The Game Design Sequence

Chapter Five

Game design is primarily an artistic process, but it is also a technical process. The game designer pursues grand artistic goals even as she grinds through mountains of code. During the process of developing the game, she inhabits two very different worlds, the artistic world and the technical world. How does one manage the integration of such dissimilar worlds? In short, how does one go about the process of designing a computer game? In previous chapters I have touched on some of the questions related to this process; I have also laid down a few precepts. In this chapter I will suggest a procedure by which a computer game could be designed and programmed.

The procedure I will describe is based on my own experiences with game design, and reflects many of the practices that I use in designing a game. However, I have never used this procedure in a step-by-step fashion, nor do I recommend that any person follow this procedure exactly. In the first place, game design is far too complex an activity to be reducible to a formal procedure. Furthermore, the game designer's personality should dictate the working habits she uses. Even more important, the whole concept of formal reliance on procedures is inimical to the creative imperative of game design. Finally, my experience in game design is primarily with personal computers, so my suggestions are not completely applicable to arcade game designers or home video game designers. I therefore present this procedure not as a normative formula but as a set of suggested habits that the prospective game designer might wish to assimilate into her existing work pattern. With these important qualifications in mind, let us proceed. Top

CHOOSE A GOAL AND A TOPIC

This vitally important step seems obvious, yet is ignored time and time again by game designers who set out with no clear intent. In my conversations with game designers, I have many times discerned an indifference to the need for clear design goals. Game designers will admit under close examination that they sought to produce a "fun" game, or an "exciting" game, but that is more often than not the extent of their thinking on goals.

A game must have a clearly defined goal. This goal must be expressed in terms of the effect that it will have on the player. It is not enough to declare that a game will be enjoyable, fun, exciting, or good; the goal must establish the fantasies that the game will support and the types of emotions it will engender in its audience. Since many games are in some way educational, the goal should in such cases establish

what the player will learn. It is entirely appropriate for the game designer to ask how the game will edify its audience.

The importance of a goal does not become obvious until later in the game design cycle. The crucial problems in game development with microcomputers are always problems of trade-offs. Everything that the game designer wants to do with her game costs memory, and memory is always in short supply with microcomputers. Thus, the designer must make trade-offs. Some game features can be included, and some must be rejected. At two o'clock in the morning, when you must face the awful decision of rejecting one of two highly desirable features, the only criterion you will have for making this painful choice will be the goal you have established for the game. If your goals are clear, your decision will be painful but obvious; if your goals are murky, you may well make the wrong choice, and whatever you choose, you will never know if your decision was correct.

How do you select a proper goal? There is no objective answer to this question; the selection of a goal is the most undeniably subjective process in the art of computer game design. This is your opportunity to express yourself; choose a goal in which you believe, a goal that expresses your sense of aesthetic, your world view. Honesty is an essential in this enterprise; if you select a goal to satisfy your audience but not your own taste, you will surely produce an anemic game. It matters not what your goal is, so long as it is congruent with your own interests, beliefs, and passions. If you are true to yourself in selecting your goal, your game can be executed with an intensity that others will find compelling, whatever the nature of the game. If you are false to yourself, your game will necessarily be second-hand, me-too.

There are situations in which it is not quite possible to attain the purity of this artistic ideal. For example, I would not claim that only immature, childish people should design games for children. Nor would I suggest that good shoot-'em-up games can only be done by shoot-'em-up personalities. The realities of the marketplace demand that such games be written, and it is better that they be written by mature professionals than by simpering fools. Such emotionally indirect games, however, will never have the psychological impact, the artistic power, of games coming straight from the heart.

Once you have settled on your goal, you must select a topic. The topic is the means of expressing the goal, the environment in which the game will be played. It is the concrete collection of conditions and events through which the abstract goal will be communicated. For example, the goal of STAR RAIDERS apparently concerns the violent resolution of anger through skillful planning and dexterity. The topic is combat in space. The goal of EASTERN FRONT 1941 concerns the nature of modern war, and especially the difference between firepower and effectiveness. The topic is the war between Russia and Germany.

