* actually have a great deal of fixed terrain and fixed enemy locations—far more than most people imagine.[4]

Some of the ways in which *Diablo II* deviates from the roguelike formula are actually very informative about the degree to which the designers wanted to recapture the old roguelike spirit. In early roguelikes, item identification is an important bottleneck that prevents players from being able to use just any item they find. Not only are many items unidentified when the player picks them up, but the item identification scroll itself is an unidentified item that needs to be used blindly in order to determine what it is. This kind of volatility and blind chance was one of the things that David Brevik said he liked about early roguelikes, but which didn't fit the voice of the *Diablo* games.[5] Indeed, not only was that kind of blind volatility diminished, it was actually reversed in some of the game's mechanics. In *Diablo II*, Deckard Cain can mass-identify an entire inventory's worth of items.

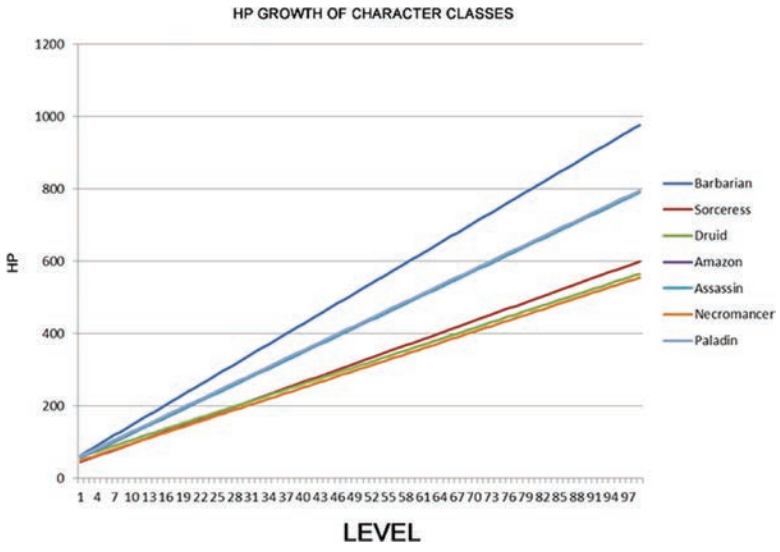
It would be easy to see this change as modern developers dumbing down their product to reach casual players. This is an assumption game critics have made for as long as there have been identifiable videogame genres. But, like many RPGs of its time, *Diablo II* is actually dropping one idea to embellish another. They wanted to widen their audience, yes, but they also wanted to speed up and expand the game.

By eliminating the identification bottleneck, *Diablo II* is able to center the game's focus on the item acquisition loop. There was nothing accidental about this. Brevik

wanted to recapture the experience of playing Angband and Moria, two roguelikes he enjoyed in college. He remembered that he and several friends would play in the computer lab together (even though these were single-player games), and they would shout to one another to come and look when a rare and powerful item would drop. "You basically never actually killed the balrog," he said. "You would play for those really rare items instead."[6] Max Schaefer said something similar, saying that his focus was in creating powerful items that allowed players to have differing experiences of the game.[7] Although permadeath and other roguelike design ideas were important to the designers, the primary experience that they wanted to recreate was the ecstatic discovery of a very rare item that caused a sudden swing in difficulty.

The Legacy of the Tabletop: Character Classes

Another way that *Diablo II* diverges from its roguelike ancestors is in the use of character classes. Although the character class was one of the foundational ideas of the RPG, the roguelikes from which *Diablo II* borrows many of its core ideas did not use character classes in a significant way. This was probably a product of their idiosyncratic reduction in scope. *Diablo II*, despite having a comparatively massive budget, mostly sticks to the reduced scope that it inherited from the roguelike. Real-time gameplay doesn't really take *Diablo II* away from the volatility, danger and endless (but narrow) variation of the roguelike, it just recasts those things in real time. Character classes, however, are a major deviation from the roguelikes that directly inspired the game. Every character class feels distinct, and their differences are reflected in the underlying systems of the game. Below is a graph that visualizes the primary statistical difference (HP) between classes in *Diablo II*.[8]



Although the most obvious trend is the divide between fragile casters and hearty melee fighters, the classes exist on a spectrum rather than in two undifferentiated lumps. There are quite a few differences between classes, although we're only going to examine the statistical parts in this section. In the next section of this book, we will also examine the action-game differences between classes, like differences in mobility and crowd control skills. For now, we're going to focus on simulated skill systems (RPG stats) and the few class-based items the game employs.

Historically, character classes in Western RPGs have been distinguished by the abilities they possess and the gear they equip. *Diablo* adheres to the former criterion, but not the latter. Each class features distinct abilities, especially in their passives. When it comes to gear, *Diablo II* experiments with a small amount of class-based gear, but most of it is usable by any class. In this section, we're going to primarily examine how character abilities distinguish each character class, and how that is a product of the traditional tabletop RPGs that influenced its designers. Most of this section will be about traditional RPG abilities and the RPG systems that support them. Although the character classes of *Diablo II* resemble their tabletop antecedents in many ways, there's a big difference beneath the surface. In tabletop RPGs, some characters hardly deal any damage to enemies at all. Instead, they heal, protect, enhance, obscure, redirect, negotiate, and do many other things. In *Diablo II*, all characters are constructed to slay hordes of enemies. Barbarians and Paladins are much better at tanking than Sorceresses and Necromancers, but they have to be able to slaughter the demonic hordes by themselves, or else *Diablo* loses its unique voice. This is one of the primary reasons why this book contains so little about multiplayer mechanics. As more players join a *Diablo II* game, the HP, experience values, and item drop rates increase in a way that's very favorable to the player, encouraging large parties. However, there are no statistical changes to enemies that would require the tank/DPS/heal balance. A team of eight Sorceresses will perform just as well as a more balanced team, and that's intentional.

Character Systems Overview

Before we get to each class, I want to unpack some of the systems that those classes share. There are three different dynamics that differentiate character classes: base stats, elemental availability, and mobility skills. Base stats are the numbers that make all of *Diablo II's* simulated skills work. Base stats are things like strength, dexterity, energy, and vitality. These stats give characters physical damage and carrying capacity, hit rating and evasion, mana and HP, and a few other benefits besides. Every time a character gains a level, the player can invest five points into one of these base stats, but depending on character class, the benefits will be different. The player also invests in the base stats of skills, putting points into offensive spells, mobility skills, and passive abilities to raise their effectiveness. These too differ greatly between classes—and sometimes even differ greatly between different builds of the same class. Finally, each class has a unique elemental profile. That is, each class has access to a few types of damage, but no class has access to all the types.

1. The Different Kinds of RPG, and How *Diablo II* Borrows from Them

Base Stats

Like all RPGs, *Diablo II* is built on a system of simulated skills. A simulated skill is any skill that a character learns, rather than the player who controls that character. In *Mortal Kombat*, for example, the player has to learn and perform a button combination in order to use a fireball attack. In *Diablo II*, the Sorceress can learn the fireball attack at level twelve if the player chooses it. After that, the player only has to point and click. Because *Diablo II* is an action game, there's some skill in aiming the fireball and other skills like it, but making the fireball more powerful is mostly a matter of stats rather than skill. (There are a small number of skills that really benefit from action mechanics, but we'll get to them in the next section.) Fortunately, *Diablo II* offers quite a few different ways to augment the basic stats of a character or character's skill, which is why these stats merit their own section.

Vitality and Energy

The most basic form of simulated skill in *Diablo II* is the stat point. Most RPGs have stat points that a player can invest in a character, either by random chance or out of a budget. *Diablo II* gives the player points to invest out of a budget earned after a character gains a level. Most of its stats closely resemble those that were originally invented in *D&D* with a few minor differences. For example, the vitality stat in *Diablo II* closely resembles the constitution stat from *D&D*. Investing points into either stat will allow the character to gain more HP. The exact implementation is different—points invested in vitality directly affect HP in *Diablo II*, whereas in *D&D*, there is also some randomness in the process. The fundamental idea is, however, very similar. Energy does for spell resources (the mana pool) what vitality does for HP. The *Diablo II* energy stat doesn't have a great analogue in *D&D* for this reason. In *D&D*, the character has to rest for a certain amount of time before replenishing his skills. In *Diablo II*, the player has to use a potion or a skill that regenerates points. Also, energy stat has no effect on the power of spells. This is relatively rare—even in games that allow direct investment of points into skills, there is usually a separate character stat which also contributes to the power of some spells. In *Diablo II*, this is only true of skills with a physical component. Spells in *Diablo II* only gain power from skill-specific points.

Strength and Dexterity

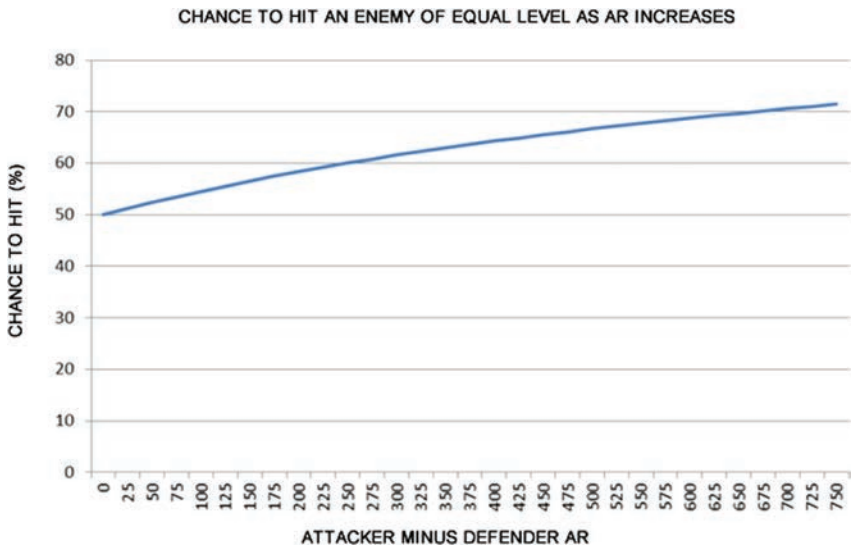
Strength and dexterity in *Diablo II* are also close to their *D&D*/roguelike ancestors, with one small change that has some huge consequences. Strength allows characters to wield heavier armor and heavier weapons, and to deal more damage with melee attacks. All of that is totally orthodox RPG design. Dexterity plays a huge role in hit rate for physical attacks—both melee and ranged attacks. This is also orthodox to the tabletop games that existed at the time of *Diablo II's* release. The only real deviation from contemporary tabletop stats is that dexterity also governs ranged damage. We'll cover this in more depth in the section on hit rates, but the basic idea is that dexterity is twice as useful for ranged attackers as strength is for melee fighters. Not only does dexterity help their damage, it also

makes their attacks more accurate at the same time. Because ranged attackers get to "double dip" in dexterity, many ranged attacks are balanced (one might say penalized) in a way that melee attacks are not. In theory, this is a perfectly acceptable tradeoff, but in execution there are some oversights. We'll examine those abilities in the character sections, but it does seem like the genesis of this problem is the fact that ranged attackers get two bonuses.

The only really unorthodox dynamic in the implementation of stats in *Diablo II* is the way that stat points confer different amounts of benefit for different characters. In most RPGs, a single point in any given stat confers the same amount of general benefit, regardless of class. There may be class-based abilities that apply higher or lower multipliers to those stats, but basic rolls depending on a stat like wisdom or charisma will be the same for any two character classes who have the same value in that stat and no other modifiers (like feats). In *Diablo II*, the returns on HP and mana investment differ significantly across the characters. These returns are intuitive, however; the Barbarian gets twice as much HP from vitality as the Sorceress, and the reverse is true for mana and energy. The other characters are somewhere in the middle.

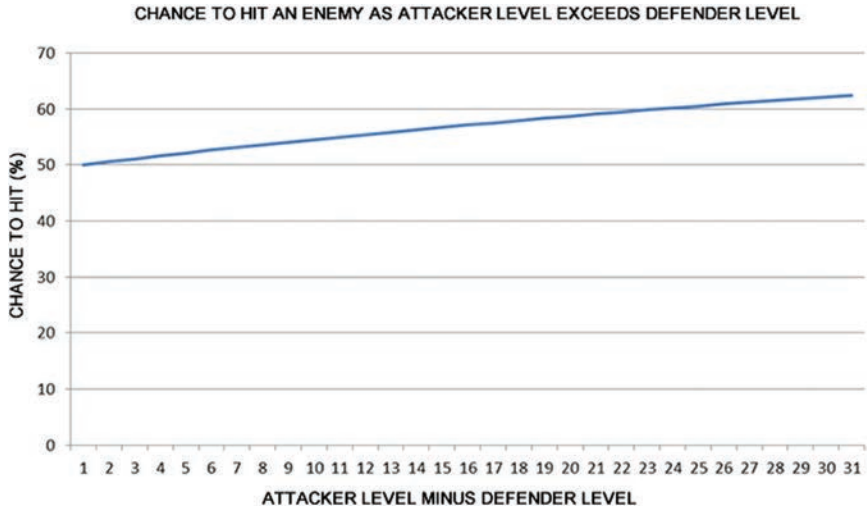
Dexterity, Level and Hit Rates

In *Diablo II*, magical attacks have a 100% hit rate if the spell animation comes in contact with the targeted monster's sprite. Physical attacks only hit if the player has a sufficiently high level, a sufficiently high hit rating, and a little bit of luck. The single most important determining factor in hit rates is the attacking character's hit rating, measured against the defending monster's defense rating. Below is a graph showing how increases in hit rating increase the overall chance of hitting a monster of an equal level.[9]



1. The Different Kinds of RPG, and How *Diablo II* Borrows from Them

Investing five stat points (one level's worth) gets a character 25 points of AR, which is the subject of the x-axis. Character level is the other factor under the player's control that affects a character's chances to hit with a physical attack. Below is a graph that shows how gaining levels affects hit chances if attack rating remains static.[10]



The slope of the line in the second graph is shallower, although it can be hard to tell just by looking. Even after gaining 30 levels relative to the character's target monster, the chances of hitting only go up by about 12 percentage points. Still, player characters will have a hard time hitting enemies that are a higher level than they are, and an easier time hitting enemies that are a lower level. Although it is never spelled out anywhere in the UI, one of the ways that character classes differ is in the way that level is applied to their hit ratings. Certain classes get bonuses to their "effective level" when making a physical attack. For example, a level 15 Barbarian making a melee attack is treated (for the purposes of hit rating) as though he were actually level 35. That's usually worth about nine percentage points in the chance-to-hit rating (Table 1.1).

Table 1.1 Melee Bonus Levels[11]

Character Class	Bonus Levels When Melee Attacking
Barbarian	+20
Paladin	+20
Assassin	+15
Amazon	+5
Druid	+5
Necromancer	−10
Sorceress	−15

These bonuses mostly make sense. The three characters that tend to fight up close the most often, the Barbarian, Paladin, and Assassin, all get significant bonuses to their effective level. The casters, meanwhile, are penalized when attacking since they really should not be doing so very often. It's the middle two classes, Amazon and Druid, that have problems, but we'll see about those in the respective class situations.

Dexterity, Level, and Block Rate

Dexterity also contributes to defense and blocking, two simulated skills that serve the same ostensible purpose, but which operate independently of one another. Both defense and block rates are responsible for deflecting enemy physical attacks so that they do no damage. Neither has any effect on spells, which are instead governed by elemental resistances. Defense works exactly the same way for the players as it does for the enemies, except that players get to factor dexterity into the equation. The contribution of dexterity isn't much, however.

$$\text{Base Defense Rating} = (\text{Dexterity}/4)^{[12]}$$

Even Amazons and Throw-Barbarians (who put lots of extra points into dexterity) only get the same amount of defense from their dexterity stat that a Paladin gets from a decent helm or boots. All that dexterity isn't really helping the defense stat much. Moreover, defense itself is not that useful. Defense rating only works while the player character is walking or standing (not running), and it's very difficult to get a high enough defense rating for it to help much after level 60. With a skill like the Barbarian's Iron Skin or Paladin's Defiance Aura, defense can be augmented to the point that it offers real protection. Getting a significant amount of defense from items and dexterity alone requires some serious item farming, but we'll cover that in the third section of this book.

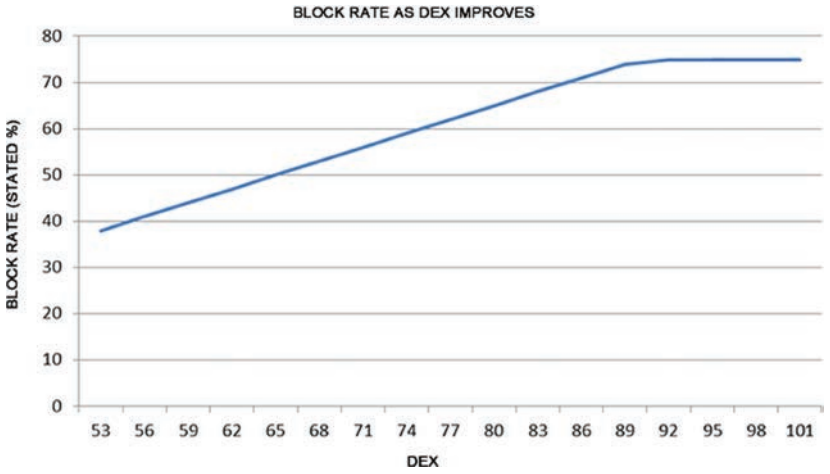
The block skill is much more likely to provide significant protection to a player, although it has its drawbacks. Any character that equips a shield (or an Assassin with the Weapon Block skill) can block incoming physical attacks. The primary

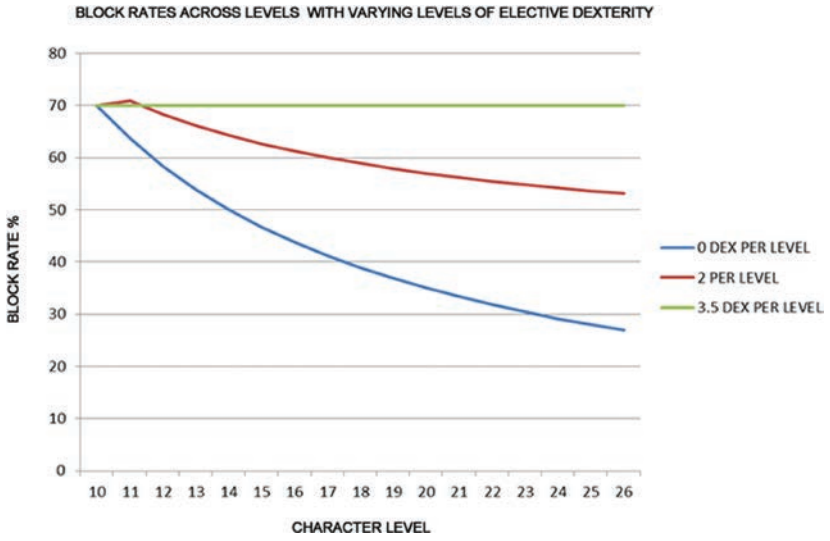
1. The Different Kinds of RPG, and How *Diablo II* Borrows from Them

contributor to the block rating is the shield's own block stat, which is separate from that shield's defense rating.

