Chapter 1. A Chronology of Game Programming

"In the beginning the Universe was created. This has made a lot of people very angry and been widely regarded as a bad move."

—**Douglas Adams**

KEY TOPICS

- Phase I: Before *Spacewar*
- Phase II: *Spacewar* to Atari
- Phase III: Game Consoles and Personal Computers
- Phase IV: Shakedown and Consolidation
- Phase V: The Advent of the Game Engine
- Phase VI: The Handheld Revolution
- Phase VII: The Cellular Phenomenon
- Phase VIII: Multiplayer Games
- In Closing

The art and science of game development have experienced a huge progression since the early days. Hardware has improved by orders of magnitude, whereas game complexity and richness have simply exploded. To better understand how games are coded today, and why, it is essential to look back and review the history of game programming. I have divided that lengthy narration into what I consider the eight most defining moments that shaped the current industry. Let's time warp ourselves to those moments and take a look at how things were done back then. This way the reasons behind today's practices will become obvious. I will avoid the trivial and focus on programming practices. Because raw data is useless without perspective, it's also important to provide some context in which to interpret the data and to understand how today's programming practices were developed.

Phase I: Before *Spacewar*

The video game industry didn't begin as part of any established master plan. Rather, it emerged when independent groups of people, working in different countries, came up with the same ideas at approximately the same time. These were traditional game creators looking for a way to diversify their business, technology companies taking advantage of newly discovered solid-state technology, and some isolated visionaries who imagined a new form of entertainment, which would grow with the passing of years into the industry we know today.

As you will soon see, the dawn of the video game age came in the early 1970s. But most of the companies that played a key role in that defining period of time were born much earlier. These companies started in different business sectors, but somehow ended up realizing that games played on electronic devices could effectively make money and become a new business sector.

An illustrative example would be the industry giant Nintendo, which was established as a traditional gaming company in 1889 by Fusajiro Yamauchi. Initially incorporated under the name Marufuku Company, its core business was to manufacture and sell Hanafuda, a type of Japanese playing cards (see Figure 1.1). In the year 1951, Marufuku was renamed The Nintendo Playing Card Company—Nintendo meaning "leave luck to Heaven." Later, as the first electronic games and devices appeared, Nintendo diversified its business by starting an electronic gaming division. As time went by, the traditional gaming business faded away, and Nintendo became the company we all know today. So Nintendo would be the perfect example of an already existing corporation that changed its sector of activity to embrace emerging technologies.

Figure 1.1. Hanafuda card.

Companies such as Sony followed a completely different approach. Created to focus on consumer electronics, the company known as Sony today was founded by Akio Morita and Masaru Ibuka as the Tokyo Telecommunications Engineering Corporation in 1946. The core business of the company was making tape recorders, which were miniaturized with the advent of solid-state transistors. As soon as the

company's products began reaching the European and American markets, the founding team decided to change its name to make it easier to remember by non-Japanese customers.

The company was thus renamed Sony (from the Latin word "sonus," which means sound) in an effort to make its brand easily recognizable. Sony quickly became one of the leading vendors in the consumer electronics arena, especially in the audio and visual areas. Brands like Walkman, Trinitron, and many others are a testament to the impressive product line it assembled. But Sony stood away from the gaming business until the late 1980s when work on the first Sony PlayStation began. The rest is history. Today Sony builds consoles and creates games in many studios worldwide. The PlayStation gaming platform is central to its overall business strategy, which has been successfully expanded outside the consumer electronics division.

A third, minor group of companies provided some middle ground between technology and gaming companies. Sega is a classic example of this group. Its story started in 1940 when Martin Bromely, Irving Bromberg, and James Humpert founded Standard Games, a coin-operated machine manufacturer in Honolulu. In 1951, the company moved to Tokyo and was later registered as Service Games (or, simply put, Sega) of Japan in 1952. The move made it easier for the company to supply coin-operated machines to U.S. military units stationed in Japan. Some years later, in 1965, Service Games merged with Rosen Enterprises, another company that dealt with everything from instant photo booths to mechanical arcade games. Rosen Enterprises had been founded in 1954 by Korean War veteran David Rosen. Rosen experienced firsthand the popularity of mechanical coin-operated machines (like the world-popular pinball machine) in U.S. bases stationed in Japan. Rosen began exporting them to Japanese territory under the name Service Games, or Sega. As the business took off, Rosen started producing his own games by purchasing a Tokyo-based slot machine and jukebox company.

The rest is history. Sega began producing arcade machines first and later expanded its operations to home consoles and game development.

However, it wasn't Nintendo, Sony, nor even Sega that led the way to electronic entertainment. These companies entered the emerging game industry following the footsteps of the true pioneers who came up with the initial designs and business model proposals. Clearly, someone with the vision of how games would be played on electronic devices was required to spark the process. That vision came from researchers working for universities and the military because they were the ones with access to cutting-edge hardware (according to the 1950s standards, that is).

The first of these early-day pioneers worked as a nuclear physicist at Brookhaven National Labs in New York. His name was William Higinbotham and he was a self-confessed pinball player. In the 1950s, Brookhaven was a government-supported research facility that focused on nuclear energy. Visitors toured the facilities, where peaceful uses of atomic energy were showcased. These included pictures and equipment displays, illustrating everything from power plants to radiation-based medicine.

Higinbotham, who thought those visits were boring, designed a strange device by using spare parts from the lab: an oscilloscope, some capacitors, two potentiometers, and a small analog computer. He dubbed the invention "Tennis for two" (see Figure 1.2). It was a simple two-player table-tennis game where the court and ball were displayed on the oscilloscope. The player could change the angle by which the ball was hit by turning the potentiometer. The game was mostly hard-wired, so it wasn't game programming just yet.

Figure 1.2. Tennis for two.

As with most geniuses, Higinbotham did not realize what he had achieved, not even when people started waiting in line to play the game at Brookhaven. The year was 1958, and by then other people had reached similar results worldwide. As early as 1952, A.S. Douglas presented his Ph.D. thesis on human-computer interaction at Cambridge, UK. As an example, he coded a tic-tac-toe game on an EDSAC computer to illustrate his principles.

Dozens of stories like these have been found relating to the 1950s decade, but Higinbotham's example is one of the best documented complete works from these early days.

Another visionary worth remembering is Ralph Baer, who came up with the home console concept as early as 1951. While working for Loral (an airborne electronics company), Baer got the assignment to create a cutting-edge TV set, which he proposed to enrich by using some kind of playable game. The company management ignored the idea, but 15 years later, while working for a different contractor, Baer gave his idea a second try. He succeeded this second time, and work began on what would become the world's first home console, the Magnavox Odyssey.