Most game designers start off by selecting their topic, with their goals subordinated to their topic. Indeed, they commonly describe a game under development by its topic rather than its goal. When I tell other designers that I am working on a game about leadership, I am met with quizzical expressions. Is it a space game, or a wargame, or a dungeon game, they wonder; they seem satisfied when I tell them it's a game about King Arthur. It is a serious mistake to subordinate the goal to the topic. Although your initial flash of inspiration may focus more on the topic than the goal, you must have the determination to take control of the design and impose your own goals onto the topic rather than allowing yourself to be swept away by the momentum of the topic.

Selecting a good topic can be time-consuming, for each potential topic must be carefully examined for its ability to successfully realize the goals of the game. Many topics carry with them some excess emotional baggage that may interfere with the goals of the game. For example, my most recent game design effort uses the Arthurian legends as its topic. My goal in the game is to examine the nature of leadership. I found the Arthurian legends to be a compelling vehicle for this goal. Unfortunately these

legends contain a strong component of male braggadocio, the vanquishing of opponents by brute force. This theme directly contradicts some of the points I want to make with the game, thus weakening the utility of this topic for my ends. I find the legends so powerful and so malleable that I am willing to accept and work around this potential pitfall. Top

RESEARCH AND PREPARATION

With a goal and topic firmly in mind, the next step is to immerse yourself in the topic. Read everything you can on the topic. Study all previous efforts related to either your goal or your topic. What aspects of these earlier efforts appeal to you? What aspects disappoint or anger you? Make sure that you understand the mechanics of the environment your game will attempt to represent. Your game must give the authentic feel, the texture of the real world, and this can only be achieved if you firmly understand the environment of the game. While researching EXCALIBUR, I studied the history of Britain during the period AD 400-700. I found little in the history books that was harmonious with my goal of depicting the nature of leadership. But in the Arthurian legends I found recurring themes more closely related to my goal. You may well find yourself adjusting your goals as you perform this research function; such erratic decision-making is an embarrassing admission of poorly-defined goals, but reflects an honest willingness to adapt to the exigencies of the topic-environment. It is a departure from the ideal in which I have sinfully indulged myself many times.

During this phase it is critical that you commit little to paper and above all, WRITE NO CODE! Take long walks as you contemplate your game. Cogitate. Meditate. Let the goal, the topic, and the facts gleaned from your research simmer together in the innards of your mind. Weave them together into a whole. Take your time with this phase; impatience now will lead to mistakes that will kill the game. I give myself at least three weeks to develop a game idea in this stage before proceeding to the next step. With EXCALIBUR I expended several months on this stage. During this time I kept my fidgeting hands busy by writing an opening graphic display that had little relevance to the final game.

You will generate during this phase a great variety of specific implementation ideas for your game. They will not all fit together neatly---like any hodgepodge, they will require much sorting and rearranging before they can be used. You should not wed yourself to any of them. A large collection of candidates for implementation is a useful resource during the design phase. A laundry list of implementation ideas that must be included is a liability. Indulge yourself in creating implementation ideas, but be prepared to winnow them ruthlessly during design.

For example, I recently designed a corporate politics game in association with another person. During the research and preparation phase, we came up with a long list of clever ideas that we wanted to into the game. We had agreed that the game would have a feminist point of view without being preachy. We wanted to have a demanding boss, tough projects, deadlines, brownie points, one male chauvinist pig, neutral males, neutral females, family and home obligations, mentors, and the competition for the big promotion. We managed to include almost all of these ideas in the final design. We were not able to integrate the family elements into the game. Every design we created failed to do justice to our desires. In the end, we had to discard this desirable element. Top