The other component in block rating over which the player has control is dexterity. Dexterity improves block rates a lot more than it improves defense, although the required investment is high. Block rate maxes out at 75%.

Below there are two graphs because there are two important dynamics to examine. First, it's a lot easier to reach 75% block by investing in dexterity than it is to reach 75% evasion (armor) rating via defense via any means. In the first graph, the player-character's level is static. The second graph shows what happens to block rate as the player character gains levels.[13,14]





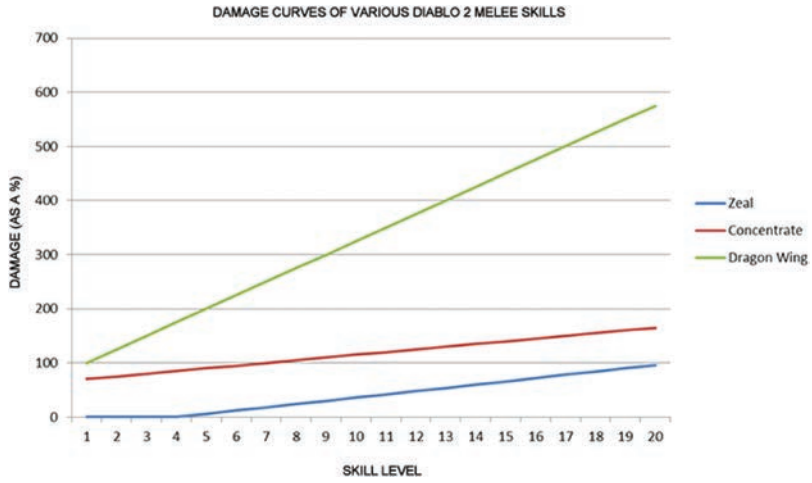
Block rates actually go down as the player gains levels unless the player is investing heavily in dexterity. For some dexterity-heavy characters, this isn't a huge problem, but for classes that aren't stacking tons of dexterity, this dynamic forces players to find higher-level shields whose higher block rates can mitigate the lack of dexterity. (Really, all classes benefit from higher-level shields, but the need is greater for classes that don't invest much in dexterity.) Indeed, the class-based differences in shield usage run deeper than dexterity investment. The classes themselves have block rate modifiers that operate similar to the hit rating modifiers. Any given shield has a 10 percentage-point spread for its block rate. For example, if the player equips their character with a Small Shield, the caster classes (Sorceress, Necromancer, and Druid) have a base block rate of 20%, the Paladin has a 30% base block rate, and the Assassin, Amazon, and Barbarian will fall in between at 25%.

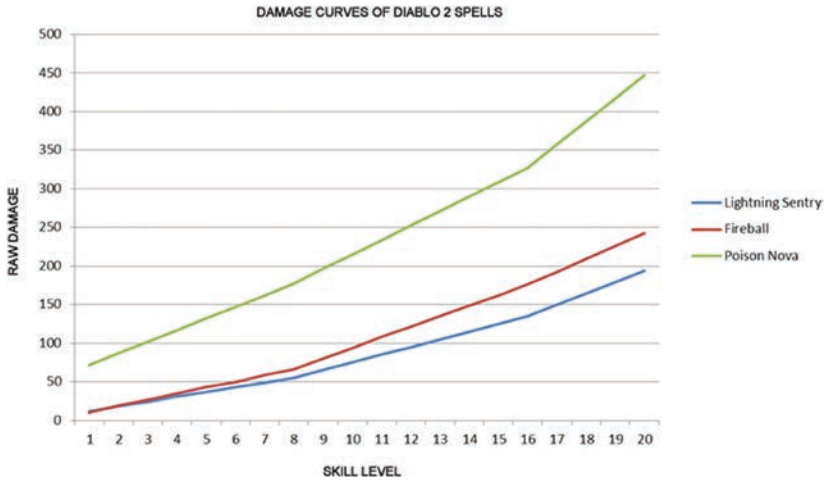
Skills

In *Diablo II*, skills themselves have levels that the player can improve. Every time a character gains a level, they gain one skill point that can be invested into any unlocked skill. This is not unique in the history of RPGs; even contemporary games like *World of Warcraft*, *The Witcher*, *Skyrim*, and the *Deus Ex* series use this mechanic. What's unusual about *Diablo II* is the number of times a player can invest in a single skill. Every skill in the game can be leveled up 20 times, although many skills make for very poor investments.

1. The Different Kinds of RPG, and How *Diablo II* Borrows from Them

The results of each point are fairly significant as well. Below the next page I have visualized the growth in damage of some of the major skills from a few classes.[15,16]





The growth is mostly linear, although spells have a little bit of increased slope at the end. The most natural response to this graph is to wonder if the linear growth of monster HP will simply neutralize any gains the player character makes in skill power. The answer is complicated. For the first difficulty setting, and a large part of the second difficulty setting, focused investment in the best skills will give the player increased killing power. In the second half of the game, deficits start to eat away at the player's marginal killing power. At that point, the player has to synthesize gear, skills, and good strategy, but that is the domain of part three of this book.

Elemental Availability

The last major, systemic difference between the character classes is the types of elemental abilities available to each. Including physical-type damage, there are six elemental attributes an attack can have in *Diablo II*. Each class has access to a few elements; none has access to all of them (Table 1.2).

Table 1.2 Elemental Availability Across Character Classes

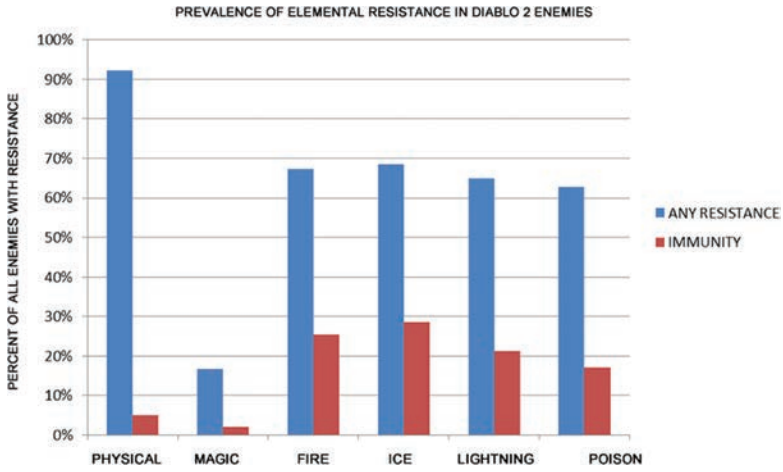
Class	Fire	Cold	Lightning	Poison	Magic	Physical
Amazon	Yes	Yes	Yes	No	No	No
Assassin	Yes	Yes	Yes	No	No	Yes
Barbarian	No	No	No	No	Yes	Yes
Druid	Yes	Yes	No	Yes	No	Yes
Necromancer	Yes	No	No	Yes	Yes	Yes
Paladin	No	No	Yes	No	Yes	Yes
Sorceress	Yes	Yes	Yes	No	No	No

1. The Different Kinds of RPG, and How *Diablo II* Borrows from Them

(Note that this table is made up of *viable* builds. The Assassin has magic damage in the Shadow Arts but doesn't kill with it. The Assassin and Amazon both have poison damage as well, but their poison damage skills are almost universally deprecated, and are not counted.)

You can see how the classes exhibit a great deal of variety in terms of elemental availability, and how there's not a lot of overlap. Six elements are more than enough to make seven classes (and more than a dozen possible builds) distinct in their elemental strengths and weaknesses. Yes, certain classes have a greater ability to split their skill points up between multiple trees than others. The "Meteorb" Sorceress, Tornado/Hurricane Druids, Berserk Barbarians, and Traps Assassins—all of these can mix two elements. Most builds cannot mix three elements without having a very specific gear set, which maintains the game's balance.

In *Diablo II*, the elemental system governs only which creatures or characters take greater or lesser damage from enemy attacks. The balance is mostly orthodox to the RPG tradition, although immunities are occasionally applied in counterintuitive ways. In most RPGs, a creature that is vulnerable to one element is strong against its opposite—icy creatures are vulnerable to fire, and strong against cold, for example. Sometimes that is true in *Diablo II* (certain families of enemy, like the Fallen, share fire resistance), but often elemental strengths and weaknesses don't make any intuitive sense. Thanks to the procedural addition of monster affixes, an enemy can spawn completely immune to fire and ice damage alike. While any unique enemy can spawn with almost any elemental modifier, most of the elemental resistances in the game are fixed. Below are graphs that visualize the prevalence of resistances on the highest difficulty setting—the point at which elemental resistances become common.[17]



There are a couple of important trends visible here. First, immunities to non-elemental magic and physical damage are much less common than resistances to fire, cold, and lightning. This dynamic is in proportion with the prevalence of extremely powerful fire, cold and lightning attacks in the game, however. These three elements make up 49% of all types of spells or attacks the player can use. But what about resistances that don't reach the level of immunity, and yet are still widespread? The data are clear: physical resistance is extremely common on the highest difficulty setting (although all resistances are higher on that setting). To a certain degree, this makes sense. Physical damage is the most common type of damage; only the Sorceress class lacks a viable build which depends on physical damage. But there seems to be a huge oversight with regards to this dynamic. Physical damage is already subject to another form of mitigation. Skills which deal physical damage are almost always subject to attack rating. This is a disadvantage already; fire magic might be resisted by monsters, but those spells don't miss unless the player aims poorly. Although there are builds that make physical damage viable in late-game *Diablo II*, those builds are a lot harder to make and don't offer much in the way of compensation for that fact. This is a flaw that we'll see come to particular prominence in the bow-using amazon.

Synergies

At high levels, most classes rely on a small number of skills to deal the bulk of their damage. If those skills can only receive 20 skill points worth of investment, what does the player do with surplus skill points? The skill synergy allows one skill to enhance another, allowing players to invest skill points in a way that still enhances their primary skills.

1. The Different Kinds of RPG, and How *Diablo II* Borrows from Them

When it comes to synergies, the big difference between classes comes down to the linearity, number degree of their synergies. A linear synergy is any synergy that takes place linearly across a skill tree. For example, the Sorceress's Blizzard skill has synergies from the Frost Bolt, Cold Bolt, and Glacial Spike; those three skills are prerequisites for learning Blizzard, so they are "in line" with the skill. When the player finally does invest in Blizzard, the Sorceress already has a minimum of 15% bonus to its damage. The second important aspect of synergies is their number and degree. Blizzard benefits from three direct synergies, with an average amount of 5% damage per point invested. (I.e., Blizzard has three linear synergies, and their degree is 5% per point.) This means that a player investing all available points into Blizzard synergies might gain as much as a 300% bonus to the damage of the spell. Itemized skill bonuses don't contribute to synergies, but skill damage above level 20 is a little steeper in terms of damage growth, making synergies more valuable. Thus, certain classes have huge advantages in damage because of the linearity and degree of their synergies.

The impact of synergies on character classes is not always strictly quantitative. For example, several of the Barbarian's combat abilities rely on synergies that are non-linear. Both Concentrate and Berserk receive significant amounts of magic damage from skills in the Warcries tree, which is completely independent of the Combat Abilities tree. The Barbarian really does need those Warcries for buffing and crowd control purposes, however. While the Barbarian can't stack points into linear synergies the way that a Sorceress can, his synergies encourage the player to invest in skills with great utility value. This is part of the Barbarian's design philosophy, which requires the use of a movement and crowd control skill in conjunction with a main attack skill. Players tend to build their Sorceresses around one big skill, and then focus on inflating the damage of that skill. For the Barbarian, that strategy doesn't work quite as well, and the synergies reflect it. Similarly, the Necromancer's Curses skill tree lacks synergies entirely. Even the strongest curses won't kill most enemies, so the player needs to pair them with another offensive skill. Although the Poison Nova skill gets synergies from other poison skills, its best complement (what you might call a "soft synergy") comes from the Lower Resist curse, which allows it to do a lot more damage than it normally would. Accordingly, all analysis of a class's synergies will try to take into account what those synergies (or lack of them) mean for the character.

As a last note before diving into the character classes, I want to point out that I have only analyzed the most common builds of each class. I don't delve into the mechanics of unorthodox (but viable) builds like the Double Throw Barbarian, the Whirlwind Assassin or the "Dentist" Necromancer. Even without them, most of the mechanics of each class are represented by the builds I do cover.

Character Classes

Amazon

Builds: 2 – Bowazon (uses bows and the bow/crossbow skill tree) and Javazon (uses javelins and the javelin skill tree)

Bowazon Build

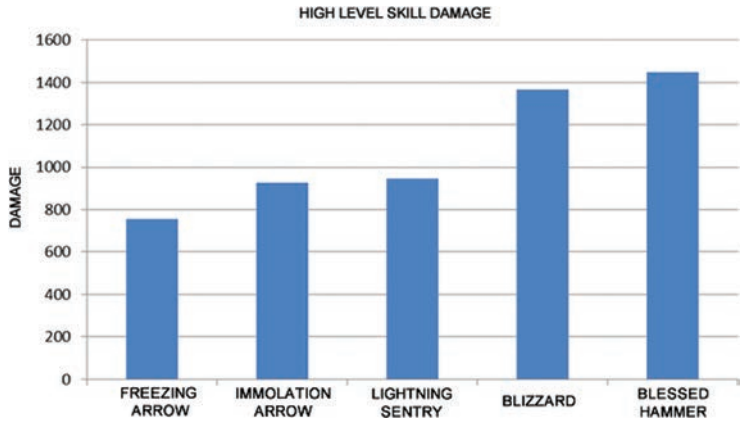
Build strength: Long range, two benefits from dexterity
Build weakness: Low damage, no spells, too many skills to invest in
Elemental availability: Moderate
Synergies: Poor, top-level skills only gain moderate damage from one other skill, and there aren't enough skill points to spare to take advantage of it
Overview: The Bowazon is built on classical RPG checks and balances. Unfortunately, *Diablo II* is not the kind of game that relies on those things, and so the Bowazon ends up a little underpowered relative to her peers.

The great, intrinsic advantages of the Bowazon are her long range and the fact that two benefits are gained from dexterity. As we'll see with several classes, ranged attacks in *Diablo II* are not all equal in effective distance. The Paladin has to be at the center of his target area when casting the Blessed Hammer. The Druid has to be at the center of his Hurricane. Even the Javazon has to be fairly close to reliably hit with her Lightning spells. The Bowazon, like the Traps Assassin and the Sorceress, can sit back at quite a distance, and enjoy the safety that distance affords.

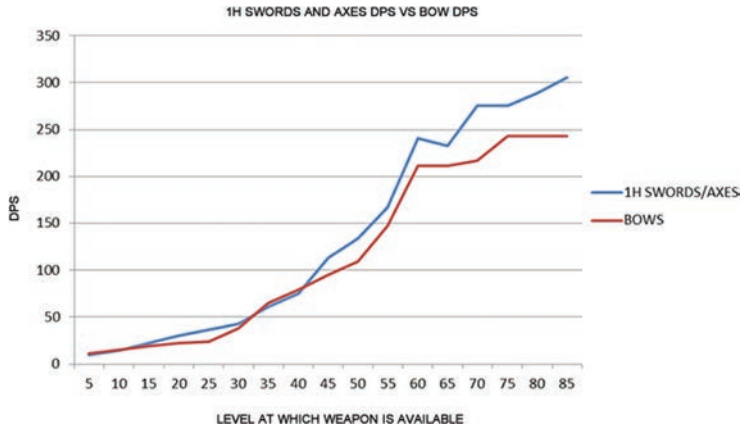
The Bowazon even gets to add the extra protection of having the beefy Valkyrie minion to distract enemies. At high levels, the Valkyrie is hearty enough that it can even tank bosses for a reasonable amount of time, although it has trouble controlling boss aggro. (Technically, the Javazon can also employ a Valkyrie, but the greater range of the Bowazon makes the Valkyrie a little more useful for the latter build.) The Bowazon's other advantage is her ability to gain two benefits from dexterity. As we saw in the section on dexterity, the stat's contribution to hit rating is considerable if the player chooses to stack a lot of it. Because the Bowazon's damage output is also tied to dexterity, she can spend most of her points on it. The problem that we'll end up seeing is that this double advantage simply isn't as strong as it needs to be in order to make up for other weaknesses.

1. The Different Kinds of RPG, and How *Diablo II* Borrows from Them

The Bowazon's biggest weakness is that her skills and weapons aren't powerful enough. The big three skills a late-game Bowazon might use, Freezing Arrow, Immolation Arrow, and Strafe simply don't do that much damage.[18]



I've visualized the damage of several high-level ranged skills, adding in the main synergies for them. The Bowazon skills come out lower. The other skills visualized are spells. Unlike Bowazon shots, spells never miss (as long as the animation connects with the enemy sprite). To be fair, because the Amazon skills are attacks, they also gain weapon damage not visualized here. But physical resistance is the most common type of resistance, and bows are just not that strong. Below and to the right is a comparison of some bows and melee weapons.[19]

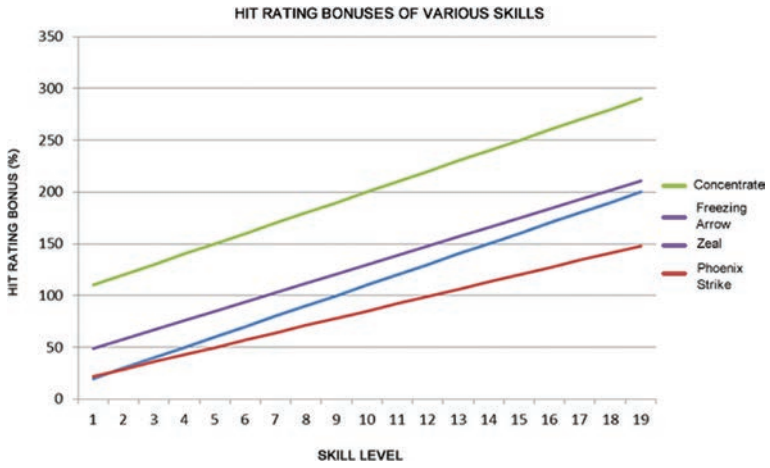


Note that the melee weapons are *one-handed weapons*, and they still surpass bows. That means melee characters are doing more DPS with one hand than Bowazons

are with two, meaning they can equip a shield or another weapon. There are definitely bows in the game that make up some of the damage gap, especially through means of very high attack speed, but casters get elite weapons that offer +skill affixes and faster cast rate modifiers, too, so the gap remains.