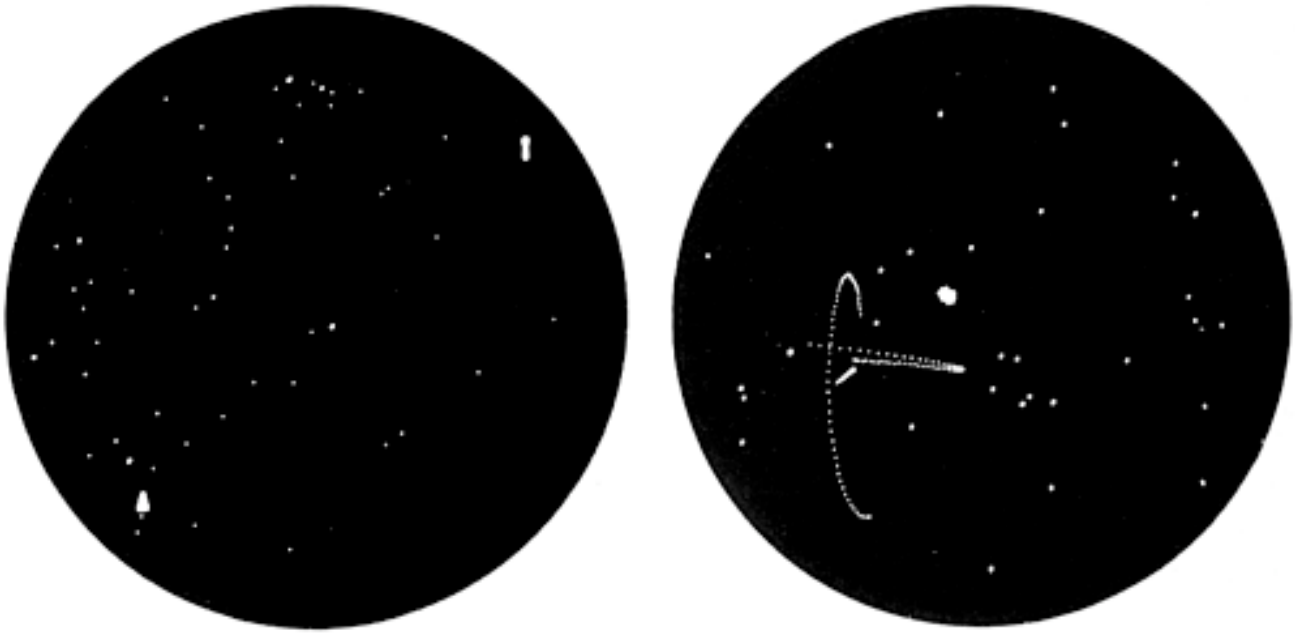
As a summary of this early age of development, by the late 1950s/early 1960s, some companies had developed solid positions in classic games (Nintendo, Sega, and so on). Other key players such as Sony and Matsushita were exploiting the benefits of solid-state technology. Additionally, some early pioneers were already aware of the potential of technology as a tool for play. Some test games surfaced—all implemented in specific hardware: Machines that effectively *were* the game. Game programming hadn't really appeared yet because programmable devices were rare. By 1960, a catalyst between traditional games, technology providers, and researchers was needed—a single successful game that would show where the three trends would merge to create meaningful business opportunities.

PREVIOUS NEXT

Phase II: *Spacewar* to Atari

The turning point for the games industry came in 1961 when Steve Russell, an MIT student, coded a simple two-player game on a Digital PDP-1 minicomputer. The game was called *Spacewar*, and it displayed two spaceships on a computer screen (see Figure 1.3). Each ship could move laterally while shooting at the other player.

Figure 1.3. Spacewar.



The game did not reach consumers but certainly served as an influence on many people. It was a game in the classic sense of the word. It used the new technology, and it defined the path for many others. The game had

- Competition between two peers
- Rules of action
- A clear victory condition

In fact, this structure is not very different from traditional games such as chess. The main difference is the technology layer that supports the gameplay. Through the years, this technology layer has skyrocketed in its complexity, and games have established themselves as a rich, unique media. The overall three-rule structure, however, has remained basically the same.

Many people were deeply influenced by *Spacewar*, but we will focus on two industry pioneers who were illuminated by the elegance of the concept. Nolan Bushnell was exposed to the game while studying engineering at the University of Utah. He envisioned computer games like *Spacewar* filling arcades, where people would pay to play game after game. Some years later, his vision would materialize when he founded Atari and created the first coin-operated (coin-op) machines.

The story of Atari is well known. After seeing *Spacewar*, Bushnell began working on reasonable-cost, dedicated machines where games could be played. His first game, years before the dawn of Atari, was called *Computer Space*, which was a version of *Spacewar* that he hard-wired and plugged into a TV set in his daughter's bedroom. Nutting Associates, a manufacturer of arcade games, bought the

Computer Space idea, hiring Bushnell to oversee production. In 1971, 1,500 *Computer Space* machines were manufactured and installed in the United States. But players found it too hard to play, so the game received a cold reception. Bushnell tried to create new game ideas, but after some discussions, he left Nutting. As a result, he founded Atari in 1972 with Ted Dabney, taking the name from the Japanese game of *Go*. Atari is the equivalent to a check move in chess.

On the other end of the spectrum stood Ralph Baer, the games-on-TV pioneer from the 1950s. By 1966 he had already left Loral and was working for Sanders Associates, an army contractor. He was given the green light to research his TV set idea, which he patented in 1968. He saw electronic games as secondary uses for TV sets, which already enjoyed a large installed base. At that time, he had succeeded in creating two TV-based games (a chase and a tennis game) as well as a very early light gun design. By 1970, TV manufacturer Magnavox had licensed Baer's technologies. Under Baer's guidance, work on the first game console began. The system, dubbed the *Magnavox Odyssey*, was introduced in 1972, the same year Atari was born.

By the end of 1972, the electronic games business took the world (starting with the West Coast of the United States) by storm. The first Atari game, *Pong*, hit locales, becoming the first successful example of a coin-op machine. Within two weeks, *Pong* machines in California began to break down due to quarters flooding the coin drop mechanism, something Bushnell had never even dreamt of. At the same time, Magnavox sold 100,000 units of the *Odyssey*, a remarkable feat considering distribution was made only through official Magnavox stores.

During this period, two distinct business models appeared: arcade games, paid per play; and home systems, where games could be purchased and played repeatedly. Both have evolved and still subsist today, with the latter being the dominant option. A third trend had yet to appear; one in which games needn't be played on dedicated hardware.