DESIGN PHASE

You now have a clear idea of the game's ideals but you know nothing of its form. You are now ready to begin the concrete design phase. Your primary goal in the design phase is to create the outlines of three interdependent structures: the I/O structure, the game structure, and the program structure. The I/O structure is the system that communicates information between the computer and the player. The game structure is the internal architecture of causal relationships that define the obstacles the player must

overcome in the course of the game. The program structure is the organization of mainline code, subroutines, interrupts, and data that make up the entire program. All three structures must be created simultaneously, for they must work in concert. Decisions primarily relating to one structure must be checked for their impacts on the other structures. Top

I/O Structure

I prefer to start with the I/O structure, for it is the most constraining of the three. I/O is the language of communication between the computer and the player; like any human language, it is the funnel through which we must squeeze the avalanche of thoughts, ideas, and feelings that we seek to share with our fellow human beings. I/O will dictate what can and cannot be done with the gains.

I/O is composed of input and output. Unlike human languages, the two are not symmetric. The computer has two means of output to the human: graphics on the screen and sound. In the future, we may see more exotic devices for output for games, but for the moment these are the two most common. Graphics are the most important of the two, perhaps because we humans are more oriented towards vision than hearing. For this reason, many game designers devote a large portion of their energy towards the design of quality displays. Indeed, some designers go so far as to design the display first and let the game develop from the display, as extreme an example of goal-less design as ever there could be.

Don't make the common mistake of creating cute graphics solely to show off your ability to create cute graphics. Graphics are there for a reason: to communicate. Use graphics to communicate to the user forcefully and with feeling, and for no other reason. Plan functional, meaningful graphics that convey the critical game information while supporting the fantasy of the game. Don't use graphics tricks as a crutch for a bad game design. If the game is dull and boring, no amount of graphics gift-wrapping is going to fix it. The worst examples of this mistake are the games that alternate boring game segments with cute but meaningless graphics displays. Use of sound should follow the same rules: use it to tell the player what's going on in the game. The only place where striking but uninformative graphics and sound can be useful is at the beginning of the game, and then only if they help to establish the mood or tone of the game.

Storyboards are a graphics design tool that tempt many game designers, for they are a well-developed technology from the film industry. They are not appropriate to games, because storyboards are an intrinsically sequential technology. Games are not sequential, they are branching tree structures. The game designer who uses an intrinsically sequential tool risks having her designs made subtly sequential. The tool shapes the mind of its user; the saw suggests that we cut wood, and the freeway suggests that we drive wherever it takes us, not where we choose to go. In like manner does a storyboard impress its sequentiality upon our games.

Devote special care to the input structure of the game. The input structure is the player's tactile contact with the game; people attach deep significance to touch, so touch must be a rewarding experience for them. Have you ever noticed the tremendous importance programmers attach to the feel of a keyboard? Remember that players will do the same thing with your game. A case in point is provided by the games JAWBREAKER and MOUSKATTACK (trademarks of On-Line Systems). In both games the joystick entry routine admits an unfortunate ambiguity when a diagonal move is entered. This gives the player the impression that the joystick is unresponsive. I have seen players slam down the joystick in frustration and swear that they would never play the damn thing again. Remember this well as you plan your input structure: will your input structure frustrate and anger your players?

The input structure lies at the heart of a fundamental dilemma all game designers must face. An excellent game allows the player to interact heavily with his opponent, to invest a great deal of his personality into the game. This requires that the game offer the player a large number of meaningful

options, enough options that the player can express the nuances of his personality through the choices he makes. Yet, decisions must be inputted, and a large number of options seem to require an extensive and complicated input structure, which could well be intimidating to the player. Our dilemma, then, is that an excellent game seems to require a hulking input structure.

The dilemma is resolved through the designer's creativity in designing a clean input structure that allows many options. This does not come easily. Many schemes must be considered and rejected before a satisfactory solution is found. Yet, such a solution is often possible. In designing SCRAM, a nuclear power plant game, I faced the following problem: how can a player control an entire nuclear power plant with only a joystick? At first glance, the task seems hopeless. Nevertheless, the solution I eventually discovered works very well. The player moves a cursor through the plant display. With the cursor adjacent to a piece of controllable equipment, the player presses the joystick button and pushes the stick up to turn on or increase power, and down to turn off or decrease power. The system is simple and easily understood once the player has seen it.

