In late 1981, Dr. Alan Kay recruited me into Atari Research and challenged me to dream. Most people take a lazy approach to dreaming. They put their feet up on the desk and engage in idle mental forays for half an hour, and they call it dreaming. To me, dreaming is a much more deliberate and difficult process. Dreaming is hard work!

I see dreaming as occupying a middle ground between fantasy and planning. A fantasy is the free indulgence of human desire, unconstrained by the limitations of reality. A plan reduces human desire to a pedestrian statement of objective, and then posits a sequence of steps that will attain the objective. Fantasy springs from human desire, where planning hews to reality. Dreams live in the gray zone where fantasy merges with planning. Where a fantasy spreads its wings and soars off into space, and a plan plods along earthbound, step by bitter step, a dream takes a running start and leaps as high as it can to gain a handhold by which to hoist itself up.

Fantasies and dreams both create alternate universes. A fantasy's alternate universe is unconcerned with the real universe. It is a desirable universe, but not an attainable one. A dream creates an alternate universe that is both desirable and attainable. That requires the dreamer to sketch out all the ramifications of his dream, to create a complete and consistent universe. A fantasy universe can be fragmented or inconsistent, because human desire is often fragmented and inconsistent, but a dream universe cannot be so self-indulgent. A good dream universe is a
complete image, not a partial sketch. Every detail of the ideal is clearly specified, every consequence worked out. Only when we know precisely where we’re going can we begin planning how to get there.

So, under Alan Kay’s prodding, I set to work on my dream. It took me a year and a half to give it form. I wrote the first edition of this book, The Art of Computer Game Design, as part of my process of forging my dream. By 1983, I had my dream:

I dreamed of the day when computer games would be a viable medium of artistic expression—an art form. I dreamed of computer games expressing the full breadth of human experience and emotion. I dreamed of computer games that were tragedies, games about duty and honor, self-sacrifice and patriotism. I dreamed of satirical games and political games; games about the passionate love between a boy and girl, and the serene and mature love of a husband and wife of decades; games about a boy becoming a man, and a man realizing that he is no longer young. I dreamed of games about a man facing truth on a dusty main street at high noon, and a boy and his dog, and a prostitute with a heart of gold.

What elevated these thoughts from fantasy to dream was the identification of the central problem: the concentration of computer games on things rather than people. I wrote an essay entitled “People, Not Things!” in which I bemoaned the fact that all computer games were about things, not people. “You chase things,” I noted, “and things chase you. You acquire things, expend things, utilize things, shoot at things, but it’s always things, things, THINGS! There are never any real people in our games!” My goal was clear: I must perforce design a game about people.

But how? Given the wimpy hardware available to me in 1982, I had difficulty imagining a game about people. So I got down to basics. If the game is about people, it has to be about their behavior. More specifically, it has to be about their behavior toward each other. After much random flailing about, I hit upon the earliest expression of Crawford’s First Rule of Software Design (Lesson 43).
The answers to this question tell you everything important about the design. The user's choices and actions lie at the essence of any software design, not just games. You can describe a word processor in a thousand ways, but the best description provides a list of what the user does: type words, edit them, and print out the results. The user of a spreadsheet enters numbers and formulae, and then fiddles around with those numbers and formulae to explore various scenarios.

My little rule, simple as it might be, offers vast utility. It can sniff out a design error in a flash. Pick out any piece of badly designed software and ask the magic question, and you'll get an answer that reveals the fundamental flaw. The old checkbook balancing programs of the early 1980s were utter failures, but the personal financial management programs of the 1990s were big successes. Why? Because the checkbook balancing programs allowed the user to do nothing more than enter the amounts on bank checks and then print out the results—that's not much better than doing it by hand with a calculator. The personal finance programs, in contrast, allowed the user to enter all financial data, categorize it, prepare budgets and compare them with actual spending, and use the data in tax returns. These are useful things to do, which is why many people like to use such programs.