The real problem is that bows are the victim of traditional balance mechanics that are out of place in a game like *Diablo II*. Ranged weapons face a penalty for the safety they provide by firing at long range. In an orthodox tabletop RPG, they should incur a penalty; that balance is a basic tenet of good design. In the RPGs from which this tenet comes, however, bow-using characters aren't responsible for killing hordes of demons by themselves. In those games, bow-users tend to pick the most dangerous targets and eliminate them tactically while the tanks and melee fighters kill everything else. That's not a viable strategy against the demonic hordes of *Diablo II*. As we've seen, both amazon bows and amazon abilities fall short of their peers. The most elite bows in the game do enough damage to make the Bowazon comparable to other classes, but obtaining them involves killing hordes and hordes of enemies—which is hard to do with her relatively lower power!

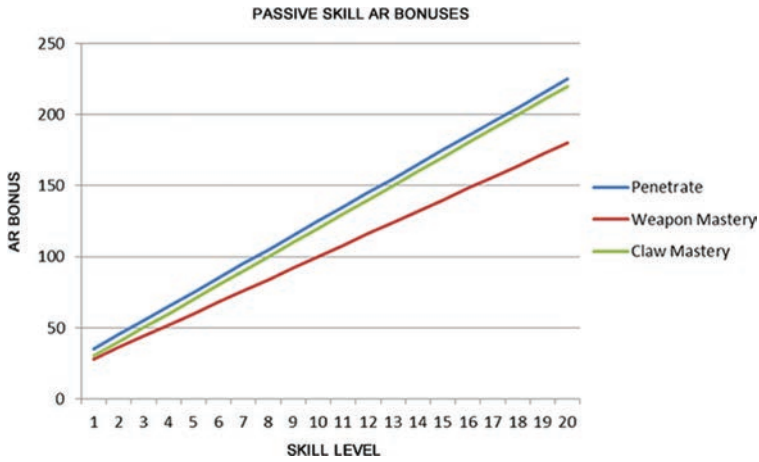
There are two places in which the Bowazon might catch up to the other classes: hit rating and in utility/defense skills. All of the Bowazon's damage skills are attacks, and are therefore subject to attack rating (except for Guided Arrow). But, for all their dexterity stacking, amazons don't hit significantly more often than other classes. The biggest reason for this is that neither their weapons nor their skills add that much to attack rating. As we saw in the section on attack rating, the amazon has a lower effective level when striking, losing her about fivepercentage points of hit rating vs the melee classes. Her dexterity provides a nice boost, gaining her back about 14 percentage points. After that, it's all up to weapons and skills.[20]



Her skills do not provide her with a significant advantage versus other classes. Moreover, they provide a significantly lower damage multiplier than those other

1. The Different Kinds of RPG, and How *Diablo II* Borrows from Them

skills, especially when we remember that those melee classes are using better weapons. There are still passive skills to consider, however.[21]



Penetrate keeps up with the other skills, but it doesn't outpace them by an amount that would balance its big flaw. What's not shown here is that Barbarian Weapon Mastery, Paladin Fanaticism, and Assassin Claw Mastery all also provide the character with damage and/or attack speed in addition to hit rating. Penetrate only provides attack rating. Comparatively, the amazon bow passive is just too weak. The designers have simply overcompensated for the Bowazon's range in a way they don't do for Javazons, Traps Assassins, Necromancers, and Elemental Druids.

In theory, all of this could be avoided if the Bowazon's skills gave her more options than other classes have. It seems that this was the designers' intent, but in practice it doesn't compensate for her other shortcomings. The Amazon has an entire skill tree of passive and magic skills that could complement her bow skills reasonably well. Skills like Slow Missiles, Critical Strike, Penetrate, Pierce, Evade, and Dodge would be great if the Bowazon could afford to invest in them heavily. She cannot. Bowazons need to invest 20 points into their main elemental arrow and another 20 into Strafe if they hope to deal any significant damage at all, and then another 20 into Valkyrie. Most players also like to invest in utility skills like Pierce or Guided Arrow. With the prerequisites for those skills, the build barely has any spare points to invest in defensive skills or Critical Strike, and that just isn't enough. Perhaps this is where the Bowazon's class-specific items are supposed to come into play. Even as normal-quality drops, the Amazon-only bows put points into all Bow and Crossbow skills. Is the player supposed to save points in bow skills and reallocate into passive skills? It wouldn't work. The damage and attack rating bonuses in bow skills are too low, so the player must max them out. It appears to be yet another case of over-balancing the Bowazon, forcing her to rely on the elite items that every class wants, but not every

class needs. These problems are not fatal; the build is still playable, but for the most part, other builds avoided these kinds of systemic pitfalls.

Javazon Build

Build strength: Medium range, two benefits from core stat, massive splash damage
Build weakness: Almost all damage comes from one elemental type
Elemental availability: Poor
Synergies: The lightning skills are all part of a big linear synergy that makes several of them very powerful
Overview: The Javazon has serious limitations, but these limitations are appropriate to the *Diablo II* context. The Javazon is ranged and relatively fragile, but also very powerful. Her big limitation is that almost all of her damage comes from lightning-elemental attacks, meaning that roughly 20% of enemies on hell difficulty are simply immune to the damage she does.

By making one change, the Javazon build corrects the shortcomings of the Bowazon build. Several of the best Javazon skills are spells, and therefore always hit. Thus, the Javazon doesn't have to worry about attack rating as much, nor the extremely widespread resistance to physical damage. The real drawback for the Javazon is that she deals almost all of her damage as lightning, and therefore simply cannot kill enemies who are lightning immune without having rare, resistance-removing gear, or companions who can do that for her. This limitation is one that makes sense for *Diablo II*, however. The Javazon has a strength she can exploit, and a weakness she needs to avoid. The Bowazon is somewhat mediocre across the board. For instance, one solution to the elemental availability problem is to simply avoid lightning immunes; there are plenty of areas in which this is possible. The other solution is to journey with a companion who can lower enemy resistances, like a Necromancer or Paladin. This is exactly the kind of limitation *Diablo II* ought to employ, since the game was explicitly designed to be a multiplayer affair. This doesn't work for the Bowazon, however. There's no location in the game where having underpowered attacks isn't a problem, and there is no party composition that can make a Bowazon more valuable than some other class might be.

It may be the case that the Javazon's greater power is a product of a design problem which the designers knew about but had no solutions for. Like the Sorceress, the Javazon relies on multi-target attacks, especially the tremendous Lightning Fury. Lightning Fury doesn't actually do a ton of damage on its own; most of its strength comes from the number of targets it can hit at one time—more than 20 at higher levels. Lightning Fury also seems to be the "other side of the coin" for Bowazon skills. Although it hits like a spell, Lightning Fury depends on attack speed rather than casting speed. Increased attack speed is an attribute found on items in much greater abundance than faster casting speed. Bowazons benefit from faster attack speed, too, but because that build uses weaker attacks that often miss,

1. The Different Kinds of RPG, and How *Diablo II* Borrows from Them

this isn't as helpful. For a gigantic explosion of lightning that mows down crowds, increased attack speed is incredibly helpful, and fairly easy to find.

Lightning Fury shares lots of synergy bonuses with several other skills in the lightning tree, making the Javazon build one which is hard to mess up.

You can see how those big synergies allow several skills to be useful against single targets, while Lightning Fury is useful against crowds. The big linear synergy also allows the Javazon's class-specific items to provide considerably more benefit. By adding points to every skill along that linear synergy, the Javazon-specific weapons also trigger all the interlocking synergies. Why couldn't this have been the case for the Bowazon build? Even if the Bowazon fires from longer range, her power isn't reduced proportionally to the advantage that range gives.

Assassin

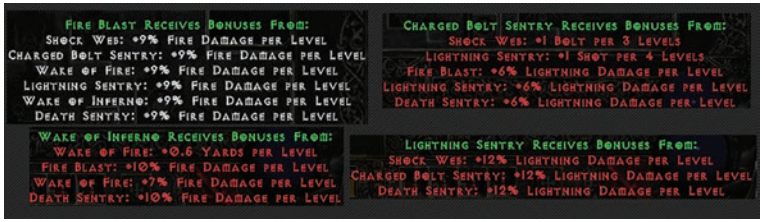
Builds: 2: Trapsassin, kicksassin

Trapsassin Build

Build strength: Long range, spell damage
Build weakness: Traps can be hard to place in certain tight corridors; fire damage has an effectively shorter range than lightning damage
Elemental availability: Moderate, can deal great fire or lightning damage
Synergies: One of the game's few multi-element linear synergies gives the Trapsassin build some very high damage
Overview: The Trapsassin fires at long range and has two big sources of elemental damage. Her ability to summon a minion also protects her much like the amazon's Valkyrie protects that class. The Trapsassin, however, uses spell damage. Like the Sorceress, she can sit back and let her spells melt enemies she can't even see.

The Assassin who specializes in the Traps skill tree is like the Necromancer or Sorceress, who can easily hit enemies who aren't even on screen. Traps can be placed around corners and they cannot be targeted by the enemy. They can attack independently of the Assassin's primary target, which is a blessing and a

curse—sometimes they'll fire on a target other than the one which is the most dangerous. Still, though, their impressive range means that the Trapsassin can usually sit back comfortably while her spells destroy the enemy. This build also has easy access to two elements with enough damage to handle endgame content. This damage depends heavily on the large linear synergies in the traps skill tree.



The synergies for every skill are abundant, but of particular note is the skill Fire Blast, which can gain up to 700% damage increase from its synergies. That skill deals fire damage, and yet it benefits from a huge linear synergy made up of both lightning and fire skills. Thus, the Trapsassin can deal two types of damage in amounts viable for the endgame.

Like staves, wands, and maces, the claw class of weapons frequently comes with +skill modifiers, even on gear that can be purchased from low-level vendors. Unlike those weapons, however, these weapons usually come with category bonuses rather than specific skill bonuses.

Some magical staves, wands, and Necromancer off-hand items have categorical bonuses, but it's a lot more common on claws to see +1 to a skill tree or all

1. The Different Kinds of RPG, and How *Diablo II* Borrows from Them

Assassin skills on blue items. The prevalence of categorical bonuses is probably a product of the Assassin's huge, interlocking synergy tree, or the many combos that Kicksassins have to use.

Kicksassin Build

Build strength: Lots of types of damage, finishing moves with a variety of practical effects

Build weakness: Not as rugged as Paladin or Barbarian, defensive skills require more micromanagement than in other melee classes

Elemental availability: Very high—significant amounts of cold, fire, lightning, or physical damage, plus the attack rating bonuses to actually hit with them

Synergies: No big linear synergy chains, but skill synergies mixed with passive skill synergies power the Kicksassin up considerably

Overview: The Kicksassin is a melee fighter with a lot of different attacks to use, many of which have elemental components. Her finishing moves have practical effects, and the Shadow Discipline skills give her tons of utility on the battlefield, although it can be difficult to switch to those skills in the heat of battle.

The Assassin who specializes in the martial arts skill tree (typically called a "Kicksassin") can fight in close quarters adeptly, although she uses slightly different means to do so than the Paladin or Barbarian. One of the greatest advantages of the Assassin lies in her elemental availability. The top-level Martial Arts skill, Phoenix Strike, can deal significant amounts of fire, cold, or lightning damage. Given that no enemy in the game can be immune to more than two elements at a time, this means that the Kicksassin will always have an attack that can deal some extra damage.

The attack also benefits from some minor synergy bonuses, although the player has to divide skill points between various skills to give each element a synergy boost. The crucial drawback of the attack is that, unlike the skills in the Traps tree, Phoenix Strike is an attack rather than a spell. Thus, in order for Phoenix Strike to hit, the player has to land a physical attack successfully, which may require several attempts at close range. That said, we've already seen in the Bowazon analysis that Assassins get great attack bonuses from Claw Mastery, in addition to gaining damage and critical strike. Moreover, the attack rating bonuses from Phoenix Strike are also great, and combine together to put out damage that is viable for any part of the game. Add to this the practical effects of finishing moves like area damage, teleportation, and major lifesteal and the Assassin can fight just as well as any other melee class.

The Kicksassin's drawback is in its survivability. Although she can equip virtually any armor (as most classes can), her defensive passive abilities aren't enough to allow the Assassin to simply deflect most blows, as a melee Barbarian or shield-using Paladin might do for much of the game. The Assassin's skills help, but she has to rely on some unusual, two-function abilities to do so. Skills like Cloak of Shadows, Burst of Speed, and Fade need to be renewed fairly often, whereas Paladin and Barbarian skills are either inherent or toggled. Burst of Speed and Fade cannot be active at the same time, and so the player has to manage which one is necessary in any individual situation. Similarly, there are spells which help to control crowds, like Psychic Hammer and Mind Blast. The former applies knockback and is useful for keeping approaching crowds away. The latter mind-controls a group of common enemies and makes them fight their allies. Both of these are necessary for good crowd control, but as with the defensive abilities, it can be difficult to switch between them quickly. The *Diablo* UI isn't made for diverse ability usage (this is a topic we'll discuss in the next section) and so the Assassin player has to do more work managing crowds than a Paladin does, for example.

Barbarian

Builds: Not as distinct; Barbarian skill trees don't lock him into skills the way other classes do, except for his chosen Combat Skill. Other than the main Combat Skill, Barbarians can choose from a variety of movement, crowd control, and passive abilities.

Class strength: Very durable, doesn't need to max out many skills to get good use from them

Class weakness: Tends to require use of more skills than other classes do, requires good action mechanics, can run into stalemate situations with enemies who just won't take damage from his physical attacks.

Elemental availability: Poor. He mostly deals physical damage. There are two attacks that deal significant magic damage; one has defensive drawbacks while the other requires major cross-tree synergy investment.

Synergies: The Barbarian's synergies often connect across skill trees, and linear synergies aren't abundant.

1. The Different Kinds of RPG, and How *Diablo II* Borrows from Them

Overview: Regardless of which attack skill he wants to use, successful Barbarians need to diversify their skill set a little more than other classes. Barbarians can rely on one skill for killing, but they need movement, crowd control and buffing skills as well

If the primary tactical dilemma of melee fighters in *Diablo II* is the choice between gear that emphasizes attack or survivability, the Barbarian solves that dilemma by allowing the player to complement his choice of gear with abilities that eliminate the drawback. At first glance, this looks like a bit of bad game design. What's the point of creating a tactical decision if the player can just go around it? There are three reasons why the answer to this question is more complicated than traditional game design maxims might lead you to think.

1. To a certain degree, this question is inappropriate for *Diablo II*. There are plenty of gear combinations in *Diablo II* that can turn any character class into a nearly-invincible killing machine. As such, the Barbarian does not violate any of *Diablo II's* major design principles.
2. In another sense, the question is still pertinent, because all of that ludicrously powerful gear is rare and only found at the end of the game. The Barbarian already has some really powerful passive bonuses at level 30. Although the Barbarian doesn't break the game, he can make a dent in the designer's time table.
3. For all his great abilities, the Barbarian never becomes the best at anything. He has the highest possible defense, but that doesn't mean he's the best at avoiding damage. He has great bonuses to hit and attack power and can deal tons of damage, but that doesn't necessarily allow him to chew through enemies as fast as the Sorceress or Hammerdin can. So really, his lack of obvious flaws is checked by a lack of powerful strengths.

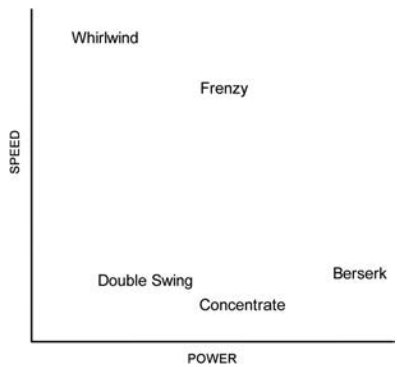
In a vacuum, the Barbarian violates many of the traditional RPG tactical balances. In the context of *Diablo II*, the Barbarian makes more sense.

The central tenet of Barbarian design is that he has to use multiple active skills to survive and kill his enemies. All classes invest in multiple skills, of course, but many casters rely on one skill most of the time, while investing the rest in synergies. The Barbarian tends to use several skills more often. This is somewhat like the Assassin, who has to refresh her defensive buffs and control crowds with her Shadow Disciplines skills, but in this case it's primarily an offensive concern. Nearly all Barbarians will use Battle Orders to double their health in the same way that Assassins use Fade. Many of the Barbarian's crowd control skills are also useful killing skills, with way more damage built into them than the Assassin's similar abilities. Barbarians can knock enemies away with Bash on a one-by-one basis, or they can scatter enemies with the AoE (area of effect) Howl ability. Barbarians can even block off parts of the map by using the Grim Totem skill, which lasts for a set amount of time. On the other hand, most Barbarians can

only deal significant damage up close, and so he needs to close in on his chosen target—even when something else is in the way. Thus, he has several options for getting closer to enemies. He can use Leap or Leap Attack to jump right on an enemy. He can use the Taunt ability to draw enemies to himself, especially in conjunction with Howl.

He can also use the passive skills Increased Stamina and Increased Speed to simply run to his targets faster than other enemies can intercept him. We're going to cover this in a lot more depth in the section on action mechanics, but what should be clear is that the Barbarian has a variety of ways to get around the battlefield—especially when moving towards an enemy.

The Barbarian also has several different ways to do his killing. Regardless of the minutiae of builds, there are two primary choices that the designers set up for Barbarian players. The first decision is whether players want to sacrifice defense to deal damage or not. The second choice is whether players want to use fast weapons or powerful ones. The choices are not binary; players can sacrifice some defense or all of it, and can choose weapons on a spectrum of speeds and powers.

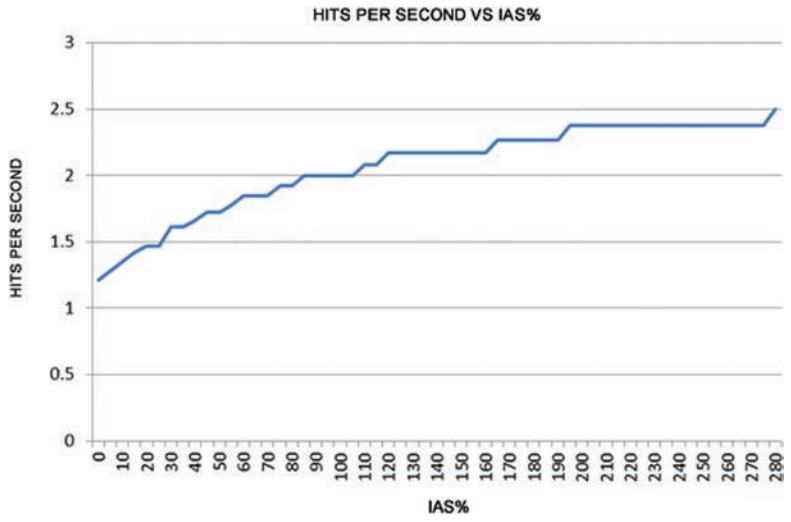


There are a lot of subsequent decisions to make here. Magic damage, the Barbarian's one alternative to physical damage, exists on both ends of the spectrum. With a synergy from Battle Orders, Concentrate can gain magic damage equal to its

1. The Different Kinds of RPG, and How *Diablo II* Borrows from Them

physical damage while still providing significant bonuses to defense. Berserk, on the other end of the spectrum, provides a lot more magical damage, but it sets the Barbarian's defense rating to zero while active. Moreover, Berserk removes all effects like lifesteal (which does exactly what it sounds like) and so removes the best auxiliary defense mechanism a Barbarian might have. Frenzy and Whirlwind, the other two melee attacks that are viable at high level, are in the middle. Frenzy does not directly impair defense, but it does require the Barbarian to set down his shield. Whirlwind also does not affect defense, but the Barbarian cannot use potions while in the middle of it, nor redirect his path.