There is a general solution, at the theoretical level, to the dilemma of option richness versus input cleanliness; I call this solution "the webwork". To design a webwork game, we start with a small number of pieces. We then define a relationship that applies to all pairs of pieces. The set of relationships between pieces constitutes a webwork. The webwork can easily become quite complex, yet few pieces are required to create the webwork. In general, the number of pairwise relationships is equal to N*(N-1), where N is the number of pieces. Thus, four pieces can generate 12 pairings, 8 pieces can generate 56 pairings, and 16 pieces can generate 240 pairings. With fewer pieces to manipulate the player faces fewer I/O problems without sacrificing a rich set of relationships in the game.

Backgammon illustrates the simplicity and power of webwork games. Backgammon has only 30 pieces and 26 positions for them to occupy. The relationships between pieces are fairly simple and are expressed through the ability to move and bump. Yet, on any given move, each piece has an offensive, defensive, blocking, or blocked relationship with most of the other pieces on the board. This is partly because almost every other board position in front of the piece can be reached, given the right die roll. It is no accident that the length of the playing area (24 steps) is exactly equal to the maximum die roll. It had to be that way to squeeze all of the pieces into range of each other, thereby maximizing the number of significant pairwise relationships.

Most webwork games rely on spatially expressed webworks; these are easy to depict and easy for the player to visualize. Few games have non-spatial webworks; my own GOSSIP is one such game. Curiously, GOSSIP uses a spatial webwork for its internal computations even though the game webwork is non-spatial. This may imply that game webworks are intrinsically spatial; it may equally well imply that I cannot shake my mind-set free from spatial webworks.

The choice of input device is an important design decision. I maintain that a good game designer should eschew the use of the keyboard for input and restrict herself to a single simple device, such as a joystick, paddle, or mouse. The value of these devices does not arise from any direct superiority over the keyboard, but rather in the discipline they impose on the designer. Simple input devices go hand-in-hand with simple input structures. Complex input devices encourage complex input structures.

The I/O structure is the most important of the three structures in a computer game, for it is the face of the game that the player sees. It is the vehicle of interaction for the game. It is also the most difficult of the three structures to design, demanding both human sensitivity and complete technical mastery of the computer. Give it the care it deserves. Top

Game Structure

The central problem in designing the game structure is figuring out how to distill the fantasy of the goal

and topic into a workable system. The game designer must identify some key element from the topic environment and build the game around that key element. This key element must be central to the topic, representative or symbolic of the issues addressed in the game, manipulable, and understandable. For example, in EASTERN FRONT 1941, I started with the enormous complexity of modern warfare and extracted a key element: movement. Movement dictates the dispositions of the military units. Moving into an enemy's position initiates combat with him. Moving behind him disrupts his supplies and blocks his retreat routs. Moving into a city captures it. Movement is not equitable with all aspects of war; it is, instead, the key element through which many other aspects of war are expressible. It is easily manipulable and immediately understandable.

A more difficult design challenge came from the game GOSSIP. This game addresses social relationships. The enormous complexity of the subject matter and the intricate twists and turns of human interaction together suggest that the subject is beyond treatment in a game. After much thought I was able to isolate a key element: the "statement of affinity". One way or another, many of our social interactions boil down to one of two declarations: a first-person statement of feeling ("I rather like Sandra"), and a third-person statement ("Well, Tom told me that he doesn't like Sandra one bit"). The key element encapsulates the grander array of human interactions rather well. It is easily manipulable; indeed, it is quantifiable. And it is quite understandable. The isolation of the statement of affinity as the key element of human interaction made possible the game GOSSIP.