Phase III: Game Consoles and Personal Computers

Electronic games, either for home systems or arcade machines, were an instant success. Different manufacturers such as Coleco (short for Connecticut Leather Company, a clear example of extreme diversification), Fairchild, and many others introduced their own home gaming systems, and competition fostered innovation. Game cartridges were introduced in 1976, trackballs in 1978, and so on.

Game Consoles and Game Developers

It was during this fast growth period, sometime in the 1970s, that the distinction between console manufacturers and game developers was established. Console manufacturers would invest large sums of money in creating the hardware platform and selling it at an attractive price to consumers. Sometimes, the price would hardly cover (or even be below) the real cost of the hardware. Manufacturers would also build tools (very primitive at this stage) that allowed outside companies to create games for the console. The reason why other companies were given access to the business was obvious. As game consoles began to compete against each other for market share, being able to offer many games became the main competitive advantage, and hence the importance of the developer. No manufacturer could ever have the bandwidth to deliver the amount of games required to make its console platform attractive.

Thus, outside companies began coding games for console systems. At this stage, little or no quality control was performed by the console manufacturer; quantity was valued over quality. For each game sold, the developer agreed to pay a royalty fee to the console manufacturer to cover the console development costs and thus generate revenue. Some console manufacturers focused on the hardware business only, whereas others chose to act as developers as well, sometimes causing conflicts of interests. Did the outside developer have the same information as the internal team at the manufacturer? Sometimes first-party teams had access to better information than third parties, which caused complaints. With some variations, this production model still applies today. The main difference with today's model is that consoles are sold way below their manufacturing price, especially when they debut. This is a risky decision, because the hardware manufacturer has to sustain significant losses for some time. Then, as the console manufacturer begins receiving royalties from developers, it is able to recover from the initial losses and finally make money.

The king of programmable consoles at the time became the Atari VCS, also known as the Atari 2600, which was introduced in 1977 with a price tag of $249. Because it became the de facto standard, I will use it to exemplify what coding for a console meant in the 1970s. The console featured a 6507 CPU equipped with 128 bytes of RAM, which were used to store state variables such as the life and ammunition levels. The program was stored in an external, interchangeable cartridge, which was inserted into the console. One side of the cartridge was full of electrical connections. Thus, inserting it into the console integrated the data chips in the cartridge as part of the console's hardware, usually as memory banks where the program code and data were stored.

The memory cartridge method was used with minimal variations until the Nintendo 64 and the Gameboy Advance, whereas more modern systems employ different kinds of disks as storage media. Storing games on CD-ROMs and DVDs makes them much cheaper to manufacture, at the cost of increasing the risk of piracy and having to enhance the memory capabilities of the console to store the program. But for the Atari 2600, 6KB of ROM were usually built into the cartridge and held the game code and data together in a memory chip. The 6507 CPU ran at an incredible 1.19 MHz and had an address space of 8KB.

Assisting the CPU were the television adapter, known as TIA or Stella, and the I/O chip, called RIOT. Stella was accessed through registers and had the extreme limitation of not having a frame buffer or physical memory to hold the screen's contents. Modern adapters (from CGA onward) have a memory region that holds the screen data pixel by pixel, so copying data to that region effectively paints the screen. The Atari 2600 did not have such a buffer. So, the screen was drawn by reading some Stella registers serially, synchronized to the electron beam from the TV set. The CPU had to synchronize itself with the electron beam and write those registers at exactly the right speed, so it looked correct and produced no flicker. As an example of the limitations imposed by the hardware, here is the sequence of a game loop for the Atari 2600:

Start the vertical blanking interval
Start the vertical sync interval
    Here is space for 80 micro instructions

```
End vertical sync
    Perform game computations here
End vertical blank
Now the screen rendering starts...
    Send each line to the register
    6 instructions can be fit here
Loop lines until the screen is rendered
Go to first step
```

The 2600, as all consoles were at that stage, was coded in machine-specific assembler. Program data and source code were part of a single memory block. Data was just bytes of information trailing after the code, which program code never jumped to. An untested program that began to execute data addresses would simply go mad. Here is an excerpt of a combat game for the 2600, showing how code and data were part of a whole:

```
B15A7      STX  D2
        LDX  #03
B15AB      LDA  L1765,Y
        EOR  D1
        AND  D2
        STA  COLUP0,X
        STA  D6,X
        STA  D8,X
        INY
        DEX
        BPL  B15AB
        RTS
J15BD      LDA  #00
B15BF      INX
        STA  A2,X
        BNE  B15BF
        RTS
L15C5      .BYTE  $0E ,$0A ,$0A ,$0A ,$0E ;  0
        .BYTE  $22 ,$22 ,$22 ,$22 ,$22 ; 11
        .BYTE  $EE ,$22 ,$EE ,$88 ,$EE ; 22
        .BYTE  $EE ,$22 ,$66 ,$22 ,$EE ; 33
        .BYTE  $AA ,$AA ,$EE ,$22 ,$22 ; 44
        .BYTE  $EE ,$88 ,$EE ,$22 ,$EE ; 55
        .BYTE  $EE ,$88 ,$EE ,$AA ,$EE ; 66
        .BYTE  $EE ,$22 ,$22 ,$22 ,$22 ; 77
        .BYTE  $EE ,$AA ,$EE ,$AA ,$EE ; 88
        .BYTE  $EE ,$AA ,$EE ,$22 ,$EE ; 99

L15F7      .BYTE  $F8 ,$F7 ,$F6 ,$06 ,$06
        .BYTE  $06 ,$16 ,$17 ,$18 ;     $15FC
        .BYTE  $19 ,$1A ,$0A ,$0A ;     $1600
        .BYTE  $0A ,$FA ,$F9 ,$F8 ;     $1604
        .BYTE  $F7 ,$F6 ,$F6 ,$06 ;     $1608
        .BYTE  $16 ,$16 ,$17 ,$18 ;     $160C
        .BYTE  $19 ,$1A ,$1A ,$0A ;     $1610
        .BYTE  $FA ,$FA ,$F9 ,$E8 ;     $1614
        .BYTE  $E6 ,$E4 ,$F4 ,$04 ;     $1618
        .BYTE  $14 ,$24 ,$26 ,$28 ;     $161C
        .BYTE  $2A ,$2C ,$1C ,$0C ;     $1620
        .BYTE  $FC ,$EC ,$EA ;          $1624
```

Most of the time, games were built by a single person who laid down the memory map, wrote the program code, designed the graphics, and even provided the sound. Sometimes an artist helped out, but doing graphics is a relatively straightforward task on such limited hardware. Code reuse, so popular today, was virtually nonexistent. At most, programmers would borrow sequences of microinstructions from a previous title so a new game could be coded faster. But generally speaking, coding for such a machine involved lots of craft and skill, as anyone who has coded assembly programs for a living knows.