And so I asked myself, “What should the player of a people-style game do?” The obvious answer is that the player should be able to do social actions with other people. So what do people do to and with each other? The answer to this question is as vast as the range of human culture, so I needed to find some clean, simple subset of
human behavior, some set of actions that form a closed set of actions. After much trial and error, I hit upon the answer: gossip. People love to gossip about each other.

Of course, gossip is a complex behavior requiring the use of language; there was no way I could get the full range of gossip behavior into a computer. But I could get a tiny subset of it in place: the declaration of affinity. A great deal of gossip boils down to statements of affinity: "I like Jane," "I hate Tom," and so forth. Such statements have a pronounced effect on the listener, serving to alter the listener's own affinities. After all, if your best friend tells you that he likes Mary, then you are likely to be more favorably inclined toward Mary.

Even at this simple level, gossip behavior offers all sorts of interesting possibilities. For example, it works in reverse: If your best friend tells you that he hates Mary, then you are likely to be less favorably inclined toward Mary. And if someone you loathe is Tom's best friend, you are likely to be less favorably disposed toward Tom. This behavior also feeds back to the speaker: If your best friend tells you that he likes Tom, then your estimate of his friendship will diminish.

This can all be summarized by a simple statement: "People like people who like people they like, and vice versa versa versa." (I put in the extra "versas" to cover each of the different "likess" in the statement.) The idea can also be presented in mathematical form:

\[
\begin{align*}
CA \{\text{Listener, Speaker}\} &= DA\{\text{Speaker, Object}\} \times A\{\text{Listener, Object}\} / K1 \\
CA \{\text{Listener, Object}\} &= DA\{\text{Speaker, Object}\} \times A\{\text{Listener, Speaker}\} / K2
\end{align*}
\]

where

- CA is the change in affection.
- Listener is the person hearing the gossip.
- Speaker is the person speaking the gossip.
- Object is the person being gossiped about.
• DA is the declared affection.
• A is the absolute affection.
• K1 is some constant greater than one, say, 10.
• K2 is some different constant greater than one.

Now, these are simple differential equations, but they apply individually to each member of an entire group of people, and so the overall behavior of this system of equations can be complicated.

All the hard work lay in getting up to the point where I realized that gossip restricted to declarations of affection could provide the basis for a game. From that point forward, it was easy. I wrote a quick version in BASIC in one day; it lacked graphics, but the game was playable. Much tuning would be required, but the gameplay seemed solid. It was a ridiculously simple game, but then, so was Pong. This game was to people-games as Pong was to videogames.

AI

I had an excellent play-mechanic in hand, but now I needed some AI to control the computer people. How were they to respond to the gossip they heard?

At this point, it is worthwhile to point out the obvious: The solutions I describe never came quickly. I can’t remember the many trials and errors I pursued, but there were quite a few. Weeks rolled by while I struggled with these problems. The gap between problem and solution in this book is often just a line of text, but in the real world it was much greater, and the eventual solution was nowhere near as clear as I present it here; most of the time I bumbled towards it in a reverse Drunkard’s Walk (that’s an old mathematical problem), eventually getting myself to the street light.

The solution that I eventually chanced upon relied, once again, on an analogy with the real world. In this case, I imagined the social system in the game to be rather like a set of springs tied together. Each person
With enough imagination, you can find models to solve any problem.

**GOSSIP**

represented one node; seven springs, one for every other person, were tied to that person. Each spring had a "desired length" given by the affection that the one person held for the other. Now, if you tried to build such a system, you'd get a tangled mess, but in the pure mountain air of software, tangles don't exist. The idea is that each spring pushes two people apart or pulls them together; if you let the system "relax," the springs will all push and tug at each other until each person is in his optimal social position relative to the others. Good friends are close; hated enemies are far away.