The other dimension along which Barbarian attacks can be measured is their need for attack speed, as increased by items. This is one part of this document that theorycrafters (ultra-hardcore players who use advanced mathematics to create perfect characters) will object to. Most serious *Diablo II* players insist on maximum attack speed in their melee builds, no matter how hard it is to obtain it. I agree that, if you were trying to build the perfect killing machine, you should have the maximum possible attack speed all the time. To accomplish this, most serious players will also perform thousands of repetitious runs, lasting literally hundreds of hours. The game was not designed to force all players into this behavior. Attack speed, as implemented by the *Diablo II* designers, has a diminishing returns curve built in.[22]



These diminishing returns are in place to force the average player (who is not willing to do hundreds of hours of Lower Kurast runs, for example) to build around what he or she lacks. Because of the diminishing returns on attack speed, players who already have fairly high attack speed aren't going to get as much return

on more items or skills which add to it. The reverse is also true. If a Barbarian has high damage, but little attack speed, his player will have an easier time acquiring a little bit of attack speed through his skills than acquiring marginally higher damage on his weapon. The Whirlwind skill breaks this dynamic a little bit, in that its gains from attack speed are greater than those of other skills. Whirlwind has its own hit-check table that is a little more favorable to attack speed, although it, too, suffers from diminishing returns. The last consideration for Whirlwind is that it costs the same amount of mana no matter how many hits it outputs, so higher attack speed gives the player more hits per mana, even at short distances—explaining its position on the diagram.

To summarize the essence of the Barbarian, I want to point to a set of his most important passive skills: Masteries. Skills like Sword Mastery and Polearm Mastery show how building a Barbarian is a much more reactive process than building other classes. There isn't just one such mastery skill; each weapon type has its own skill. Although character construction is all about tactical choices, the split in weapon mastery skills seems unnecessarily punitive. If the player commits to one type of weapon, do they have to throw away all the other weapon types they get on their journey through the game? This way of thinking gets the Barbarian build exactly backwards. The point of weapon masteries is not to proscribe weapon types, but to adapt to what the player encounters. By saving some skill points, the player can wait for a great weapon to drop and then invest in the appropriate mastery. This is what the Barbarian is like, generally. The Barbarian can choose what passives skills, movement skills, and attacking skills best complement the gear he actually has. For most other classes, players choose a build and then seek out the best items for it. Master-level players do the same for Barbarian, but for most players, the Barbarian allows for a little more flexibility and reactive play.

Necromancer

Builds: 2: Bone/Poison and Summoner

Bone/Poison Necro

Build Strength: Ranged character with terrific crowd control abilities and great itemized skill bonuses.

Build Weakness: Direct damage build has to work a little harder for damage than his fellow casters.

Elemental Availability: Good, can reliably inflict poison and non-elemental magic. Skilled players can inflict fire damage, although it requires certain battlefield conditions. Can mitigate enemy resistances, in any case.

Synergies: No huge, linear synergies in the manner of the Sorceress, Trapsassin, or Hammerdin, but all curses are synergistic with other spells, by design.

Overview: A more technically advanced caster, the bone/poison Necromancer is nevertheless a powerful class when played by an experienced player. Instead of dealing tons of damage directly, the Necromancer uses curses

1. The Different Kinds of RPG, and How *Diablo II* Borrows from Them

to weaken his opponents before hitting them with sources of damage. Minor skills in the Poison & Bone tree also work as great crowd control effects with minimal investment. The Necromancer's peculiar itemization also works to enhance his power in orthogonal ways.

The most important thing to know about the Necromancer's spells is that they simply lack the top-end damage of the Sorceress, Hammerdin, Trapsassin, or Hurricane Druid. Factor in the Lower Resist curse, and suddenly the build makes a lot more sense for poison damage builds. (Lower Resist does not affect pure magic resistance for some reason, although the bone spells are still useful as supplemental damage.) The Necromancer also has other advantages over his fellow casters, mostly in the form of utility skills. Regardless of character class, the average player is going to build their character around three primary skills, investing 20 points into each, plus about 5–10 points into all the prerequisites that unlock those skills. Most players aren't going to use those prerequisites in high-level play, because they're too weak. The Necromancer is one class where that weakness is not a problem. Curses like Decrepify are still reasonably useful at low levels, and Bone Wall can easily be spammed to clutter the map and prevent enemies from advancing too quickly. The poison Necromancer will still probably focus on three spells (like Poison Nova, Bone Spirit, and Lower Resist), but he's got other options as well.

Necromancer itemization is a little unusual as well. The Necromancer's class-specific item is normal in that it has class-specific +skill affixes. It's unusual in that it occupies the shield slot, meaning that the Necromancer can also equip a wand which has even more +skill bonuses. In the earlier parts of the game, this makes it easy for casual Necromancers to build up skills above their character's level. At the highest levels, the powerful unique items available to all casters neutralize this advantage. Another quirk of Necromancer itemization is the unusual prevalence of faster cast rate (FCR) affixes on Necromancer items. The FCR affix is equally useful for all casters, but Necromancers have a much easier time accumulating a significant amount on relatively common gear. Thus, the relatively low damage of a spell like Bone Spirit can be augmented by a much faster cast rate.

Summoner

Build Strength: Offloads the responsibility for fighting (and tanking) enemies on to the Necromancer's minions, allowing the Necromancer to throw curses and spells from the back.

Build Weakness: Depends on the availability of enemy corpses. Consumes more skill points than the poison/bone build, reducing the Necromancer's utility. Summons sometimes have trouble standing up to boss monsters.

Elemental Availability: Poor. The Summoner relies on his minions' physical damage, and in some cases, moderate fire damage from a golem. The Necromancer can mitigate this with curses, however.

Synergies: The Summoning tree has numerous passive bonuses which grant increased health, power, and even resistance to summoned monsters, but these bonuses are more complicated than similar bonuses in other classes and builds. Curses operate synergistically with any build.

Overview: The Summoner build requires lots of management, but greatly reduces the danger to the Necromancer by putting an army of minions between him and the enemy. At higher levels, these minions can easily handle enemy swarms, but they have trouble with bosses unless the player manages the Necromancer minions very carefully. The biggest problem is that the player is not limited by mana, but rather by the supply of enemy corpses.

The Summoner Necromancer is able to raise an army of skeletons or resurrected enemies, as well as one golem. The most important design dynamic for the Summoner is that the strength of skeletons and golems is dependent only on the Necromancer's skill levels, whereas the strength of enemies raised by the Revive skill depends also on the strength of the enemy raised. The two sets of skills share some passive abilities (Skeleton Mastery and Summon Resist), but otherwise present a problem for players. How should Necromancer skill points be invested? A Sorceress's early investments into low-level skills are usually synergistic with her top-level skills. Investments into Raise Skeleton do not help the late-game Necromancer in the same way, but the Summoner needs those points in Raise Skeleton in order to survive the early game. Patch 1.13 allows players to reallocate all their talent points later in the game, but for more than a decade, this wasn't possible. This didn't make the Necromancer unplayable; he is and has been a fun and powerful character. This lack of early-skill-to-late-skill synergy is unusual among *Diablo II* characters, however, and it's curious that the designers never addressed it in any of the numerous patches that came before 1.13.

Summoner mechanics are also unique, although there is much of difference between the use of skeletons, golems, and revived monsters. The easy part of the Summoner build is that the player can simply sit back and cast a few curses while his minions do most of the work. The hard part is keeping all those minions alive, especially against bosses. Skeletons, golems, and revived monsters are always dying, and the Necromancer has to replace them by reanimating another corpse. Most of the time, this is not a problem, since *Diablo II* sends the player through thousands of monsters. Bosses, however, present a problem. Unique enemies, with their enhanced power, can easily kill all of the player's minions without yielding any new corpses to raise. This sends the player running around the map, trying to raise a new army while a unique monster chases them. The effect is even worse with the act bosses, who (with the exception of *Diablo*) all wait in a relatively small location with few enemies to raise. Expert players eventually figure out that certain revived monsters are better for bosses than others, but no other class has to go and track down a certain pack of enemies every time they want to farm the act bosses.

The problem of facing bosses as a Summoner or "minion master" (as the class is more generally called) is one that is repeated in many different games. Both the original *Guild Wars* and *Hellgate: London* (another Brevik game) put their minion masters through similar difficulties when taking on bosses. Minions in those games are more than adequate for common trash mobs, but are often inadequate for dealing or surviving damage from bosses. Although he didn't address this topic specifically, David Brevik commented to me that he thought there were certain unsolvable problems in RPG design—like perfectly balancing AoE spell damage.[23] If many different game companies cannot balance the minion master even with the advantage of lots of hindsight, then we shouldn't fault the designers of *Diablo II* for a relatively small imperfection. After all, the Summoner build is still viable, even if players face an uneven challenge when playing it.

Paladin

Builds: 3: Fanatidin, Smiter, Hammerdin (the former two are both melee)

Fanatidin/Smiter

Build Strengths: High defense and damage output, can get attack speed from a skill instead of an item, can wear a shield and block at a high rate.

Build Weaknesses: Somewhat limited elemental availability in his highest-damage skills.

Elemental availability: Theoretically high, as he has abundant physical and magical damage, and can add several other elements to his physical attack via Vengeance. That said, Paladins rarely choose this last option as it is mutually exclusive with better attacks. So the effective elemental availability is much lower than it seems.

Synergies: Unremarkable, medium-strength synergies.

Overview: The Smiter and Fanatidin are different in one key dynamic: the former is better against single targets and the latter against multiple. They are both melee fighters, however, and primarily deal physical damage.

The Paladin class is naturally durable, thanks to good bonuses from vitality and the highest natural block rate in the game. Indeed, even a low-level use of Holy Shield can max out the block rate of an otherwise mediocre shield. These two melee classes are viable for just that reason. Many newer players prefer the Fanatidin because of Fanaticism's numerous, large bonuses. Indeed, Fanaticism is one of the few skills in the game that gives significant attack speed bonuses without compromising defense (as Frenzy incidentally does), or locking out other skills (as shapeshifting skills do). Its bonuses to damage and attack rating also scale well and can make even simple builds with low-level gear into effective killing machines. More experienced players often prefer the Smiter build for its high top-end damage against single targets. These players tend to run (or teleport) right for bosses. Both builds rely on a similar setup and similar gear, however, and are mechanically very easy. By and large, the player clicks on

a target until it dies. All of the effort in building one of these Paladins is in preparation and gear.

Hammerdin

Build Strengths: Huge ranged damage, can still wear great armor and have a high block rate.
Build Weaknesses: Very limited elemental availability.
Elemental Availability: Very poor, only does magic damage.
Synergies: Benefits from one of the most infamous skill synergies in the game
Overview: The Paladin is also a competent caster class and can use magical combat skills to destroy his opponents at range. One would expect, based on the setup of his physical skills, that the Paladin's magical options would be divided between single-target and multi-target spells, but it isn't so.

Both the Blessed Hammer and Fist of the Heavens spells are meant to hit multiple targets, and they both do a good job of it. The problem is that Blessed Hammer enjoys one of the only multiplicative synergies in the game. As we saw in the amazon section, the Paladin's bonuses to hit rating and damage are comparable to any class's best passive skills. Blessed Aim, Might, and Fanaticism, are a great help to a melee Paladin and his melee friends. None of them can match what concentration does for Blessed Hammer, though. Blessed Hammer's normal synergy is from Sacrifice/Blessed Aim. It gains 12% more damage per level of either; 20 levels in either will multiply the power of Blessed Hammer by 2.4. Normally, the synergy from Concentration would simply add to the multiplier. A 240% bonus from Concentration would normally be added to the 240% bonus from Sacrifice to provide a 480% bonus. Instead, Concentration does this:

$$(\text{Blessed hammer} * 2.4) * 2.4\ [=5.76]$$

In later levels when the player has lots of +skill affixes on their gear, the returns on this equation erupt in tremendous fashion. Almost all of the power of the Hammerdin comes down to this one equation and its deviation from the normal mechanics of *Diablo II*. (It also explains why the Hammerdin is ludicrously popular to people who know about the math.)

Druid

Builds: 3: Elemental, Shapeshifter, Summoner (though this last build isn't really used)

Elemental Druid

Build Strengths: Powerful ranged caster, wide variety of elemental options in one talent tree.
Build Weaknesses: Without significant use of synergies, spells do less damage than other casters.

1. The Different Kinds of RPG, and How *Diablo II* Borrows from Them

Elemental Availability: High, can use spells that deal cold, fire, and physical damage.

Synergies: There are abundant and powerful linear synergies, but these only bring the spells up to the level of other casters. In essence, the Druid has to spend more skill points than the Sorceress to achieve the same damage.

Overview: The Druid is a great example of traditional balance ideas implemented in the context of *Diablo II*. Although the elemental Druid can mix and match different elemental skills fairly easily, his abilities don't naturally put out as much damage as less flexible classes like the Sorceress or Hammerdin.

The Elemental Druid is one of the few builds that has three or more types of elemental damage available in one skill tree, and the only one that has that damage available as entirely spells rather than as attacks.

Accordingly, the Druid can mix and match elements quite easily, thus avoiding the trap of elemental resistance on the hardest difficulty. The Druid sacrifices damage to do this; most of his skills are only half to three-quarters as strong as their analogues in the Sorceress skill trees. Moreover, the Druid has no powerful passive or aura as the Sorceress and Paladin do. What the Elemental Druid does have is a set of powerful linear synergies. Particularly in the fire elemental subtree, the player can raise the damage of spells like Armageddon to levels roughly

comparable with other casting classes. By spending that many points in synergies, though, the Druid loses utility. This is exactly the kind of balance dynamic one would expect from an orthodox RPG. *Diablo II* isn't that kind of RPG, however. The point of the game is to become an unstoppable killing machine, so this balance dynamic seems a little out of place. It doesn't ruin the game; other classes can choose versatility over concentrated power as well. The Druid just has a much easier and more obvious choice.

The place where the Elemental Druid really gets hurt by the extra skill points needed for top-end damage is in his ability to dabble in summoning skills. There is no truly viable Summoner build for Druids, but the summoned minions can be useful for soaking up damage or providing utility skills. A Druid trying to put 80 skill points into elemental spells and synergies doesn't really have room to take advantage of that. Although extraneous or unusable character builds are a fixture of otherwise great titles in RPG history, *Diablo II* doesn't really contain any totally useless skill trees. The summoning tree isn't useless, but it's not always easy for Elemental Druids to take advantage of it.

Shapeshifter (Bear/Wolf) Druid

Build Strengths: Although this is a melee build, it has access to several significant sources of elemental damage as well. This build also has relatively fine control over its balance of tankiness or deadliness.

Build Weaknesses: Although this build has a variety of passive abilities and attacks, Druids who change into the bear or wolf form can only use one spell.

Elemental Availability: High for a melee class; they can output significant physical and fire damage and have a passable poison option.

Synergies: Synergies provide elemental options, but require large investments for viable effects.

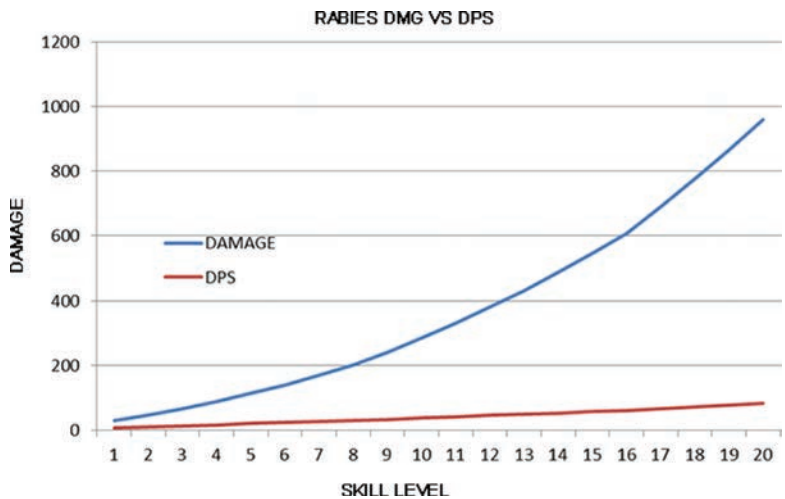
Overview: The shapeshifter build transforms the Druid into a strong melee fighter at the cost of his utility. Wolf and bear Druids are every bit the fighting equals of other melee classes but are a little more confined in terms of action abilities than their peers.

The Shapeshifter Druid transforms into a wolf or bear, gaining large bonuses to attack rating, moderate damage increases, and a variety of specific skills. One of the curious things about the shapeshifting skills tree is that although many of its skills are actually active skills that require continual refreshing, they operate like passive skills. Werebear and Werewolf, for example, only provide passive bonuses to stats, but they cannot be toggled. Rabies and Feral Rage both operate like passive skills as well, but must be cast. Fire Claws, Shock Wave, and Fury operate more like the combat skills of other classes, but all of them are enhanced versions of the basic attack. Sometimes the Shapeshifter Druid can feel a little monotonous for this reason, until he reaches higher levels and can diversify his skills a little bit. Nevertheless, Shapeshifter Druids can be powerful. Fury, for example, is a more

1. The Different Kinds of RPG, and How *Diablo II* Borrows from Them

powerful version of the Paladin's Zeal skill, with the various passive and pseudo-passive skills standing in for fanaticism. One problem the Shapeshifter Druid has is that he cannot cast most of the spells in the Elemental Skill tree. The only spell he can cast is Armageddon, which is a decent source of fire damage, but it requires lots of investment in synergies to become really powerful, and eats up mana that this build doesn't always have. Most of his best attacks are physical, naturally, but he has another source of fire damage and a somewhat useful poison damage skill. Fire Claws is naturally weak (topping out around 400 damage per strike at level 20) and seems, at first glance, like it would only be useful during the mid-game. Fire Claws also has one of the largest synergies in the game. It's not a linear synergy; all of the contributing skills are actually in the Elemental Skills tree. There are five synergetic skills, and all of them offer a 22% bonus to Fire Claws per level, giving the Shapeshifter Druid the power to deal thousands of fire damage if he commits to the synergy. Unlike Armageddon, this skill doesn't consume much mana, and relies on attack speed rather than casting speed. It's a big commitment in terms of skill points, but is a viable means of delivering elemental damage.