Personal Computers

While consumers were busy playing with game consoles, the personal computer revolution was about to begin. Computers appeared as a by-product of the discovery of the integrated circuit by Texas Instruments in 1959. The first computer, the Digital PDP-1, appeared in 1960 and featured a keyboard and a monitor. It debuted at $120,000, clearly positioning computers in the corporate market: Accounting, engineering, and databases were their main uses. By 1964, the BASIC language appeared, allowing intuitive programming.

Douglas Engelbart invented a two-dimensional pointing device called the "mouse," which is now widely used. By 1968 (one year before humanity reached the moon), the first home computer, the Honeywell H316, was introduced at $10,600. The Apollo Guidance Computer, developed specifically for the Apollo 11 spaceship, was an impressive feat. It ran at 2MHz, had 64KB of ROM and 4KB of RAM, and was capable of performing 50,000 additions per second.

In 1972, when Atari and Magnavox became successful, computers were gaining ground. The C programming language was introduced, the first ideas about laptop computers appeared at Xerox's Palo Alto Research Center (PARC), and Intel introduced the 8008, a 16KB, 200kHz, 60,000 instructions-per-second, low cost chip. From this moment on, computers advanced at blazing fast speed. In 1974, Intel unveiled the 8080, a 2MHz, 16KB, 640,000 instructions-per-second chip. That same year, the MITS Altair 8800 was announced in Popular Electronics for about $400.

One year later, Microsoft was founded. Its flagship product was a BASIC interpreter for the Altair. It was the first programmable language for a personal computer. Years later, Microsoft would create the operating system for the IBM PC, and the rest is history.

Then, in 1976, one of the most popular garage startups surfaced, and the personal computer industry was born. Apple was founded by Steven Jobs and Steve Wozniak. Jobs, with an eye on product design and business, had been at Atari since 1974. Wozniak, the hardware genius, worked for Hewlett-Packard. Their first product, the Apple I, sold in kit form for $666, and was introduced at the Homebrew Computer Club. Some simple games began to appear for the Apple I, but they were mainly clones of those existing on game consoles.

But it was with the advent of the Apple][in 1977 when games began to pour in quickly. A decisive factor in this direction was the Apple][']s CPU: a 6502, extremely similar to the one running Atari's own 2600. The computer featured new 16KB RAM modules to reach a theoretical 48KB of total RAM, which was larger than anything on the market at the time. The bundle also included a BASIC interpreter to code your own programs, a QUERTY keyboard, a cassette interface, and a dedicated game I/O connector for paddles, joysticks, and so on. It could display up to 280*x*192 pixels in 16 colors, making it an ideal delivery platform for games.

The Apple][was state-of-the-art technology, clearly surpassing the products of the competition, including game consoles. It could be programmed, used for games and office applications, had a large memory area, and offered "full" color support (the full 16 colors in the palette, that is). The downside was, obviously, the cost. The base kit cost $600, whereas a full-featured, 48KB "supercomputer" was $2275, which was mainly due to high memory costs of those early days. But Apple][s were sold by the truckload, and hundreds of games were coded for it—a notorious example being the first installments of the popular *Ultima* series.

Five years later, IBM unveiled the IBM PC, which ran on an Intel 8086 CPU and Microsoft's Disk Operating System (MS-DOS). The main advantage of the PC was that the architecture was soon to become open. This enabled other companies to design and sell their own PCs, which were all compatible with the original IBM model. Competition fosters innovation, and an open platform is the best way to enable evolution. The personal computer era had begun.

Phase IV: Shakedown and Consolidation

With the Atari 2600 and Apple][on the market, there was a clear sense of direction in the games business. Thus, the period comprising the late 1970s and early 1980s is sometimes dubbed "The Golden Age." All that was needed was to keep improving these initial products, to create better platforms and games at each iteration. On the console side, Atari reigned supreme for many years, even when Nolan Bushnell left the company in 1978.

Following the departure of its founder, the company enjoyed prosperity due to the huge number of games being published. This period is also characterized by the entrance of Japanese developers into the equation. Taito (a pachinko manufacturer) introduced *Space Invaders*; Nintendo (remember the Hanafuda card company?) began by selling an *Othello* arcade game only to later introduce *Donkey Kong* (the first game by now-celebrity designer Shigeru Miyamoto) in 1981; and Namco (known as the Nakamura Manufacturing Company, maker of merry-go-rounds) introduced such classics as *Galaxian, Pac-man, Galaga,* and *Pole Position*.

Pac-man, whose initial character concept comes from a pizza with a slice missing, is based on a Japanese folk hero called Paku, known for his appetite. The game was to be called Puckman, but the potential risk of American graffiti artists changing the P to an F made Namco go for *Pac-man* as the final name.

Atari's golden days were about to end. The company did not keep a careful watch on the quality of games for the 2600, and some highly anticipated games for the platform were a flop. *Pac-man* and *E.T.* games were released with great hype, only to see consumers reject them due to their low quality. The legend says that thousands of copies of these games ended up in a massive landfill in New Mexico. By December 7, 1982, Atari announced that 2600 sales had, for the first time, failed to meet predictions, making its parent company, Warner Communications, suffer a 32 percent stock price drop in a day. In early 1983, with Atari still struggling to recover from *E.T.*, a crash occurred in the video game market. Retailers had tons of unsold games on store shelves, and many of them offered very low quality. Game prices dropped sharply. Titles that previously sold for $40 were selling for 99 cents, which resulted in many companies going out of business. Others would survive, but would never quite recover from the hit.

It took a while for the industry to recover from such a bad experience. Luckily, by 1985, some companies introduced a new wave of products and a revised quality policy. Nintendo introduced the Nintendo Entertainment System (NES) (Famicom in Japan), backed by a roster of internally developed, high-quality games. Shortly thereafter, some of the Atari developers such as Namco became Nintendo licensees. By 1988, classics such as *Super Mario Bros* and *Legend of Zelda* had been introduced, making Nintendo the top seller. Sega did not stand still either, introducing the Master System in 1986. Internally, both the NES and Master System were equipped with 8-bit processors (a 6502 and a Zilog Z80, respectively). Spec by spec, the Master System was a slightly better platform: More sprites were supported, and RAM size was increased. But Nintendo had the advantage of being the first mover with an impressive game catalog and reputation.