This imagined environment could easily be simulated on the computer, and it suggested the solution: Each person should attempt to reduce the spring tensions around him. If one spring pulls him one way and another spring pulls him the other, he'll seek to release the tension by moving toward or away from somebody else. The calculation for all this is not difficult; it's a simple physics problem. So, with a snap of my fingers, the AI was ready to go.

**Implementation Woes**

I had just taken on a summer intern for my group; my intention had been to let him poke around with the other members of the group, helping out as possible. He had great skills as a graphics programmer and this little design needed nothing more than some graphics tacked onto it. Well, yes, it would have to be converted from BASIC to assembly language, but it was such a short program that I anticipated no problems. So I assigned the Gossip project to the summer intern.
Things went swimmingly at first. He had the game screen up and running in no time. There were eight people in two rows across the screen, each one shown with head and upper torso only (see Figure 19.1). They looked rather like the panelists on a game show. The person on the upper left was the player. Beside each player was a little telephone.

19.1 Gossip main display.

I congratulated the student on his rapid progress, and suggested that he get to work on the algorithms, but he wanted to finish up the graphics first. So he went ahead and added the remaining graphics, as well as some delightful sound effects. The gameplay was simple: The player would use a cursor to designate a person to be called. Pressing the button would select that person, whose telephone would ring with an appropriate jangling sound and the handset jiggling on the telephone base. The person called would pick up the handset with a simple three-step animation, hold it to his or her ear, and say something like
“Air-oh?”, which was the best that could be done with the primitive sound capabilities of the day. Then the player would use the cursor to designate another player, the one whom he wished to gossip about, and press the trigger button again. This would highlight that person; at the same time, the player’s face would undergo a simple seven-step animation.

There were five of these animations. In one, the player had a big smile on his face and nodded his head up and down vigorously. The next, selectable by the joystick, had the player’s face nodding up and down a bit less vigorously and with a merely pleasant smile. The third showed the player’s face motionless with a blank look. The fourth showed the player frowning slightly, and nodding his head from side to side. The last showed the player looking quite angry, yanking his head from side to side most emphatically. The player would select the facial expression that reflected his feelings toward that individual, and then press the trigger button. One piece of gossip had been completed.

It looked and sounded really great; I was quite pleased. Just three weeks had passed and all that remained to be done was to implement the behavioral and AI algorithms, which, as I wrote earlier, were a piece of cake.

But my summer intern just couldn’t hack it. Try as he may, he simply could not get a few simple equations to work in assembly language. Being a lousy people manager, I failed to intervene to walk him through the code problems; I simply couldn’t believe that anybody could be so brilliant with graphics and so utterly incompetent with simple arithmetic. Pig-headedly, I urged him on with affirmations that the problem was trivial.

It’s a measure of just how bad a manager I am that this situation persisted for three months before I finally put my summer intern out of his misery. I had figured that he’d go away at the end of the summer
anyway, so I swept the problem under the rug. But he never did go back to school, and so I had to bite the bullet and let him go. Fortunately, he found a position elsewhere in Atari.

I turned the problem over to another subordinate who fixed it and got the program running in about a week. With a little tuning, we had the whole thing ready to ship within a month of the departure of the summer intern.

**Conclusions**

**Gossip** had the bad luck to be published in 1983, just as Atari was beginning its death spiral. With all the chaos of the layoffs, jobs like publishing games proceeded at a snail’s pace and without much in the way of verve and élan. There is some question as to whether it ever appeared in the sales catalog; I myself don’t know. I don’t even know its sales figures; it seemed that, every time I called someone to ask, that person had been laid off. I’m sure it sold poorly; few people recall the game.

But **Gossip** deserves a place in history: it was the first computer game about people instead of things. It opened up all sorts of interesting possibilities for further development. Games in which the gossip mechanics are extended to cover additional modes of expression, Games with larger groups. Games with gossip augmented by other dimensions of behavior. Sadly, nobody else ever followed up on these ideas; even today, the most people-oriented game on the market, **The Sims**, boasts a level of interpersonal interaction no higher than that offered by **Gossip** in 1983.