The Druid also has a similar ability to deal significant poison damage by means of a large synergy. The Rabies skill has a neat perk: the poison spreads from enemy to enemy. Thus, in large crowds, the player can infect quite a few targets with a decent amount of poison damage. The problem is that it's poison damage, which is generally difficult to use in *Diablo II*. *Diablo II* is so fast-paced that poison damage, which takes place over several seconds, doesn't always kill fast enough. Rabies damage scales nicely, but the time component also scales up.[24]



The player can get around this with mob-tagging behaviors and lots of +skill affix items, but great gear will make any build work. Rabies is a cool skill, but it has a

systemic handicap in that almost all poison skills (except for Poison Nova) are hurt by expanding time denominators.

Sorceress

Builds: 3: Fire, Cold, and Lightning. Technically, there are sub-builds of these three elements, but the mechanical differences and strategy are not different between those builds in the same way that the Hammerdin is different from the Fanatidin.

Class Strengths: Can easily put out tons of ranged damage, uses spells exclusively, has the best mobility skill in the game.

Class Weaknesses: Most fragile character in the game.

Elemental availability: High, can use Fire, Cold or Lightning. Can use any two of those if the player is willing to trade away some top-end damage.

Synergies: Has large, linear synergies for the highest-damage skills in every tree, plus strong, synergistic passive skills.

Overview: A relatively straightforward class with some obvious drawbacks, the Sorceress is easy to build and play through the first two difficulty settings but faces the same late-game problems as every other class.

From a mechanics standpoint, the Sorceress is probably the simplest class in the game. The player chooses a target and fires spells at it until it dies. There's not much else for the Sorceress to do except run (or teleport) away from approaching enemies. The Cold Sorceress is the only version with significant crowd control abilities, and she trades damage (as all cold abilities do) for this effect. The overall strategy of the Cold Sorceress doesn't differ that much from the Fire or Lightning Sorceresses; the player still keeps enemies at a distance. Likewise, Cold Sorceresses can use spells to increase their armor, and Lightning Sorceresses can increase their effective health (through Energy Shield), but the player should still be fighting at range. There are other small differences between the types of Sorceresses. Lightning and Fire Sorceresses benefit a little more from faster cast rate affixes than Cold Sorceresses do, but all the classes benefit most from +skill affixes and mana. Moreover, no affix or skill radically changes the way that the Sorceress class is played.

Although I'll address it again later, I want to mention Teleport, the Sorceress's mobility skill. The interesting thing about teleport, from the perspective of character classes, is that it embodies the Sorceress's design philosophy so well. Teleport is a terrific mobility skill, but unlike the similar skills of other classes, it doesn't have a landing effect. Thus, Teleport is much better at getting the Sorceress out of battle than into it. This is exactly what we would expect from a class that is always supposed to be firing from range and avoiding contact at all costs.

The Action RPG and *Diablo II's* Action Heritage

Diablo, the predecessor to *Diablo II*, began its life as a fairly orthodox roguelike. While the project was still early in development, the main Blizzard team that

1. The Different Kinds of RPG, and How *Diablo II* Borrows from Them

oversaw the project told David Brevik that the game engine should work in real time. This decision changed the course of development in a big way, and introduced many important game design ideas that are not native to the roguelike subgenre. Because it started as a roguelike, *Diablo's* relationship to its roguelike source material is fairly straightforward; it adapts roguelike concepts for a mass-market audience. The relationship between *Diablo II* and its action-game ancestors is more complicated and theoretical. One of *Diablo II's* greatest achievements is the way it recreates the fundamental structure of action games (Nishikado motion) through procedural means. This section will look at the action heritage of *Diablo II* and explain the way it reinterprets that heritage through an RPG lens.

One issue I want to tackle before looking at the history of the action RPG is how to define the level-up. Critics and fans have argued about the point at which an action game has enough RPG content that it becomes a true action RPG, instead of just being an action or action-adventure game. (David Brevik himself pondered this question in another interview[25] without coming to a conclusive answer.) For example, many question whether *The Legend of Zelda* games are really RPGs. I say that they are, because Link can gain more health, magic, strength, etc. over time. These gains are usually not connected to experience points, nor are they part of explicit "character levels," but does that mean the game is not an RPG? For me, *Zelda* is still an RPG, because these things are still level-ups. A level-up is a permanent, periodic increase in the power of a player-character. When Link gains more heart containers, his maximum life is permanently increased, and that permanency makes it a level-up. When he gains the use of the Master Sword, it represents a significant increase in power over the starting sword, and he has it for the rest of the game. These gains are all connected to finding loot, but as we'll see many more times in this book, loot is often the most important kind of level-up. Link makes one or more of these gains in every major dungeon. The period is simply measured in dungeons rather than being measured in EXP. Whether or not you agree, this definition is enormously important to the rest of this book.

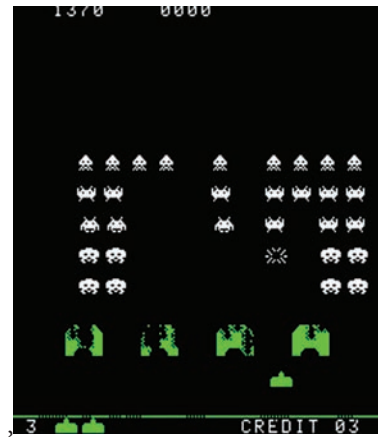
The Birth of the Action RPG

Once RPGs started to appear on PC and home videogame consoles, it was inevitable that design ideas would leak between the booming world of digital action games and the older world of tabletop RPGs. For designers of RPGs, this was a boon, for it gave them another way to escape from the overshadowing influence of *D&D*. For all of its systemic completeness, *D&D* never implemented the kind of real-time dexterity challenges found in action games like *Space Invaders*, *Pac-Man*, or *Galaga*. (It's difficult to imagine exactly how it could do so, to be fair.) By implementing action-game challenges against a background of RPG systems, designers could combine their favorite parts of RPGs with any of the various action subgenres. The result would be a hybrid game that felt significantly different from either of its parents. I call this the *combination* strategy, which allows designers to disguise relatively old RPG tropes in the clothes of an action game.

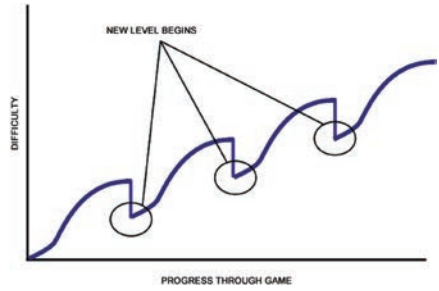
The action RPG had a major and immediate historical impact on the design—and audience—of RPGs. For one thing, the action RPG was the genre that introduced the RPG form to console gamers who had never seen it on the tabletop. Games like *The Legend of Zelda*, *Gauntlet*, *Golden Axe*, *Secret of Mana*, and *Ys* taught console game players how to play RPGs by allowing them to rely on their existing action-game skills to compensate for their lack of RPG knowledge.

Two Difficulty Curves

The first game that is still relevant to the history of action game design is *Space Invaders* (1978). The designer of that game, Tomohiro Nishikado, was also the engineer responsible for the construction of its arcade machinery. Because of the technical limitations of the processor he used, Nishikado discovered that the aliens moved faster when there were fewer of them on screen.



This causes each level in the game to become more difficult as the player progresses through it. Rather than correct for this unusual property, Nishikado kept it as a design feature. Indeed, he embellished it by making each successive level start at a slightly higher level of difficulty, thus reiterating his serendipitous difficulty structure at the macro level.



1. The Different Kinds of RPG, and How *Diablo II* Borrows from Them

I call this up-and-down motion in videogame difficulty *Nishikado motion*. Nishikado motion is the fundamental structure that underpins nearly all of mainstream videogame design. Even today in the era of extensive player psychology research and interest-curve diagrams, Nishikado motion remains the central pillar of mainstream videogame design.

Difficulty in traditional RPGs is structured very differently than it is in mainstream videogames. Games of all kinds teach the skills that players need to learn in order to overcome later challenges. In action games, players can often fail to learn, might only partially learn, or might even forget the dexterity skills that a game teaches. Players of action games can also become literally exhausted by the demands of a fast-paced action game, and their hand-eye coordination can suffer after levels as short as a few minutes. Nishikado motion accommodates this by routinely delivering lulls in the intense action, to let the player regain their stamina and catch up on skills they lack, or that they have forgotten. Traditional RPGs are different. In an RPG, the player's character does most of the learning. It's the character who learns how to pick locks, cast fireballs, parry sword-strikes, and speak with animals. The player just has to declare the action and roll dice. Thus, RPGs can afford a more linear difficulty curve. What's more, because most things in a traditional RPG depend on statistics, it's actually a lot easier for RPG designers to precisely tune their difficulty curves. This is not to say that all RPGs are perfectly linear in terms of difficulty. Even the best RPGs are inconsistent from quest to quest if the player hasn't made certain choices, but the overall emphasis on undulating arcs of difficulty is less prevalent (and less necessary) in tabletop-style RPGs than it is in console videogames.

Although traditional RPGs are probably a little easier to balance than action games of equal scope, certain RPGs are harder to balance than others. *Diablo II*, like most games that have some roguelike DNA, has an occasional balance issue stemming from its widespread use of randomness and procedural generation. *Diablo II's* volatility is a little different from its roguelike counterparts in that there's a much greater emphasis on positive variation. That is, in *Rogue*, *Angband* and *Moria*, lucky players will barely finish the hardest challenges. In *Diablo II*, lucky players can quickly become unstoppable demigods. This is an intentional design decision, one that shaped that everything in *Diablo II*, from the construction of monsters to the way that loot tables work. One of the most interesting things we'll see about *Diablo II's* difficulty, however, is that it is very similar to the up-and-down rhythm of Nishikado motion. It's not exactly the same, but it's clearly analogous. I call this type of difficulty curve *Schaefer variation*, after the Schafer brothers, one of whom (Max) first explained the phenomenon to me, and who continue to use the structure in the *Torchlight* series. (I want to note here that Brevik and designer Stieg Hedlund also had a considerable role in shaping this design structure; it was clearly a group effort. Moreover, Brevik has toyed with the dynamic in his later games as well.)[26,27] Schaefer variation is simply Nishikado motion accomplished through procedural means. In Chapter 3 of this

book, we'll take a look at the exact details of how this happens, but the idea of Schaefer variation is an important one throughout this book.

The Action Parts of *Diablo II*

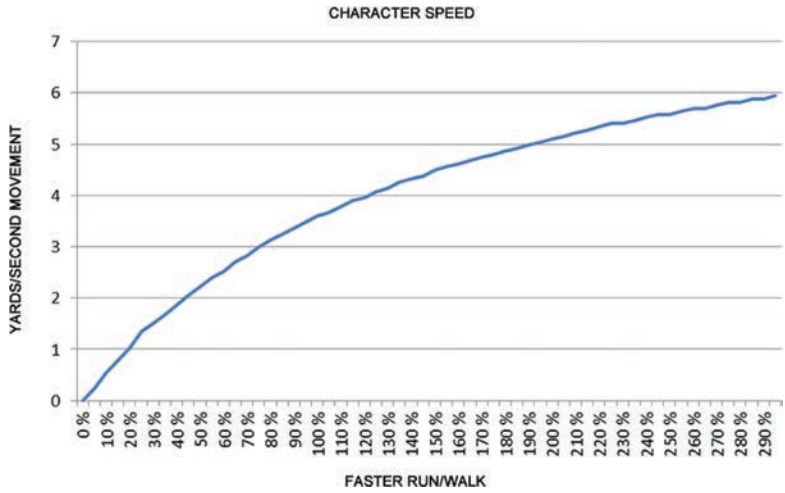
The influence of the action genre runs much deeper in *Diablo II* than merely theoretical ideas. *Diablo II* is definitely an action game, and its action mechanics shape the player's experience in ways that separate it from other games in the same genre. No two action RPGs are the same in the way that they balance and mix their action and RPG components. For example, the most famous action RPG series of all time, *The Legend of Zelda*, depends heavily upon the non-combat applications of its action mechanics. Whether it's hitting switches with a boomerang, leaping chasms with a hookshot, or lighting a series of torches before the first one burns out, there are many tasks that depend on player dexterity and sense of timing but do not involve combat. During those moments, the game almost moves entirely into the action genre. Thus, we could say that *Zelda* games depend a lot more on their action components than *Diablo II* does. *Diablo's* action mechanics are almost entirely built around combat. In the moments when they're not fighting, by contrast, players of *Diablo II* are doing things like crafting or managing gear, which is a part of the game that depends almost entirely on RPG systems. The only action mechanic that the player uses outside of combat is enhanced movement for exploring dungeons, like Teleport, Leap, or Burst of Speed. Then again, all exploration in *Diablo II* is frequently interrupted by combat. When we talk about the role of the action genre in *Diablo II*, we're talking about combat.

Diablo was originally conceived as a traditional, turn-based roguelike. When the game was altered to operate in real time, many of its fundamental RPG mechanics had to change as well. Traditional RPGs take place in discrete turns. Each player-character or enemy gets a chance to act in the order that their simulated skills and chance allow. During their turns, each character can move once and attack once, with some small variations. What happens to this time-tested procedure when real time is applied, and everyone is taking their "turn" at once? All those mechanics either need to be adapted or dropped. For the most part, *Diablo II* adapts them to real time in straightforward, intuitive ways that place most of the emphasis on simulated skills, but there are some action-specific adaptations as well. The following section will analyze each of the ways that *Diablo II* adapts RPG ideas into an action context, and the implications for the game as a whole. Some of this material was briefly covered in the character class section, but here we'll spend more time on the exact mechanical operation of action abilities with an emphasis on quantitative analysis and diagramming.

Movement

Most movement in *Diablo II* is uncomplicated. All characters move at a speed measured in yards per second. Characters run 50% faster than they walk, but while running, they have no effective defense rating. There are lots of items that grant increased movement speed, especially unique boots and magical charms.

Many expert players seek to max out their movement speed through these items as an essential part of farming/leveling runs. Like many of the things that these players seek to max out, these bonuses suffer significant diminishing returns as the character nears maximum speed.



The gains in speed flatten out quickly. Diminishing returns on real-time effects is a theme we'll be revisiting often in this section. While many ultra-hardcore players still insist on having maximum speed, these diminishing returns are in place to encourage players to seek a more balanced approach to their gear. That said, speed bonuses granted by abilities (like Vigor or Burst of Speed) do not experience diminishing returns when combined with other speed bonuses—although the skills do grant lower bonuses per point invested in them. I will discuss movement abilities in their own section, below.

Monster Movement

Monsters in *Diablo II* mostly follow the same rules as players. They have set run and walk speeds, although these can be increased by affixes and auras. Some monsters can leap and teleport, much like player characters can, although they appear to wait much longer between uses of these abilities than players do—probably because of a special timer built into their movement behavior, or else every battle would be nearly as difficult as player-vs-player combat. There are several monster affixes that enhance their speed as well. Champions and Fanatic enemies move faster than normal, and any unique enemy with the "extra fast" affix will move at double speed. Because these speed bonuses are based on abilities rather than gear (which enemies don't wear), the enemy speed bonus does not experience diminishing returns, and so some of the fastest enemies in the game

can become terrifyingly fast if they happen to fall within a movement speed buff of a procedurally spawned enemy. This is one of the ways in which we'll see (in Chapter 3) how the difficulty of the game can suddenly spike thanks to unfortunate procedural generation.

There are two modes of enemy movement that the player doesn't have: flying and phasing. The Scavenger class of enemies can fly over the battlefield, immune to all attacks, before landing to strike (below, left).

This is the only family of enemies which moves like this, and it gives the enemy minimal advantage, as they cannot attack while in flight. Similarly, *Black Souls* (above, right) can phase out of reality, becoming hard to see (and therefore hit) while moving. The Black Soul can be exceedingly dangerous because of how quickly it can phase in, take a shot at the player, and then phase out again. Although many movement abilities (like the Cave Leaper's jump) are annoying to deal with, this is the only one that makes an enemy significantly more dangerous without a procedurally added affix.

Player Characters and Their Movement Options

With one (glaring) exception, melee classes have the best special movement options. Like basic movement, these abilities are not especially complicated. The rules are as follows:

1. The player-character must have line of sight to the target location
2. The ending location of the ability must be navigable (not a wall or a pit)
3. There must be no impassable obstructions between the character and the ending destination

These rules are fairly straightforward. The first two apply to all movement skills equally; the last rule varies in its application from skill to skill. Ground-based

movement abilities are subject to enemy attacks and can be stopped as such; a leaping Barbarian does not have the same problem. Ease of movement isn't the only factor in the utility of a movement skill, however. Other things like movement speed, movement distance, mana and time costs, and impact on enemies all affect the usefulness of an ability. The Paladin and Barbarian movement skills have reasonably strong attacks built into them but aren't terribly fast or far-reaching. The Assassin's Burst of Speed ability is easy to sustain indefinitely and includes an attack buff but is easily stopped. The Sorceress's Teleport is twice as expensive as other movement skills, and has no effect on enemies, and it leaves the caster somewhat vulnerable. Yet, Teleport can move the player across the landscape faster than any other skill in the game, and doesn't collide with terrain objects except map edges. Many of the game's most popular items (like the coveted Enigma runeword) offer the Teleport skill to non-Sorceresses as their primary benefit. The skill is simply too desirable to pass up.

There are three great object lessons about game design that we can learn from the way that movement abilities in *Diablo II* are configured. The first lesson is that, if you're designing an action RPG, you can bet that players will find ways to employ action solutions to your RPG problems. One theme common to action games of all genres (whether platformers, survival horror or anything in the open-world format) is that players eventually learn to avoid most enemies. Many enemies offer either no reward, or only a trivial one. In a game with powerful running and jumping abilities, it is easy (and sensible) for the player to avoid all of these encounters. (There is an entire genre of YouTube videos where players do this for long stretches in *Dark/Demon's Souls* games.) Sure enough, that's what experienced players in *Diablo II* do. Whether it's rushing to Baal or Mephisto, or running around Lower Kurast looking for the infamous super-chests located there, players use their character's movement skills to exploit all manner of shortcut and AI patterns in order not to fight enemies. This is yet another area in which the Teleport skill is overpowered; it deals no damage and has no stun or knockback effects. It doesn't need them, though, if the player is simply avoiding combat.