During these years, the consolidated console business model finally became clear. The console manufacturer created the hardware and produced (internally or through contractors) a first batch of titles, which showcased the hardware's capabilities and helped sell the first units. Then, developers wanting to jump on the bandwagon needed to apply to become licensees, and each game they built had to be tested for quality by the manufacturer. Only by enforcing this policy could consoles offer a consistent-quality game catalog, which made consumers happy with their purchases. Developers still received documentation and sometimes technical support from the console maker. They also had to pay a royalty fee to cover the console's development cost. In the case of Nintendo, there has always been a very strong presence of Nintendo-built games for its game systems, which ensures that a core of high-quality games is always available. By 1988, Nintendo was an established market leader with a well laid out licensing policy.

Let's press the pause button and explore the internals of the flagship product for this era, the NES. To begin with, Nintendo's console was built on top of a customized 6502 MOS CPU, which was enhanced to perform audio waveform computation directly on the CPU. As with all 6502s, the NES could only address 64KB of memory, but some techniques were devised to overcome that limit. Cartridges could implement a paged approach using a memory mapper to increase the addressable memory.

Graphics chores were carried out by the picture processing unit (PPU), whose main feature was support for tile-based backgrounds, sprites, and scrolling on hardware. Contents of the screen were described using four main data structures: the pattern table, the name table, the attribute table, and the sprite table. The *pattern table* was a list holding the definition of each 8×8 tile or pattern. Each tile was defined by a sequence of eight 16-bit values, each one representing one row of the tile. The idea is pretty straightforward. Each pixel in the tile can have one of four possible color values, and thus two bits per pixel are required (hence the 16 bits per row). So one tile in the pattern table could represent four different color values.

The *name table* assigned tiles to screen coordinates. The name table was a two dimensional, 30 row by 32 column matrix, with each position exactly one byte in length (thus selecting from 256 possible tiles). Multiplying the name table dimensions by the tile size would produce the total size of the background: 256 by 240 pixels. This was more than the available screen resolution, so only a portion of the

background was visible at any given time. This background could be scrolled by simply changing the value of two offset registers.

An *attribute table* was used to modify how tiles were to be mapped to the screen. Attributes were specified for each 32×32-pixel screen block (which means one block every 4*x*4 tiles), so all tiles in that area shared the same attributes. Attributes were used to further refine the color scheme to be used. The attribute table provided two high-order bits for the color, whose two low-order bits were taken from the pattern table. This way a maximum of 16 colors could be used.

The *sprite table* was used to overlay sprites—characters, potions, and so on—on top of the background. The PPU could store up to 64 sprites (8*x*8 or 8x16 pixels each), which could be quickly mapped to the screen. Sprites were prioritized; whichever sequence you decided to paint, lower-number sprites were painted after higher-order ones were processed, thus providing an easy way of layering information onscreen. The main difference between sprites and tiles was that sprites could be blended with the background in a realistic manner; a key value was provided to specify which portions of the sprite were actually transparent and should expose the background.

As these special structures illustrate, the NES was a dedicated game platform. Its structure was clearly limited to what it was supposed to do. This is a common characteristic of all game consoles: They excel at what they were designed to do, but aren't useful for much else due to the system's design decisions.

The evolution of personal computers did not stand still for a decade. The Apple][had marked the course for the industry. Computers that sold at a reasonable cost were useful as working tools, but also kept the door open for home computer gaming. Each new iteration offered more powerful hardware, easier to use programming environments, and a more compelling game experience. The introduction of the IBM PC in 1981, with its open hardware design, fostered innovations by specific peripheral manufacturers. Graphics cards and hard drives began to be manufactured by different companies; each one competing for a piece of the market share. The evolution was enormous: From the crude, 4-color CGA adapter to the 256-color, 320*x*200 VGA adapter, only five years passed. VGA was the graphics mode that made computer games so popular in the late 1980s. It offered full-screen titles with a resolution similar to game consoles and color depth to display rich game worlds. Masterworks such as *Day of the Tentacle* by LucasArts or the*Alone in the Dark* saga are classic representatives of the VGA days.

Phase V: The Advent of the Game Engine

As soon as file systems appeared on the first home computers, it became obvious that they would be a powerful tool to better organize content in games. Gone were the days of a single file mixing code and data, making projects virtually impossible to maintain. With a decent file system, one file would contain the game code (the executable file), and several files would contain the associated data. This way chaos would be minimized, and work could be divided between different people who would only care about their specific files. A musician need only access the music data, the sprite artists need only work with bitmaps, and so on. Keep in mind that hardware was evolving quickly, and larger games were difficult to create without some proper organization. However, a side effect of this organization soon surfaced: What would happen if you had a single executable file that used data files laid out in a documented format, so you could easily replace the game's data files? Effectively, this enabled the creation of many games that were identical in their gameplay formula, but different in their content. You could do a scroller set in World War II and another one in outer space without touching a single line of the source code. This may seem naïve by today's standards, but try to imagine the revolution it meant for an industry where each game was an indivisible atom with code and data all mixed up. Being able to create different games by modifying the relevant data files was a transformation. Clearly, this was the way to go for the industry; and soon reusable data-driven game engines became ubiquitous because of their obvious economic advantage over hard-coded games.

In these early games, data files were nothing but memory dumps of the data structures, with little or no structure at all. Soon, developers began creating their own file formats, so no one could steal their artwork by looking into the files. Competition fosters innovation, so better systems were devised. At the same time, developers began thinking about more intelligent ways to organize their work using file systems. One of the best ideas they came up with was to use files not only for data, but also for some behavior information. Choreographed trajectories for a scrolling game's AI system could be stored this way. The choreographed AI that made most scrollers famous could be thought of not only as pure data in the form of waypoints, but also as behaviors. Behaviors can be programmed as patterns of movement that actually define the gameplay. Thus, the leap from data files to behavior systems was made. By redefining graphics, sound, and behaviors, game developers and players could quickly derive new games from old ones. Not only did this add flexibility to the development process, but it also enabled the player community to create vast collections of content. Users creating high-quality content were frequently offered jobs at the companies developing their favorite games.

Defining behaviors in the data files brought some changes to the internal game code. What was initially a totally hard-coded gaming system became a bit like an application loader and operating system, providing some internal functionality and interpreting the remainder from the data files. The internal functionality was publicized in an interface that content developers had to learn, and they then used it to implement subsequent games. Thus, game developers began to think not so much in terms of a specific game, but about generic game systems that played different games depending on the data files. The more flexible the engine was the better. Most companies kept their file formats well hidden, partly because of concerns over intellectual property and piracy.

A prototypical example of this first generation would be the Script Creation Utility for Maniac Mansion (SCUMM) system devised by LucasArts. It was a complete authoring system that helped define the game structure, dialogs, locations, and other data needed to implement many popular adventure games. It was first used in games such as *Maniac Mansion* and *Loom*, and was employed (under different versions) in most LucasArts adventures up to *Curse of Monkey Island*.

By the late 1980s/early 1990s, some engine standards began to surface, and their specs were made public, so users could create new missions and content, and thus lengthen the life of the game. The best-known example of this era is the incredibly popular *Doom* engine, introduced by id Software in 1993. After enjoying a first hit with *Wolfenstein, Doom* can be considered the game that defined the first-person shooter genre. Its maps and weapons are still venerated by today's standards, and many games have paid tribute to *Doom* in some form or another. Internally, *Doom* was a radical follower of the engine philosophy I have exposed. Both data and behaviors were implemented via external files, so the binary game file was really a level loader. In the case of *Doom*, all engine data was encapsulated in large files using the .wad extension, and were thus called WAD files.

A WAD file was just a collection of data for the game. Data was organized in large blocks; each containing a sequence of one type of information: the level geometry, monster placement, and so on. Each of these blocks was called a lump. Then, the complete WAD file was composed of three main parts:

- A 12-byte file header
- A directory containing the names, offsets, and sizes of all lumps in the WAD file

- One or more lumps with the actual data

The header consisted of a four-byte initialization string, which could be Internal WADs (IWADs) or Patch WADs (PWADs). An IWAD contained all data necessary to play the game, so all entries of the file were full. A PWAD, on the other hand, was just a patch to the core IWAD, so only some of the lumps were present. Logically, all user-created WADs were PWADs, which provided only the lumps required to define a level. Right after the header, four bytes were used to store the number of lumps in the WAD file, and the last four bytes stored a long integer that was the file offset to the beginning of the directory to make file traversal faster.

Then, the directory was just a list of entries; each taking 16 bytes. Here, the first four bytes stored the file offset to the start of the lump. Then, the middle four values stored the lump size in bytes. The last eight bytes were available to store the name of the lump, padded with zeros. To give you an estimate, there are about 2,000 entries in the directory of the main *Doom* WAD file. But only 10 types of lumps are needed to create a new level in a PWAD.

Now we are reaching the interesting part: the lump definition. We will focus on level creation because there are many other lumps that hold information such as menu texts and character sets. A *Doom* level needs several types of lumps. The most interesting are

- **Linedefs**. A list of two-dimensional line segments that have a meaning for the game engine. These lines are mainly used as walls to block characters, although they can also block sound propagation. Part of the geometry of the level is stored this way (remember that *Doom* was a 2D game internally!).
- **Sidedef**. A definition of which texture(s) to use with each linedef. This describes how the walls look.
- **Vertices**. Core vertices use linedefs. The linedefs indirectly refer to the two-dimensional vertices in this list instead of explicitly containing their own coordinates. This way, two adjacent linedefs can share the same vertex, saving memory and making editing easier.
- **Nodes**. A subdivision of the space according to a two-dimensional Binary Space Partition (BSP). A BSP is a data structure that can efficiently classify geometrical information, such as triangles on a game level.
- **Things**. A lump that contains positions for relevant items: weapons, poisons, keys, player starting positions, and so on. Thus, it greatly affects the gameplay.

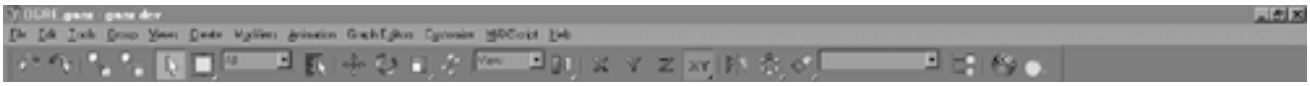
Lumps can define both the contents of the level and gameplay elements as well (for example, the Things lump). This effectively meant that new games could be derived from the base *Doom* product quickly. In fact, *Hexen* is nothing but the *Doom* engine with a new set of data files and some minor additions to the WAD format (such as the capability of some potions to appear only for characters of some special class). *Doom* had a great influence on developers, who followed the trend and began providing complete documentation and even editing tools to encourage user-created modifications. In fact, the popular game *Half-Life* began as a modification to the game *Quake II*, which grew so much that a new, completely different game was born.

Today, the concept of a game engine has evolved a great deal since *Doom*. The core game code is kept to a minimum, providing only the reusable algorithms and functionality that is essential to the game. This includes all time-critical tasks such as the rendering pipeline, networking and audio APIs, and AI interfaces where external modules can be docked. Usually, part of the higher level AI is implemented through structures and scripts in external data files. Lower level, more time critical, and less variable parts of the AI are often implemented in the engine itself. Many games use a mixture of both approaches.

In addition, extensive toolsets are provided to create content for the game. Some products use their own content creation systems (CCS), whereas others have relied on heavily modified versions of off-the-shelf products.

Gmax (see Figure 1.4), for example, is a content creation system built on top of 3d max's core. It can be bundled with your game, so enthusiastic users will be able to create new game content. Coming from max's lineage, Gmax is fundamentally a modeling and texturing package, so it focuses on level creation, characters, and so on. Games such as Microsoft's *Flight Simulator* have included Gmax with their release. The end user gets the software for free (or included in the cost of the game), whereas the developer must have a licensing agreement with the tool manufacturer—discreet in the case of Gmax.

Figure 1.4. Gmax.



Maya hasn't stood still, either.

Maya Personal Edition has been successfully bundled with popular games such as *Unreal Tournament 2003*. In this case, the game developers created a set of plug-ins for Maya PE, so users can export their data sets made with Maya to *Unreal*'s own formats. Again, the software comes at no cost for the end user.

But there is more to game engine content creation tools than modelers. All commercial engines must come with full documentation because the level of complexity and flexibility has skyrocketed with each new iteration. Keep in mind that these engines are not only sold on an "as is" basis: Some developers will choose the advanced track and hard code new features into the core toolset. A typical example is *Half-Life*. Valve, the developers of the game, needed a skeletal animation system that was (at the time) significantly more involved than the key frame-based animation system provided by *Quake*. Thus, after purchasing the engine license, they had to rewrite the animation system from scratch to support the new feature.

Phase VI: The Handheld Revolution

The NES wasn't Nintendo's first electronic device. In the late 1970s, the company became well known for a family of low-cost, simple electronic games called *Game & Watch*. These were handheld devices about the size of a cassette tape, costing between $10 and $25, which came with a single game to play. *Donkey Kong* was introduced on such a platform in split-screen format. Games were just simple combinatorial systems displayed on low-cost LCD screens. But Game & Watch machines were sold by the millions and became extremely popular, especially among young audiences who could take them anywhere and share them with friends.