That last point makes for a good segue into the second lesson, which is that qualitative differences are not the same as tactical balances. Both of those things are necessary parts of a good class system. In the previous three *Reverse Designs*, I made the case that qualitative differences in videogame design are greatly underappreciated. I maintain that stance here, with one important caveat: qualitative differences only matter if the quantitative differences don't make them obsolete. If there are two paths through a game, and one of them is much harder than the other without any commensurate reward, it doesn't matter if they're qualitatively different. Players are going to choose based on difficulty. The same thing applies to RPG abilities. It doesn't matter how great the Barbarian and Paladin movement skills feel—though they do make you feel like a badass when you use them right—if they lag way behind Teleport in speed and range. It's tempting to counter this with "but Sorceresses are fragile!" That's correct, but

remember that we're talking about mobility and avoiding enemies here. Fragility hardly matters if the player is deftly zooming past anything dangerous. In order for players to be able to choose freely among qualitatively different options, those options must be roughly equal in effectiveness.

The third lesson we can learn from movement skills in *Diablo II* is that there are always lots of creative solutions to game design problems, but those solutions need to make sense in the context of the game they apply to. The problem with movement in *Diablo II* is that Teleport is too powerful, and the designers knew it. Teleport is also far too resonant with the Sorceress class—and really, the game in general—for the designers to remove or nerf it too harshly. Their solution was to offer Teleport to any class on a variety of items like amulets, staves, or even armor. This is the quintessential *Diablo II* strategy. The design philosophy of *Diablo II* includes the tenet that any character should be able to become an overpowered demigod. Teleportation is a part of that strategy, and inasmuch as any class can gain Teleport, it makes sense in the context of *Diablo II*. Rather than nerf one class and its abilities, the designers simply make the other classes more powerful by addition.

Crowd Control Skills

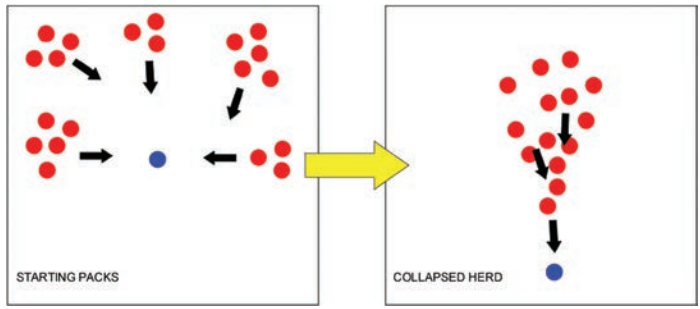
Only a few classes have skills that greatly enhance their movement. All classes, however, have the reverse, skills that make enemies slower or stop them entirely. We've already covered some of this in the section on character classes, but here we're going to pay more attention to the particular action dynamics of the various crowd control abilities in the game. One of the most common problems in *Diablo II*, especially for inexperienced players, is becoming surrounded by enemies.

In large numbers, these enemies can even kill player-characters whom they ordinarily could hardly touch. What can a player do if they lack the escape skills like Teleport or Leap—or can't use them because the crowd of enemies is too big?

1. The Different Kinds of RPG, and How *Diablo II* Borrows from Them

The answer is, of course, crowd control skills, which either restrict or redirect the movement of that enemy mob.

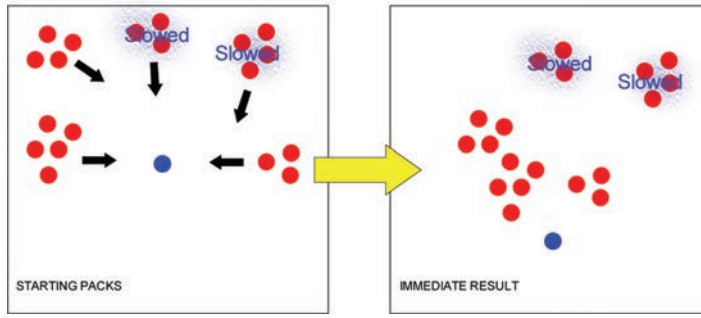
The first thing to understand when it comes to crowd control dynamics in *Diablo II* is how the enemy AI controls those crowds. Brevik and Hedlund both explained that, except for bosses, AI in the game is very rudimentary.[29,30] Most normal enemies can idle, move, or attack. Some enemies also have a simple evasion script, like the Dark Archer, who will flee the player to perform a ranged attack, or the Cave Leaper, who only comes into melee range to make brief hit-and-run strikes. Aside from these occasional, small variations, common enemies operate in the traditional videogame herd. A herd is any group of enemies whose primary movement pattern is to simply collapse on the player. In the diagram below, the red circles are enemies, and the blue one is the player.



No matter what shape a herd begins in, the enemies in that herd will collapse to form an amorphous blob around their target. The player can take advantage of this in a few ways, with or without crowd control abilities. With a few exceptions (Ghosts, flying enemies), enemies cannot move through one another. Because their AI is so rudimentary, large groups of enemies of roughly similar speed will slowly get stretched out in a path moving toward the player. The herd and its three basic shapes (initial placement, amorphous blob, and stretched-out) are an important to the way that crowd control works, as we'll see below.

Categories of Crowd Control Skills

There are three categories of crowd control skills: slows, repels, and distractions. All of these are useful, but they have different effects on the real-time crowds that the game hurls at the player over and over. Slows reduce the movement of enemies in a group. (Note that I include all stuns in the slow category because the crowd-control impact of stuns is essentially the same as those of slow effects.) The most prevalent example of the slow-type debuff is the chill effect from cold damage.

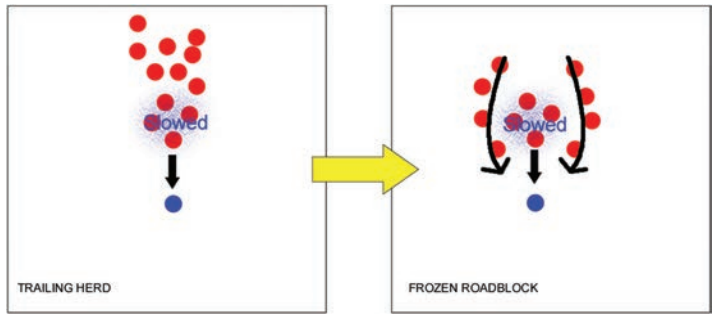


Most cold damage slows an enemy by about half, although this can vary based on the source and the target in question. Because all cold attacks apply this debuff, cold damage is weaker than other elemental damage of the same level. For example, the level-35 affix for cold damage on magical weapons adds an average of 25 cold damage to an attack, while the fire-based suffix of the same level, adds 56 damage.[31] There are also a few cold attacks, like Frost Nova, which trades away 75% of its damage (compared to same-level spells) for three times the freeze duration. This is merely an extension of the same principle that underlies all cold damage. Similarly, some cold attacks can freeze enemies solid. Like all stuns, this is just a more extreme slow, but it also trades away even more damage to achieve the highest level of crowd control.

Slow effects can also be administered through a Necromancer curse, or through a weapon affix, which appears on unique and set items. Obviously, only the Necromancer can use curses, but the slow applied by items is also limited. As with most slows, the classes that benefit from the weapon-based slows are ranged. By slowing a herd of enemies, the player gains separation from them, and can subsequently pummel them from a distance. It's a little strange that of the unique weapons which have a slow effect, 10 are melee weapons and only two are ranged. (Ranged weapons have yet another small drawback.) Nevertheless, curses and weapon-based slows stack additively with cold and with each other, and can greatly reduce enemy movement speed. There is a cap to the amount of slowing that an enemy can receive (an 85% reduction in speed or one yard per second, whichever is higher), although nothing in the game indicates this.

Regardless of the type or degree of slow inflicted, there are specific action-oriented effects of using this type of crowd control. In the context of small groups of enemies, slowing effects are simple and have predictable impacts on the herd.

Slow effects delay the threat from one or more of these converging herds. In the small groups that populate many narrower areas, this works great, but not all crowds of enemies are so small or move in toward the player piecemeal. Some herds number in the dozens and come in an indistinct blob. These groups have different dynamics when it comes to slow effects. With a few exceptions (ghosts, teleporting enemies, flying creatures), monsters cannot move through one another. Thus, enemies under the effects of a slowing debuff become a natural barrier, while giving a new shape to a large collapsing herd.

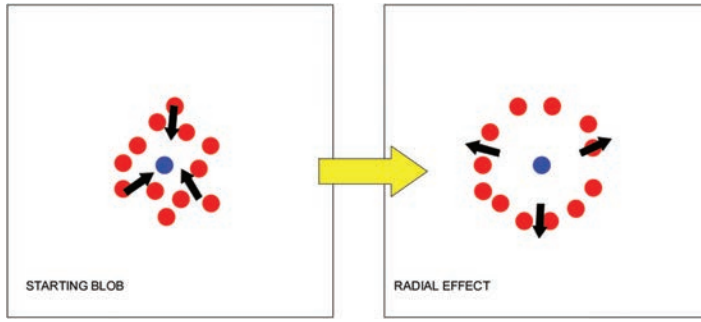


Shooting a cold ability or other slow into a crowd can actually cause secondary problems if the radius of the slowing effect is not large. The enemies will split up to come around their slowed comrades, and in doing so, spread out. For a ranged caster who's kiting a herd (leading a herd while firing at a distance), this creates more diffuse targets, requiring more spells or more kiting. This is not a huge problem, but it's one that could be solved by the right shape of attack. The "Cone of Cold" is an attack dating back into the tabletop years, and cone-shaped attacks

are plentiful in the action game heritage. The lack of such an attack in *Diablo II* is a little strange. *Diablo II* does use a few cone-shaped and linear attacks, but none of them are cold-elemental. It's not a game-breaking deficiency by any means, but it is a curious omission.

Repelling Attacks

The second type of crowd control attack is the repelling type, which does not (usually) slow or stop enemies, but forces them to move away from the source of the attack. There are four characteristics that define a repelling skill: shape, source, distance, and duration. Some repelling skills, mostly those connected to melee combat, are radial in shape.



The Barbarian's Howl skill is the best example of this, but certain weapons and armor (the unique helm Howltusk, for example) also repel enemies within a small radius of the character who equips them. Weapons only cause one monster to flee at a time, but the AoE abilities that use this kind of repelling attack don't allow players to be selective in which enemies they target. All nearby enemies will run away. This seems like it would hurt melee attackers who need to get in close to at least one enemy, but this problem is somewhat mitigated by the fact that radial repelling skills don't have a huge area of effect, nor a long period of effect. Duration and distance are an important part of repelling skills. The small window of time and space afforded by radial skills like Howl give the player just enough time to target the most dangerous enemy of the group with an engage skill like Leap Attack. This is useful for sorting through crowds more than it is for getting out of them entirely.

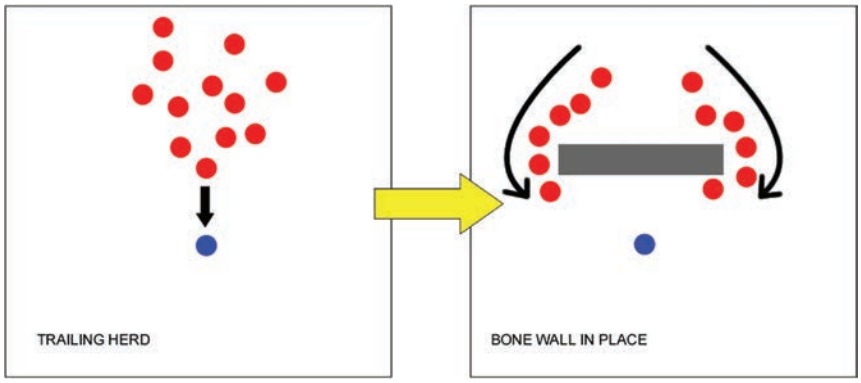
The other major shape for repelling crowd control abilities is the linear shape. This type of crowd control includes most knockback skills, which target either one or a small group of enemies which are knocked away from the attacking character in a straight line. Because of the low number of targets on most of these skills, this type doesn't control "crowds" quite as much as it controls individual members of that crowd. Knockback skills tend to be cheap and high-damage relative to other crowd control skills. Knockback is also usually the property of an attack rather than

1. The Different Kinds of RPG, and How *Diablo II* Borrows from Them

a spell. So, on the one hand, it can miss its target. On the other hand, it is one of the few types of crowd control that can easily be combined with other types. Even low-level gear and abilities can grant a clever player the means to both knock back and chill his or her enemies. Indeed, this is a good way to increase the effective duration and distance of knockback. Those two dimensions are an important part of all crowd control skills, but knockback offers little separation and very little duration. Thus, those combinatoric possibilities are key to the knockback's long-term utility.

Outsourcing the Slow and Repel

Slow and repel attacks have different dynamics if the source of the skill is not the player-character's body. The clearest example of this is Necromancer curses, which can inflict a variety of crowd-controlling debuffs, like slow. This spell is in the typical radial shape, but it's centered on the player's mouse click, rather than on the character's body. This affords the player a lot more discretion in changing the shape of a collapsing herd, and also allows players to use the ability without having to wait until enemies are on top of their characters. Similarly, the Barbarian can use a radial repel attack, Grim Ward, which can be placed on the battlefield. Although its radius isn't very large, the ability to place it away from the Barbarian gives the player an advantageous angle on enemy movements, and the totem can even obstruct passageways in dungeon settings. The Necromancer's Bone Wall is similar in effect, but linear in shape.



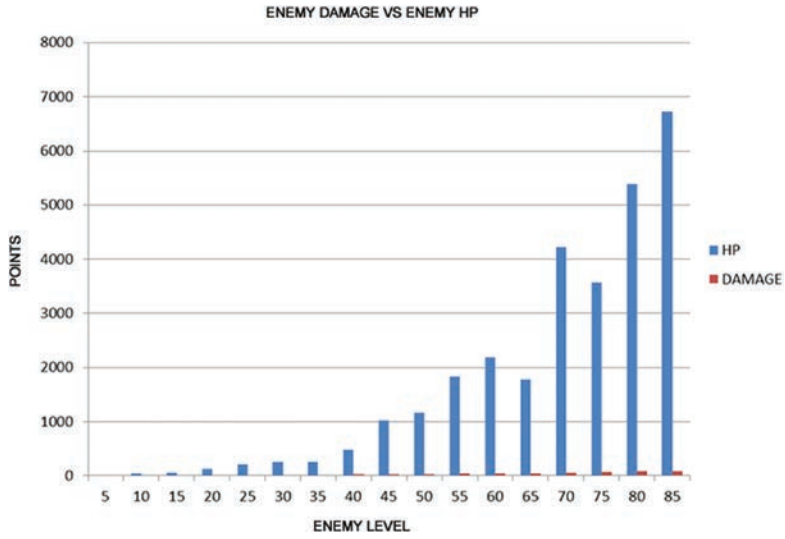
Knockback is perpendicular to the player, while the linear shape of Bone Wall runs, when placed, parallel to the player. Although Bone Wall blocks more than it repels, it nevertheless has largely the same bottleneck-creating effect as Grim Ward when used in the right spot. Putting it in the right spot is fairly easy, though, because the source isn't the player.

Enemy Infighting

Certain abilities can cause enemies to target members of their own group with attacks. The notable examples of this type of crowd control are the Assassin skill

Mind Blast and the Necromancer curse Attract. If you don't consider monster stats, these crowd control abilities are balanced sensibly. Both Attract and Mind Blast have small effect radii compared to other debuffs. Attract does not work on boss mobs; Mind Blast only has a low chance of controlling enemy monsters. When used, they still distract enemies from attacking the player, and they still create the wedge around which unaffected monsters have to walk, although the size of the effect is noticeably smaller. All of this seems like a fair tradeoff if the debuff is going to cause monsters to deal damage to one another, taking some of the burden off the player.

The problem with the balance of enemy infighting abilities is that enemies deal very little damage compared to the amount of HP they have. In fact, you can barely see the HP if you put it on a graph with damage.[32]



This dynamic exists because *Diablo II* puts much more emphasis on killing power than it does on character survivability. The most HP a player character could reasonably accumulate is around six thousand (without having to sacrifice every other stat). A damage-focused character with the same amount of time invested might be able to output four or five times that much damage per second. Thus, enemies have tons of HP on the harder difficulty settings, but deal relatively little damage. Any skill that causes a monster to damage itself or another monster just isn't going to be significant. These skills do a reasonable job of keeping enemies occupied, but Mind Blast doesn't last very long, and Attract is overwritten by any curse which might actually help the Necromancer deal damage to those targets. Other crowd control skills are simply more useful.

1. The Different Kinds of RPG, and How *Diablo II* Borrows from Them

Key Lessons from Crowd Control

Crowd control skills in *Diablo II* teach a clear and widely applicable lesson about action skills in an RPG. Although all the crowd controls are meant to be different—meant for different characters and contexts—certain characteristics that they share can make them more or less powerful. For example, the shape of a crowd control skill doesn't matter very much. As long as a linear crowd control can affect more than one enemy (as Bone Wall does), and the player can aim it well, it can be as effective as a radial crowd control (like Psychic Hammer). On the other hand, the source of a crowd control skill affects its utility immensely. Howl is a useful skill that sends enemies running away from the Barbarian, but Grim Ward and Terror can do the same thing at essentially any point on the screen, offering the player the same effect with more versatility. Finally, longer durations are a huge asset to crowd control, but are also the first target of balancing efforts. The most common crowd control effect, chill from cold spells, is also the one to which all enemies gain resistance on higher difficulties. The chill effect in hell difficulty lasts for only a fraction of the length it does on normal. Less common sources of crowd control effects don't suffer from diminished effect on higher difficulties, which is a pretty typical balance dynamic for an RPG. Rarer skills and items tend to be better than common ones.

The UI Problem

For all the nuance and complexity built into the action abilities in *Diablo II*, using those abilities is occasionally difficult because of the antiquated UI. I want to say up front that in its own time, *Diablo II* offered a slick player experience. The part of the UI which matters the most, targeting enemies with skills, works quite well. Right-clicking to target with a spell is natural and has the precision of mouse movement. Moreover, hitboxes in *Diablo II* are on the generous side for most monsters.

That said, sometimes certain terrain doodads also have these larger hitboxes and can block shots you wouldn't expect. Moreover, in large crowds, it's hard to target a single enemy because of the overlapping hitboxes. Some of this is probably intentional; it should be hard to hit enemies in large crowds. In any case, it's not the kind of game where it matters which enemy the player hits first—at least not often.

The real problem with *Diablo II's* UI is how difficult it is to switch between skills. We just saw two whole sections about the many action applications of movement and crowd control abilities. It's not unusual at all for a good player to switch between a movement ability, a primary attack and a crowd control skill, or even two of each, but how should the player switch between them? The big limiting factor is that the player's primary hand cannot leave the mouse. For people who don't have a deluxe gaming mouse, that leaves five fingers on the other hand that the player can use. Four of those fingers are dedicated to potions most of the time. The thumb can handle the item drops or the "hold still" command (with some button remapping). The designers found themselves without a convenient solution for switching between a variety of skills. They went with the obvious (but somewhat inconvenient) solution of using scroll wheels or the function keys.