Interest in Game & Watch faded away as the domestic console market exploded. Such a handheld system could not compete with the color and sound found in the titles that appeared both on the NES and the Sega Master System. But in 1989, when everyone had long forgotten about the Game & Watch, Nintendo released a game console that would become its most profitable product ever: the Nintendo Gameboy. It was a handheld, black-and-white console that could play literally hundreds of games by replacing a cartridge, just like a regular home-based console.

The Gameboy was an instant hit, which can partly be explained by the inclusion of the *Tetris* game with the purchase of the console. The product was targeted at younger audiences, using the same strategy that made Game & Watch so popular in the past. Some Gameboy classics were *Pokemon* and a special version of *Super Mario Bros*. Internally, a Gameboy was inspired by the design of the original NES. It used an 8-bit CPU similar to the Intel 8080 or Zilog Z80 and was armed with 8KB of code RAM and an additional 8KB of video RAM. The CPU ran at approximately 4MHz. Screen resolution was 160*x*144 pixels (20*x*18 tiles) but, as with the NES, this was a window of an internal memory representation, which was 256*x*256 pixels (32*x*32 tiles) across. This allowed fast scrolling by only changing two registers, SCROLLX and SCROLLY, which defined the offset of the said background to the screen. The background was painted with tiles, which were taken from a tile data table. Thus, the screen (called the Background Tile Map) consisted of only 32 rows of 32 bytes each for a total selection of 256 tiles. Each position held the identifier of the tile to be drawn there.

The Gameboy supported one overlay level, so a menu or scoreboard could easily be drawn on top of the scrolling background. This was achieved with a window that was not scrollable but painted at a fixed onscreen position. Window contents were taken from the tile data table as well, so you could position the window onscreen easily with two registers, WNDPOSX and WNDPOSY.

As with the NES, a Gameboy could paint keyed sprites on top of the background for characters, enemies, and so on. It could use both 8*x*8 and 8*x*16 sprites, and up to 40 were available. As with the NES, only 10 sprites could be displayed per scanline due to a hardware limitation. Sprite patterns were taken from the sprite pattern table and layered on the screen according to some very unintuitive priority rules: Sprites closer to the left end of the screen would have priority, and thus be laid on top of others; if two sprites happened to share the same *X* coordinate, the one located in lower positions in the sprite table would have priority.

A Gameboy was a pretty powerful piece of hardware by 1989's standards. It was a mini-version of an NES in black and white, which made it cheaper to manufacture. But what set the Gameboy apart from the competition was its lengthy battery life and the vast array of quality games available for the platform. After all, creating titles for the Gameboy was a very profitable business. It was orders of magnitude cheaper than coding for home-based game consoles (especially since the PlayStation and N64 hit the market). But, on the other hand, the price of the games was not that different. In other words, there was a great margin both for the console manufacturer and the software developer.

Among all gaming platforms, the Gameboy has definitely been the console with the longest life cycle (11 years), and some games are still being sold today. The release of the Gameboy Color in 1998, along with new iterations of classic Nintendo titles, breathed new life into the product, making it break all established records. But even great products like the Gameboy grow old, and thus by the year 2000, Nintendo had already decided to release a newer, more powerful machine. The Gameboy Advance (GBA) was designed with the mission of becoming the substitute for the original Gameboy. The GBA specs are living proof of this philosophy. Powered by a 32-bit ARM CPU working at 16.7MHz, the GBA comes with 32KB of RAM, 96KB of VRAM for graphics, and 16KB of sound RAM. The RAM is built directly into the CPU for faster access. This memory can be further expanded with up to 256KB of RAM external to the CPU.

Graphically speaking, the console uses a 244*x*160 resolution, which is close to half of the resolution of a SuperNES and not very different from the resolution of an Apple][. It can perform tile-based backgrounds, including 4096 maximum sprites (256 of which can be layered on a single scanline). This huge number of sprites is especially useful for special effects such as particle systems, because the GBA supports (to an extent) alpha blending. Sprites can also be hardware scaled and rotated. Color depth is 32,768 colors and is selected from a palette of 16M.

In addition, cartridges can hold as much as 64MB of data, putting this console light years ahead of the initial Gameboy design. The result of this design is that many old-school classics such as *Mario* and *The Legend of Zelda* can easily be ported to the GBA, ensuring Nintendo a great lineup of games people already love to play. In the end, GBA's horsepower is not that different from a reduced-size SuperNES.

PREVIOUS NEXT

Phase VII: The Cellular Phenomenon

Surprisingly, competition for the GBA comes from an unexpected challenger. Which device is sold by the millions, has a low cost, and is also handheld? The cell phone, of course: There are just millions of them, and they keep getting better all the time. When looking for interesting trends, always keep an eye on those devices that have large installed bases. They have a clear potential of becoming "the next big thing." As an example, the ratio between the leading game console and the cell phone is, in most countries, around six to one: There are six cell phones for each game console.

First generation cell phones were big and heavy, and had nothing to do with handheld gaming. They were just pretty simple communications platforms, offering only limited messaging capabilities via Short Message Service (SMS) in some countries. But each new standard brought new features to the market in the typical fast evolution pattern that arises from fierce competition. With so many phone manufacturers and carriers, it is not surprising that cellular telephony and phone-based services have evolved at such speed. For some years, Nintendo could have benefited from the lack of open, worldwide standards in the phone gaming arena. European and American phones had traditionally worked with different systems, which prevented any gaming initiative from reaching the broad consumer base otherwise captured by handheld consoles.

But phones equipped with Java began to surface by the year 2000, and their hardware specs grew accordingly. Today, phones equipped with 64MB of RAM and the processing power of a 486 or Pentium CPU are not uncommon. And what can you do with such a device? You can play *Quake, Age of Empires*, and many other games. In fact, playing on a phone has a competitive advantage over the classic PC or console experience. The phone is at the core a communications device, so the path to connected, handheld games is clearly marked.

The first success story on mobile gaming platforms has to be NTT DoCoMo's I-Mode service, launched in Japan in 1999. It is a subscriber service where users pay monthly fees to access different types of mobile content, from small applications like a map service to downloadable games. Fees are small so the use is compulsive, and colorful handsets with large screens offer a relatively sophisticated delivery platform for mobile games. I-Mode was an immediate success, and by April 2002, the service had more than 32M subscribers (with more than 28,000 new users per day). The key to its success is great content, a meaningful business model where content is paid per download (as opposed to connected, per-minute charges), and very low barriers of entry for content developers. I-Mode is based on standards such as HTML and Java, so many companies jumped on the bandwagon from the beginning.

As an example of typical I-Mode content, take *Samurai Romanesque*, a medieval role-playing game (RPG) played on I-mode terminals. Using information from the Japanese weather service, the game is able to sense the current weather right where the user is standing. If it's raining, the gunpowder used by your character in the virtual world will get wet, and you won't be able to use any firearms. Cell phones have access to any online information and also know about the user's location. This is unique to the cell phone medium: No other gaming platform can offer such a rich gaming experience.

Phase VIII: Multiplayer Games

Multiplayer games have been around for longer than you might think. The first examples can be traced back to the early 1980s, with massive adoption starting with the arrival of the Internet and the World Wide Web by the mid 1990s. Multiplayer games can offer the same multimedia values of single-player games while introducing other human players into the mix. That makes them a greater challenge and, in many peoples' opinion, more rewarding to play. Several formulas have evolved through the years, from the squad-based gameplay that was already present in the Essex Multi-User Dungeon (MUD)—one of the first documented massively multiplayer games—to human opponents that can be traced back to games like EA's *Multiple User Labor Element* (*M.U.L.E.)* from 1983.

The reason why multiplayer games have been so successful is that playing with or against an artificial intelligence cannot be compared to a human opponent. Additionally, the social component of a single-player game is virtually nonexistent.

One of the first viable examples of a multiplayer game was the Essex MUD, which was developed by Richard Bartle and Roy Trubshaw of Essex University in 1979. The original game consisted of about 20 rooms described in prose. Up to 250 players could coexist in this game world, so they could "see" each other in the room. Descriptive texts would state facts like "Peter is here," for example. Players connected to the MUD server using the EPSS network, which connected six British universities; and from 1980, players connected to the MUD server using the ARPA network from the United States. Compared with a modern game, *Planetside* (by Sony) supports more than 3,500 players per server.

The Essex MUD was a great inspiration for many developers. In fact, a very active MUD scene existed during the 1980s and part of the 1990s. MUDs faded away as graphical multiplayer games such as *Everquest* appeared. But for more than 10 years, they ruled the multiplayer arena.

However, MUDs were not the only way to go. Simple competitive games with human opponents created bigger challenges for the player, thus becoming a viable playing experience. A good and early example is M.U.L.E. In M.U.L.E., up to four players competed to conquer a new planet in a way not very distant from today's real-time strategy games. But the success of the game was limited, mainly because network infrastructure circa 1983 was not very well suited for real-time play.

The turning point came around 1993, when the Internet and more specifically the World Wide Web phenomenon exploded. In less than six months, the World Wide Web evolved from zero to infinity, taking the whole planet by storm. As a side effect, connection speeds were greatly improved, from the very early 9600bps modems to somewhere around 56kbps, and from there to ISDN, DSL, and cable. This speed increase was extremely relevant because it allowed developers to transfer all state information from one computer to another while keeping the gameplay fluid. *Age of Empires*, for example, could be played on a 56kbps modem, and that included dozens of units fighting against each other.

Today, multiplayer games have stabilized around two "orbits," which are direct descendants of MUDs and games like M.U.L.E. On one hand, MUDs gave way to graphical RPGs like *Ultima Online* and from there to *Everquest, Asheron's Call*, and many others. These games are all about immersion and socially interacting with a rich game world. Thousands of people actually live their lives inside each one of these games in a 24-hour alternate reality. In fact, these games are probably the most addictive of them all, with extreme cases counting for more than 1,600 hours of gameplay per year (that's more than four hours a day, even on Sundays). As a very intriguing side effect of the *Ultima/Everquest* phenomenon, real virtual economies have been created. Some fans have sold their properties or characters on auction web sites, often for respectable sums of money. This type of behavior should definitely make us think about the influence of the gaming scene on people's lives.

On the other end of the spectrum, reduced-scale multiplayer games have been successful as well. Here, a distinction has to be made between *competitive* and *cooperative* multiplayer titles. The difference is in the role of other players: Will they be enemies you must defeat, or will they be your teammates on a global mission? A title that clearly shows these two categories is the incredibly popular *Counterstrike*, which is actually a modification to the original *Half-Life* game. In *Counterstrike*, two teams fight each other, so there's competitive gameplay on opposite sides, and cooperative gameplay between members of the same side.

The future looks bright for multiplayer games. Network infrastructure keeps growing, and today most users already have broadband support that allows better interaction. Several middleware companies have emerged in recent years, enabling bigger and better games. Additionally, gameplay innovations are constantly introduced. Some people predict that multiplayer games will eventually engulf all other gaming experiences, so single-player games will just disappear. Although that's probably an overstatement, it clearly shows the level of

expectations the industry has put in multiplayer gaming. But remember solo and team sports peacefully coexist in the real world, so there's really no need for single-player games to disappear in favor of multiplayer experiences.

PREVIOUS NEXT

In Closing

The games industry is undergoing a huge transformation. The advent of the PlayStation 2 and Microsoft's Xbox has opened new areas that were unavailable on less powerful devices. A truly cinematic gaming experience seems only years, not decades, away. The evolution is so fast that future consoles will no longer be limited to playing games but will incorporate other features as well. They will offer video-on-demand, online capabilities, and e-commerce integrated into a simple, cost-effective, and easy-to-use solution. The game console will evolve into a completely integrated, general-purpose home entertainment device.

On the other hand, the handheld business is still exploding. There are three contenders (cell phones, palm computers, and handheld consoles), and it now seems clear that they will have to merge and integrate to offer a unified, personal electronics device. After all, consumers don't want to carry three devices in their pockets when their functionality is really not that different. Will one of the three dominate and win the battle, or will new devices surface and take the market by storm? Although the latter seems unlikely, the unexpected awaits where no one is watching.

The ultimate decision will be in the hands of the consumers as to which gaming platforms will succeed. In the last 30 years, we have seen many products flop simply because people didn't embrace them. Other products that did not look very promising became serious hits when consumers accepted and supported them.

To get a glimpse of what the future might bring for the game industry, we need to extract valuable lessons from our past, understanding what made some gaming platforms successful in the first place. The ability to convey a rich, engaging game world is fundamental. Thus, presentation values and the multimedia experience will likely continue to grow for years to come. Excellent gameplay is even more important than sharp graphics and sound. Many games (such as *Tetris*) have triumphed with little presentation values but good gameplay. But no game has ever become a big success if it simply wasn't fun or engaging.

In addition, the ability to break boundaries between the real and the virtual, and between users and developers, is becoming increasingly relevant. From user-created content to games that stream data from the real world, integrating the user more tightly into the game experience is certainly a trend to keep an eye on.

But wait, did we need 20 years to learn this? Wasn't this what the movie *Tron* was all about?