The lower part of the QWERTY keyboard is standardized, so the designers can design and playtest for it without having to do extra research. Everything above the standard QWERTY layout and letters can vary significantly across different models and eras. The space between the number row and the function keys (which can be assigned to abilities) can vary between different models of keyboards. The alignment of the function keys can vary. The size and texture of the function keys can vary. The average human hand can't comfortably reach more than two function keys without having to move the entire arm a little bit. The muscle movements of the arm (especially the non-dominant arm) are not nearly as precise for most people as those of the hand, and can cost precious milliseconds while the player is in combat. What's more, the player cannot look away from the screen because the game is so fast-paced.

There are quite a few problems with the configuration of *Diablo II's* keyboard controls, but none of them are actually fatal to the game. With some practice, players

1. The Different Kinds of RPG, and How *Diablo II* Borrows from Them

can either reprogram their keyboard or learn to use the first few function keys reasonably well. The player can also use the scroll wheel on the mouse to cycle through abilities, but that isn't as precise as analogous controls in other action games. Because *Diablo II's* systems are so fun to use, players will adapt, but they shouldn't have to. One of the few things that *Diablo III* unequivocally improved over its ancestor was the design of the hotbar. In that game, the player has easy access to six abilities and a potion by only moving their fingers. I'm ambivalent about the change in potion consumption rates (and health systems in general) in *Diablo III*, but some kind of change had to happen in order to facilitate easier ability use.

I realize that it's somewhat unfair to criticize *Diablo II* for not inventing keyboard setups that didn't exist at the time. The design of action keyboard maps and quick UIs were largely driven by the FPS boom of the early 2000s. Millions of dollars of playtesting went into the creation of those games, and each one learned from the errors of the games that came before it. I still point out the problem, however, because it's important to the history of action RPG design. Perhaps the exact changes which *Diablo III* used to allow players to mix abilities weren't perfect, but it should be clear that they were necessary, and why.

Looking Back at Action Mechanics in *Diablo II*

Although *Diablo* began its life as a turn-based roguelike, by the time of *Diablo II* the designers had put a lot of thought into the implications of including action mechanics in their game. Special abilities for crowd control and character movement show that the design team didn't just import traditional role playing ideas into the action space without thoughtful adaptation. Not only do the movement abilities of each class feel different, but they also match the design philosophies of the characters that use them. Whether it's chilling enemies, stunning them, knocking them back, scaring them away, putting up walls, or even causing enemies to fight amongst themselves, there are many ways for the characters to deal with dangerous swarms. None of these abilities are merely token efforts, either. If used correctly, each class's abilities are useful and distinct from what the other classes have. This brings up another important aspect of the influence of the action genre on *Diablo II*. When a game mixes genres well (what I have elsewhere called a composite game), it is possible for the player to use the mechanics of one genre to solve the problems of the other. Good action mechanics in *Diablo II* can absolutely make up for statistical shortfalls, especially in the early game. Good use of kiting and crowd control can allow fledgling casters to over-perform their early damage and mana shortages. Movement skills, meanwhile, can help melee classes target the most dangerous members of enemy packs and eliminate the unique and champion enemies who present a real threat, early in the game. Whereas a good command of the action mechanics is nice to have early, it becomes essential in the late game. Bad action mechanics can cost even well-built characters some deaths if the player strays carelessly into situations no character can reasonably survive. Action mechanics could easily have been a place where the designers simply added real time without any depth, but they took the time and care to make sure that the action part of the *Diablo* formula shines through.

~24~

Xbox—the First Year

Xbox had a pretty good first year. Of course it didn't even come close to being profitable, but nobody expected that it would be. What it did do is validate that Microsoft could create a console system that could compete in the market against the current leaders—Sony and Nintendo. No American manufacturer had been able to do that since the early days of Atari, Coleco and Mattel. 3DO was the last attempt by an American manufacturer to break the Japanese stranglehold on the console business, and it had failed, arguably due to a lack of good software more than by a failure in design or hardware.

And what Microsoft had that 3DO had lacked was software—notably Halo: Combat Evolved, which was a runaway hit. But there was more. Xbox had the Ethernet port that allowed players to link their boxes together, which was a big part of Halo's success, and even before Xbox Live, was an advantage.

Because it proved itself, Xbox gained more third-party developers and its portfolio of games continued to grow. Before the first year ended, it was clear that Xbox was a marketing success. However, it was not an unqualified success. There were supply problems; they couldn't manufacture enough units to meet demand. It failed in Japan and under-performed in Europe. It had to make quick adjustments to the controller. And despite the fact that there were new third-party developers coming on, Stuart Moulder points out that Xbox didn't have a particularly strong Spring lineup, and their second holiday season was uninspiring, with the exception of Splinter Cell, which helped carry it through.

Fries Takes Third Party

After the multiple staggered launches of Xbox in the US, Europe and Japan, Ed Fries was settling in. His son, Xander (short for Alexander), who was born in March, shortly after the Japanese launch, was named after Xbox. Fries says that Bill Gates signed a Japanese Xbox for Xander on which he wrote, "Ed Junior's First Toy."

Soon afterward, Robbie Bach decided to switch the third-party operations to Fries, who was already in charge of first-party. The switch moved George Peckham's group under Fries. ""So not only was I going around meeting with game developers, would work with our 1st-party team, but also meeting with all the big publishers."

Part of Fries' new challenges involved signing up companies to support Xbox Live, which was in development and expected to launch within the next year. One of their toughest negotiations was, not surprisingly, with Electronic Arts. "EA in particular were very slow to support Xbox Live; they had a lot of demands for their support of that platform. But they were the big guys, so..."

Part of what Fries faced by heading both 1st-party and 3rd-party was that he was simultaneously competing with the very companies with which he was negotiating. "It was a little controversial when Robbie decided to move 3rd-party under me. I think we were the only console that put 1st- and 3rd-party together under one person." One area of potential contention with EA was in the sports franchises. EA was the leader in sports games, and didn't much want competition, but as Fries notes, "We did have a sports division. Fortunately, it wasn't good enough to really be too threatening. I think they were confident enough that they didn't see us too much as a threat. But there were other places, say racing for example, where we were very much head-to-head competitors."

The Strategic Plan

O'Rourke claims that the marketing strategy for Xbox was very well executed. "If you look at it through a marketing lens, the core tenets of any really successful marketing driven company, there's brand at its very center and then a real strategic commitment to the messaging around the brand proposition. That's what we did with Xbox from the very beginning. Where I think some other organizations struggle, everyone on the Xbox team had a shared understanding and shared vision of what the brand stood for."

O'Rourke admits that, in addition to good strategy, there was also luck. "It was a combination of both brilliance and luck, but the brand attributes that we strived to create around Xbox mapped incredibly well to the target audience that we were going after. It differentiated us significantly from Nintendo and Sony." Of course, Nintendo was sufficiently different in their branding

and marketing strategy, O'Rourke says, that it was fairly easy to differentiate Xbox against them. "Sony was more challenging," he says. "Sony had that macro brand. Sony stood for entertainment, and the PlayStation had been very successful." For O'Rourke, the big differentiator was connectivity. We talked about what made us the next generation console, in particular being able to tell a story that it is more about connecting gamers, not just to the person next to them on the couch but to a person sitting on a couch across the city or across the state... it gave us a story and a brand position as an innovator that Sony was now having to respond to."

What We'd Never Done Before

As only he can, J Allard looks back at what they had accomplished with Xbox and how much of the effort was brand new to all of them… how he saw the process in some ways as a family affair.

"look, the most complicated msft hw product was the mouse and we outsourced a ton of it. we had never sold product to or through walmart, target or best buy. we had never published a console game. we had never "certified" 3rd party products. we had never had billions in parts inventory. consumer branding? ha. we did product branding for productivity and some bled into the consumer, but never released a major product that was consumer first like this. we never had a million square foot manufacturing facility. the list goes on. we built a whole ton of new competencies. for all intents and purposes, it was a new company that had the full backing of microsoft. how could you say one thing was more important than the other? sure, manufacturing isn't

that glamorous to talk about, but what if we missed christmas or had assembly problems? retail seems easy now, but at the time everything we did was through oem's and distributors. no wal-mart? no xbox. we had games from a lot of sources, but no 1st party hit exclusive to the platform (other than halo) then what would our p&l have looked like, 3rd party confidence? no support from EA, no madden? no gamers. i could go on forever. i appreciated every single part of the family we built and we couldn't have succeeded without every single person on the team. there was no wasted energy - no room for it. who's to say any was more important than the other? we love to tell stories with heroes, but this was a story of a family"

An Xbox team photo.

It took, a lot of brilliant, hardworking people to create Xbox, the project that was once thought to be a pipedream.

2

Planting the Seed

By the late 1970s, having gone from a leather goods supply company to a retail electronics giant in the span of just a few decades, Tandy was already a king of unlikely origins and transformations. As such, it is hardly surprising that Tandy's first foray into color computing had an equally unlikely genesis, this time on the American farm.

Not long after Radio Shack made its splash in the home computer market with the TRS-80 Model I, Tandy officials were invited to take part in an exciting and ambitious project to bring computing technology to tobacco farmers in central and southwestern Kentucky. The year was 1978, and the project, known as the Farm Information Retrieval System (FIRS), was conceived as a way to electronically deliver data directly to where the farm owners lived and worked. Proposed in March of that year by Kentucky Senator Walter "Dee" Huddleston, the $380,000 initiative was sponsored by the U.S. Department of Agriculture (USDA) and the National Weather Service (NWS).

The resulting project was simply called "Green Thumb." The unusual name might have seemed inauspiciously similar to the title of Ian Fleming's James Bond-driven novel and film, *Goldfinger,* which was named for its lead villain's obsession with gold, but the project's moniker actually fit the term's origination perfectly. While today "having a green thumb" means someone who is skilled

at gardening, the term actually came from colonial America, where farmers' thumbs would literally be stained green from cutting the stems of tobacco flowers with their thumbnails.

Green Thumb was envisioned as a farmer's "electronic almanac," where information such as commodity prices and weather data would be made available via a computer terminal at the farmer's location. This interface would provide the key information that the farmer could use to facilitate decision making on everything from crop management and cultivation practices to production costs and fertilizer and pesticide applications.

Operation was envisioned to be simple. The farmer would be presented with a menu of options, then would press a key corresponding to the information that he or she wished to see. The selection would then initiate a transfer of bits of information that traveled via a modulator/demodulator, or modem, and then over the venerable "Ma Bell" telephone system, whose twisted copper wire had covered a good part of the rural American landscape by the mid-1970s. The downloaded information could then be viewed and referenced as needed until another choice had been made or the terminal was powered down.

Expectations were high that Green Thumb would deliver: government officials estimated that the data provided by the project could contribute to a 25% reduction in pesticide use alone, making crop management cheaper and more cost effective. In order to facilitate its usage, the Green Thumb Box, as the terminal was to be called, would be offered free of charge to farmers. For the initial phase, 250 units would be made available in Kentucky: 100 would go to Shelby County, while another 100 would be sent to Todd County. The 50 additional remaining units would be kept for repair and testing purposes.

As Green Thumb began to gain momentum, the NWS sought bids from companies that could do both the design and manufacturing of the necessary hardware. Tandy Corporation was an obvious choice of bidder as a previous collaboration with the NWS led to the successful weather radio, then selling in thousands of Radio Shack stores across the country. Not only that, but Tandy's manufacturing capability in Fort Worth, Texas, was more than capable of handling the small number of terminals that were anticipated for the project.

When made aware of the project, Tandy management expressed interest, but there was a problem: the company was not set up to bid on government contracts. What Tandy needed was a partner that had experience drawing up bids for government-sponsored projects and could then be trusted to properly negotiate the ensuing contract. Tandy found that partner in Motorola. For years, Motorola had been supplying the government's space program with electronic parts and other products, and had razor-sharp experience in cutting through the bureaucratic red tape that so often accompanied the bidding process.

The synergy between the two companies was clear, so Motorola and Tandy drew up a technical and management proposal. The former would provide design and parts for the hardware, and the latter would lay out the printed circuit board (PCB), provide the casing, and then perform final assembly. Motorola's Stan Katz, who managed the systems engineering group and headed the strategic

marketing group that would try to find markets for Motorola semiconductors, oversaw Joe Roy, who would manage the technical side of the project along with Steve Tainsky, William Peterson, and Jim Reid. Tandy's Director of Engineering, Dr. John Patterson, would take charge of the assembly and manufacturing process, with Chris Kline and Jerry Heep covering the engineering responsibilities. Motorola's Don Sheppard would coordinate from above as the project manager. From the government's end, USDA official Howard Lehnert would lead the project, while Dr. John Ragland of the University of Kentucky would provide the necessary academic support.

The joint design by Motorola and Tandy consisted of a single-board computer housed in a retrofitted TRS-80 Model I case, with a final design reminiscent of the era's television videogame consoles. Atop the case there would be no QWERTY keyboard. Instead, a 12-button keypad was located in the middle, resembling a touch-tone telephone with several extra keys on the right side. The brain of the terminal system would be the proven 3870 microprocessor, which was a variant of the Fairchild F8 CPU used in the first cartridge-based home videogame console, the Fairchild Video Entertainment System (VES), which was released in 1976.

Visual display of information received by the Green Thumb Box would be achieved by connecting the device to a standard television set via a built-in radio frequency, or RF, modulator (Figure 2.1). Much like a TV station broadcasting its programming, the RF modulator transmitted video on either channel 3 or 4, with the selection depending upon which received less interference in the user's area. The Motorola-designed 6847 Video Display Generator (VDG) integrated circuit, designed by Motorola's Peterson, was capable of generating 32 columns and 16 rows of text on the television, for a total of 512 viewable characters on a single screen. Each character would be comprised of an 8×12 dot matrix. Additionally, graphics capability would be available with up to eight colors in a

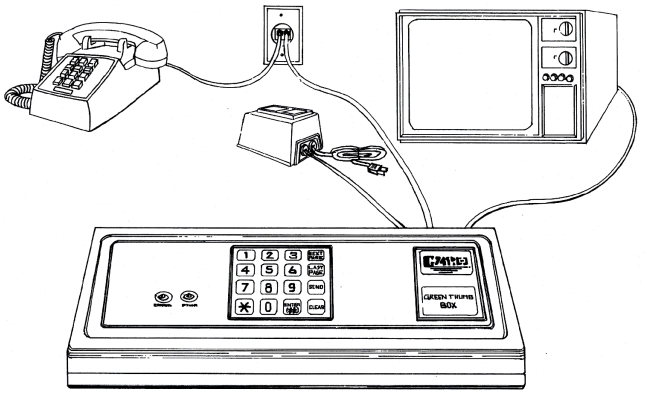


Figure 2.1

An illustration of the Green Thumb Box in operation. (From "The Green Thumb Proposal.")

64 × 32 coarse pixel resolution. A green-monochrome "full graphics mode" with a higher 128 × 96 resolution was also available.

Connectivity to the outside world was achieved by way of a built-in 300 baud* modem that connected the terminal to a telephone line,† which it shared with an existing phone by way of a splitter. To obtain information, the farmer simply picked up the phone and placed a call to a central computer, located somewhere within the county. After waiting for a few minutes for the data to download, the farmer would hang up the phone. Up to eight screenfuls, or pages, of information would then be available for off-line perusing, retained in the system's 4K or 8K of RAM as long as the unit stayed powered on.

As the project got underway, boards were soon produced and parts made available for assembly. For Jerry Heep, a former sergeant in the U.S. Marine Corps and an up-and-coming engineer who started working at Tandy in 1975, one aspect of the Green Thumb Box proved to be immensely challenging: the emission of radio frequency interference (RFI). RFI, a by-product of power being applied to electrical circuitry, can disturb or degrade the performance of onboard electric components or even those of other nearby electronics. Electrical engineers have wrestled with RFI for years in an attempt to manage it, since complete elimination was not possible. The Federal Communications Commission (FCC), the agency of the U.S. government charged with managing radio spectrum use, mandated limits on RFI in electrical products and forced electronic products to adhere to strict RFI guidelines in a section known as FCC Part 15.‡

It was not Heep's first run-in with RFI problems. The young engineer had wrestled with RFI before when designing an electronic game composed of a single integrated circuit. As Heep recalled, "You could hit that thing with a stick and the RFI would peak. It was totally unpredictable." Now, Heep was faced with trying to get the Green Thumb Box, with not one, but 28 integrated circuits, to pass the FCC's RFI tests. He warned management of the challenges that awaited them. Sure enough, try as he might, he simply could not bring the interference below the maximum allowable amount of 15 microvolts per meter. It did not matter that the terminal would be in the homes of farmers who often lived miles apart from their nearest neighbor, resulting in minimum risk of interference to others. The regulations had to be followed, and if engineering could not get the Green Thumb Box's electromagnetic interference down, the whole project could be doomed.

Once it became obvious that no amount of shielding was going to address the RFI problem, Tandy employed a clever workaround. "They issued me a

* Baud is equivalent to pulses or tones per second, so a 300 baud modem can deliver up to 300 bits of information per second.

† This is known as a direct connect modem, which was in stark contrast to most other modems of the time that instead featured acoustic couplers. These acoustic modems required that Bell System's standard telephone handset be placed into a cradle so it could listen for sounds that could then be converted to usable electrical signals.

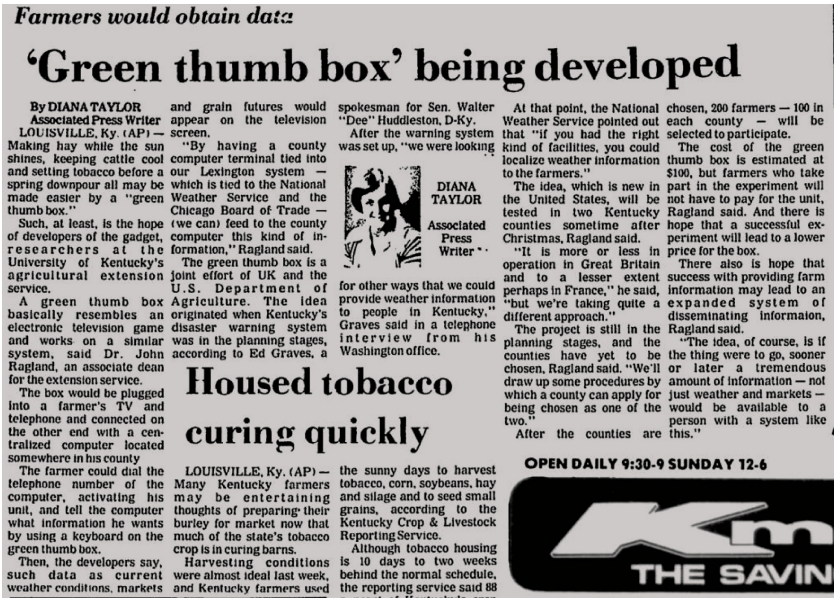
‡ It was a new revision of this same type of FCC regulation implemented on January 1, 1981, that caused Tandy to discontinue production of the noncompliant TRS-80 Model I in favor of the Model III.

transmitter's license, by the FCC, to operate an experimental television station," Heep later recalled. "The license would allow me to build transmitters which emitted a television signal." In essence, Heep's license allowed him to build and operate Green Thumb Boxes as individual little television stations. That stroke of end-run genius effectively got Tandy around the stringent FCC Part 15 regulations.

As Heep began to build the boxes, they were sent out into the field, distributed by county agents to farmers across the rural Kentucky countryside. Newspapers began to publish articles about Green Thumb, and awareness of the project started to build (Figure 2.2). Other parts of the country became interested in the promise of instant delivery of data to farms. Soon enough, over a dozen other states and their agricultural workers, including potato growers from as far away as Maine, wanted in on the action.

On March 3, 1980, after months of preparation, the switch was thrown and Project Green Thumb came online.

Project Green Thumb was deemed a success in meeting its objective as a useful tool for farmers. It was also a successful partnership between Motorola and Tandy, forging a working relationship in solving complex engineering problems that would continue into the coming decade. The project had also demonstrated that telecommunications was not only the way of the future, but with the right equipment and technology, something that was workable even under the most extreme and unusual circumstances.



Farmers would obtain data

'Green thumb box' being developed

By DIANA TAYLOR
Associated Press Writer

LOUISVILLE, Ky. (AP) — Making hay while the sun shines, keeping cattle cool and setting tobacco before a spring downpour all may be made easier by a "green thumb box."

Such, at least, is the hope of developers of the gadget, researchers at the University of Kentucky's agricultural extension service.

A green thumb box basically resembles an electronic television game and works on a similar system, said Dr. John Ragland, an associate dean for the extension service.

The box would be plugged into a farmer's TV and telephone and connected on the other end with a centralized computer located somewhere in his county

The farmer could dial the telephone number of the computer, activating his unit, and tell the computer what information he wants by using a keyboard on the green thumb box.

Then, the developers say, such data as current weather conditions, markets and grain futures would appear on the television screen.

"By having a county computer terminal tied into our Lexington system — which is tied to the National Weather Service and the Chicago Board of Trade — (we can) feed to the county computer this kind of information," Ragland said.

The green thumb box is a joint effort of UK and the U.S. Department of Agriculture. The idea originated when Kentucky's disaster warning system was in the planning stages, according to Ed Graves, a spokesman for Sen. Walter "Dee" Huddleston, D-Ky.

After the warning system was set up, "we were looking for other ways that we could provide weather information to people in Kentucky," Graves said in a telephone interview from his Washington office.

DIANA TAYLOR
Associated Press Writer

At that point, the National Weather Service pointed out that "if you had the right kind of facilities, you could localize weather information to the farmers."

The idea, which is new in the United States, will be tested in two Kentucky counties sometime after Christmas, Ragland said.

"It is more or less in operation in Great Britain and to a lesser extent perhaps in France," he said, "but we're taking quite a different approach."

The project is still in the planning stages, and the counties have yet to be chosen, Ragland said. "We'll draw up some procedures by which a county can apply for being chosen as one of the two."

After the counties are chosen, 200 farmers — 100 in each county — will be selected to participate.

The cost of the green thumb box is estimated at $100, but farmers who take part in the experiment will not have to pay for the unit, Ragland said. And there is hope that a successful experiment will lead to a lower price for the box.

There also is hope that success with providing farm information may lead to an expanded system of disseminating informaion, Ragland said.

"The idea, of course, is if the thing were to go, sooner or later a tremendous amount of information — not just weather and markets — would be available to a person with a system like this."

Housed tobacco curing quickly

LOUISVILLE, Ky. (AP) — Many Kentucky farmers may be entertaining thoughts of preparing their burley for market now that much of the state's tobacco crop is in curing barns.

Harvesting conditions were almost ideal last week, and Kentucky farmers used the sunny days to harvest tobacco, corn, soybeans, hay and silage and to seed small grains, according to the Kentucky Crop & Livestock Reporting Service.

Although tobacco housing is 10 days to two weeks behind the normal schedule, the reporting service said 88

Figure 2.2

An article on the Green Thumb Box as it appeared in the *Park City Daily News,* October 2, 1978. (From Google Newspaper Archive.)

From Project Green Thumb, destiny had spoken. It was time to take the idea to the next level, in another project that was already underway at Tandy: the VIDEOTEX.

The early success with Project Green Thumb had reinforced the fantastic idea that the future promise of personal computing was remote communication, and that the ability to connect people with on-demand information was already technologically feasible and financially practical. For the Green Thumb user, its implementation meant giving the farmer critical schedules on crop rotation or fertilizer applications. But what about other information? What if the idea behind Green Thumb could be expanded to provide a broader range of information like news, stock quotes, sports scores, and weather to households all across America? What if that data could be accessed by anyone, at any time? A myriad of intriguing possibilities existed.

Something else became apparent during Project Green Thumb: the box had taken advantage of two specific items that were in roughly 95% of American households by the late 1970s: a telephone and television. Tandy, through its 5,000 Radio Shack stores—2,000 of which were located in small towns with populations of less than 15,000 people—had been selling accessories for both devices for years. As a result, Tandy knew its customers' demographics well.

It was not a stretch then for Tandy management to postulate a world where they paired the average consumer's telephone and television to a box that could communicate to the world. The possibilities were staggering, the potential market was huge, and Tandy was uniquely positioned to take advantage of the opportunity.

This vision of connectivity for the common man was none-too-soon in coming, for in the summer of 1979, CompuServe launched a groundbreaking online service called MicroNET. Up to that point, CompuServe's options for remotely accessed information services was limited to professional use, with the Columbus, Ohio, computer services organization supporting more than 650 commercial customers, including government agencies, financial institutions, and other large organizations. Through MicroNET, the average consumer microcomputer user was finally brought into the fold.

Positioned as a way for CompuServe to sell after-business-hours computer time for its mostly idle DECsystem-10 mainframe computers, availability was initially limited but relatively inexpensive for the time. For a one-time hookup charge of $9 and then just $5 per connect hour, home and small business users could access the computer time-sharing and software distribution service in 25 major metropolitan areas* between the hours of 6 P.M. and 5 A.M. on weekdays, and all day Saturday, Sunday, and holidays. While this may seem restrictive now, for the inquisitive personal computer user of the time armed with any capable terminal, access software, and a 300 baud modem, it was the only cost-effective

* MicroNET services were available in 153 other cities for an additional $4 per hour charge.

professional alternative to dialing into hobbyist Bulletin Board Systems, or BBSs.* All that was needed now was for a company like Tandy to come up with a way for the average consumer to more easily access the fledgling service.

It therefore appeared to be fortuitous timing that with Project Green Thumb's launch, Tandy's mass-market-oriented successor, the TRS-80 VIDEOTEX Terminal, was already under development. The VIDEOTEX name was not chosen lightly, as "Videotex" was the generic name at the time for one-way ("Teletext" or, confusingly, "Videotext") or two-way ("viewdata") transmission using television screens as receivers. Videotex services were already rolled out to other parts of the world, with each based on their own standards, and Tandy was striving to be both literally and figuratively the name for such services in the United States.

Led by Heep, Tandy's VIDEOTEX, like Project Green Thumb, made extensive use of Motorola parts, but this time placed them inside a gray case reminiscent of the TRS-80 Model I's styling. A more versatile full 53-key chiclet QWERTY keyboard replaced its predecessor's 12-button keypad. A DATA indicator light was positioned on the top of the case, just above the keyboard to the upper right.

Internally, the VIDEOTEX consisted of three main components: the CPU, the video processor, and the direct connect modem. The CPU contained the advanced MC6809E 8-bit microprocessor, RAM, and supporting circuitry. The video processor was the same 6847 VDG integrated circuit used in Project Green Thumb, and provided the same display resolution and graphics modes. Similarly, the internal modem allowed for communications up to the same 300 baud speed.

Since its primary function was to provide access to remote online resources, the VIDEOTEX was only capable of limited offline activities through the latter two of its three operating modes: online, offline, and advanced. Online mode was for communication with a host computer, while offline mode was for viewing data that was automatically saved when in online mode. Advanced mode allowed offline data entry for sending when connected online to a host computer. The base model came with 4K of RAM that could hold up to eight screenfuls, or pages, of information, while expansion to 16K could save up to 32 screenfuls.

Despite its advanced internal architecture, the VIDEOTEX's external features, located on the rear of the unit, were minimal in deference to its intended usage. There was a Reset button, a TV output (TO TV) to an RF antenna switch box, a Channel Select (4 or 3) for the clearest television signal, a telephone port to connect to a modular phone jack, and a Power button (Figure 2.3).

Unveiled on May 27, 1980, access to the VIDEOTEX was initially limited to commercial enterprises and professional applications. Labels such as "AgVision," and partnerships such as the one with the Professional Farmers of America

* Telecomputing Corporation of America's The Source Information Utility was available around the same time as MicroNET and was similarly featured. Although no less of an online pioneer, The Source failed to achieve anywhere near the same subscriber numbers as CompuServe's service. A higher initial start-up fee to subscribe and greater hourly charges likely contributed to its limited success. In 1989, The Source was acquired by CompuServe and the service discontinued shortly thereafter. CompuServe itself was acquired by AOL in 1998 and, though a shell of its former self, continues to operate as an online services provider.

TRS-80 VIDEOTEX Introduces Instant Information

The TRS-80 VIDEOTEX Concept. Imagine having a public library, that never closes, right in your home or office! Or volumes of stock market data at your fingertips! Picture a continuously updated newspaper you don't have to fetch from your doorstep! Or a low-cost electronic mail service!

Now you can get quick, affordable access to many kinds of information and data services. Radio Shack's TRS-80 VIDEOTEX format—consisting of sixteen 32-character lines *plus* color graphics—makes it all possible. VIDEOTEX makes interactive information retrieval a reality *anyone* can take advantage of—today! You simply dial a phone number, press a few keys, and watch the desired data appear on your TV or personal computer screen!

Before long, you'll no doubt see many information suppliers offering data you can access with your TRS-80 VIDEOTEX system. You might even see electronic shopping, banking, car rental, classified ads, travel and theater reservations! But...

CompuServe® Information Service. Through an exclusive agreement with CompuServe, TRS-80 VIDEOTEX users in most areas can access a wealth of information on an inexpensive "local-call" basis right now. CompuServe offers local, national and international news, weather and sports from a major newswire service and major regional newspapers; historical information and daily updates on 3200 stocks, bonds and commodities; a home and educational reference service; entertainment news and reviews; and more! Plus, Radio Shack's own Bulletin service for TRS-80 owners.

VIDEOTEX is a *two-way* service, so CompuServe subscribers can also play computer games or communicate nationwide via an Electronic Mail service. And CompuServe is continually adding to its list of available services.

TRS-80 VIDEOTEX Software. Turn your computer or terminal into a VIDEOTEX information retrieval system with our exclusive software packages. You can access any VIDEOTEX information supplier with your computer or terminal and a 300-baud telephone interface. TRS-80 VIDEOTEX software comes with one free hour of access to the CompuServe information service plus your operators manual, identification number and password.

TRS-80 Color Computer Pkg. 26-2222* 29.95
TRS-80 Model I/III Pkg. 26-2220 ..... 29.95
TRS-80 Model II Pkg. 26-2221* 29.95
Dumb Terminal Pkg. 26-2224 ..... 19.95
Apple® II Microcomputer Pkg. 26-2223* 29.95
*Available Nov. 30, 1980

®Apple is a trademark of Apple Computer.
®CompuServe is a trademark of CompuServe Information Service.

A Truly-Affordable 2-Way Information Retrieval System

TRS-80 VIDEOTEX Hardware. The TRS-80 VIDEOTEX terminal works with *any* telephone to turn any TV set into a 2-way information retrieval system. It features an alpha-numeric keyboard and 4K memory that allows you to store information for later viewing—cutting "on-line" time to a minimum. The VIDEOTEX terminal provides fully interactive communication, so you can answer questions and exchange ideas, or even play computer games with other VIDEOTEX users. VIDEOTEX software is built-in, and we include one free hour on the CompuServe information service with purchase. The TRS-80 VIDEOTEX terminal is designed by Radio Shack and manufactured in our Fort Worth TRS-80 factory. It's easy to install—simply plug in a telephone cord, plug into an AC power outlet, and connect to the antenna terminals of your TV set. (Takes just minutes, and maybe a screwdriver!) 3½x13¾x14⅝." U.L. listed. Available November 30 at over 6000 Radio Shack stores and dealers nationwide.

TRS-80 VIDEOTEX Terminal. 26-5000 ..... 399.00

35

Figure 2.3

The TRS-80 VIDEOTEX Terminal as it appeared in the TRS-80 Computer Catalog, No. RSC-5, 1981. (Courtesy of www.RadioShackCatalogs.com.)

(PFA) to create the Instant Update database service, expanded upon Project Green Thumb's agricultural foundations and were used to brand VIDEOTEX Terminals. In some instances, these terminal cases were painted in a light blue color instead of the standard dull gray. The TRS-80 VIDEOTEX Terminal itself would be rolled out en masse to consumers in over 6,000 Radio Shack stores by November 30, 1980.

Tandy had no intention of leaving the burgeoning market of preexisting personal computer customers in the cold, either. VIDEOTEX-branded software packages started at just $19.95 (sans modem) and were released for everything from generic terminals to Tandy's own TRS-80 series computers. Even the competing Apple II got a version of VIDEOTEX.

While VIDEOTEX was designed to access its own custom services, BBSs, and the few commercial online services of the day like The Source and Dow Jones without bias, it was Tandy's exclusive two-year selling agreement* with CompuServe, which included Tandy-specific services, that was intended as its

* Michael A. Banks, *On the Way to the Web: The Secret History of the Internet and Its Founders* (Berkeley, CA: Apress, 2008), 61.

primary attraction. MicroNET soon began to be offered in Radio Shack stores, where many of the locations with computer centers were equipped with demo accounts to access the CompuServe Information Service.*

In turn, by the summer of 1980, MicroNET's availability at Radio Shack became a key component of CompuServe's own marketing, where the TRS-80 Model I, TRS-80 Model II, and VIDEOTEX were prominently featured in its advertisements (Figure 2.4). MicroNET and CompuServe Information Service users could gain access to all types of software, including networked multiplayer games, and a variety of personal, business, and educational programs, as well as 128K of online storage, bulletin boards for community affairs and for sale and wanted notices, electronic mail, and corporate stocks and public debt information. CompuServe promised even more was forthcoming to their rapidly expanding services menu, including regional newspapers, an electronic encyclopedia, travel information, and food preparation and gardening tips, all online features that, while strictly text-based, were still years ahead of their time.

Unfortunately for Tandy and their VIDEOTEX aspirations, while the idea that a large percentage of consumers would favor accessing online services over general computing activities would prove prophetic, it was approximately 15 years and one World Wide Web short of becoming reality. The fact that the VIDEOTEX Terminal was initially priced at $399 (about $1,100, adjusted), which made it comparable in price to a low-end personal computer, did not help its value proposition either. As such, the VIDEOTEX Terminal was a commercial failure. Even the competitively priced VIDEOTEX-branded software packages were soon surpassed in popularity by more versatile terminal software.

Although the VIDEOTEX experiment would prove a failure, luckily for Tandy, it was not its only new market strategy targeting personal computing at that time. Like many companies with talented leadership, a similarly big idea had already sprung up by the time the VIDEOTEX Terminal was under development. This formed a concurrent, two-pronged approach toward the rapidly evolving home computer market that ended up successfully acting as its own backup plan in case of one or the other's failure.

* In that same year, H&R Block would buy CompuServe to help develop electronic filings for tax returns. Although MicroNET was originally intended to represent the mainframe access services and CompuServe Information Service the information services, all such data offerings would soon fall solely under the latter name.

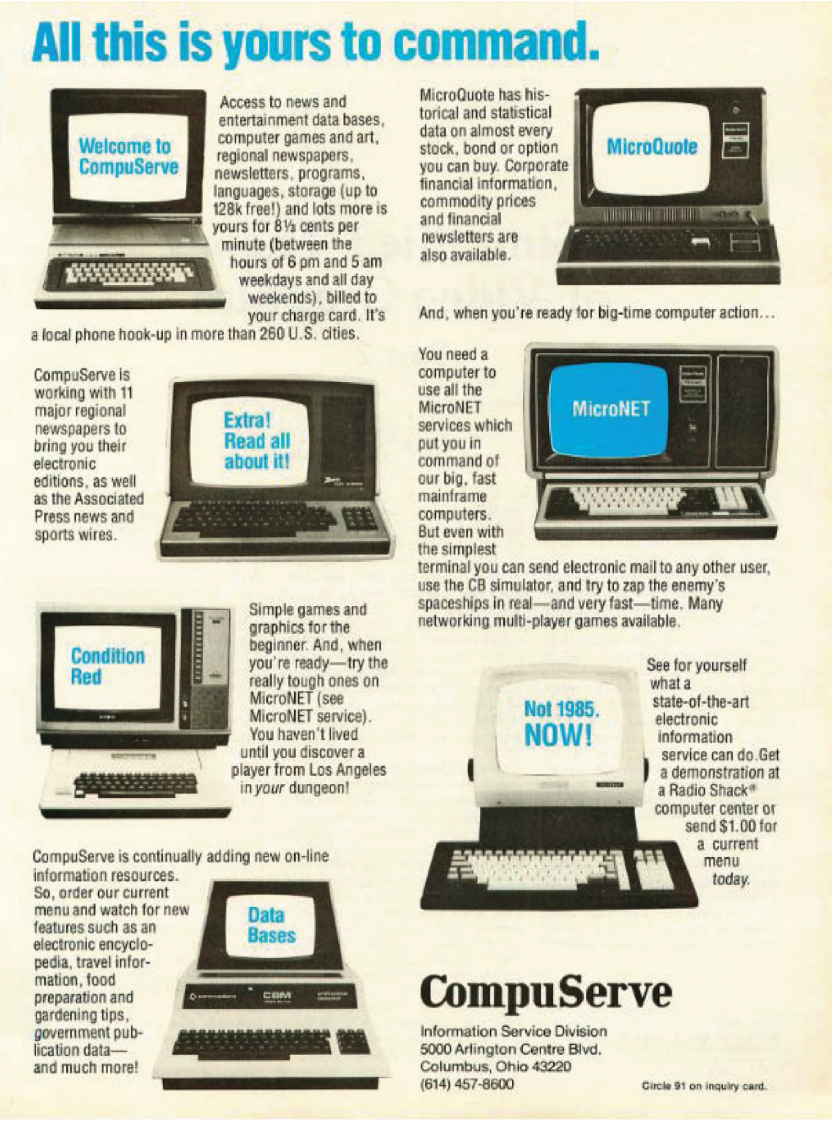


Figure 2.4

A CompuServe advertisement from the December 1980 issue of *BYTE,* featuring just three of Tandy's growing collection of platforms that could access the service.