The nature of manipulability assumes tremendous importance to the success of the game. The key element must be manipulable, but in a very specific set of ways. It must be expressively manipulable; that is, it must allow the player to express himself, to do the things that he wants or needs to do to experience the fantasy of the game. For example, in a combat game, shooting is almost always a key element. If the player's freedom to shoot is heavily restricted, the player cannot live the fantasy. At the same time, the manipulability must be concise. To use the combat game example again, if the player is required to declare the amount of gunpowder to be expended on each shot, he may well find the manipulability a hindrance to the game. The manipulability must be meaningful to the fantasies of the game. Finally, the manipulability must be focused: the options from which the player chooses while manipulating the key element must be closely related. For example, in the game GOSSIP, the key element (statement of affinity) assumes a linear sequence of values ranging from hatred through love. ENERGY CZAR violates this principle by requiring the player to choose from a large, disconnected set of options. Menu structures and use of the keyboard both arise from unfocussed key elements.

Many games employ multiple key elements. For example, most combat games include both movement and shooting. This is not necessarily bad; if both key elements are kept simple, or if one key element retains primacy, the game can be successful. However, too many key elements violating too many of these principles will rob the game of its focus.

Your main problem with creating the I/O structure is overcoming constraints; your main problem with creating the game structure is realizing possibilities. Your previous work with the I/O structure defines the limitations on the structure of the game. You can take more liberties with the internal structure because the player will not directly encounter it. For example, for the game TACTICS I developed a very complex combat algorithm that realistically calculates the effects of armor-piercing shot. The complexity of this algorithm would have confused the player had I tried to explain it. But the player does not need to understand the internal workings of the algorithm; he need only grasp its effects. I therefore did not feel constrained to design a simple-minded and intuitively obvious algorithm.

Concentrate an providing enough color to guarantee that the game will convey the authentic feel of reality. Keep your sense of proportion while adding details. It will do your game no good to provide exquisite detail and accuracy in one sphere while overlooking the most fundamental elements in another sphere.

A very common mistake many designers make is to pile too many game features onto the game structure. In so doing, they create an overly intricate game, a dirty game. As I discussed in Chapter 4, dirt is undesirable; a game is a structure that must fit together cleanly and well, not a brushpile. Dirt creates a second problem not mentioned in Chapter 4: it gums up the I/O structure of the game. For example, the long-range scan feature of STAR RAIDERS does provide some nice additional capabilities, but it adds another keystroke to be memorized by the player. That's dirty input. Fortunately this problem is overridden in STAR RAIDERS, because the fantasy puts the player at the controls of a starship, and so the player finds the intricacy of the control layout a supporting element of the fantasy rather than a hindrance. In most games, you may well be forced to give up nice elements in the game structure in order to maintain the quality of the I/O structure. On the other hand, you may be forced to go back and change the I/O structure to incorporate a game feature you are unwilling to abandon. If you do so, do not simply tack on a now command; rethink the entire I/O structure and modify it so that the new command fits well with the rest of the I/O structure.

Designing the game structure is emotionally very different from designing the I/O structure. While designing the I/O structure, the designer must thread a precarious path between the Scylla of expressive power and the Charybdis of expressive clarity, even while the storms of hardware limitations toss her design to and fro. While designing the game structure, the designer finds herself on a placid sea stretching flat to the horizon. The challenge taunting her now is "Where do you go?" Top

Program Structure

The program structure is the third object of your design attentions. This structure is the vehicle which translates the I/O structure and game structure into a real product. One of the most important elements of the program structure is the memory map. You must allocate chunks of memory for specific tasks. Without such safeguards, you may end up expending excessive quantities of memory on minor functions, and having insufficient memory remaining for important tasks. Definitions of critical variables and subroutines are also necessary. Finally, some documentation on program flow is important. Use flow charts or Warnier-Orr diagrams or whatever suits your fancy. This book is not primarily concerned with programming; if you need guidance on program development, consult any of the many excellent books on program development. Top

Evaluation of the Design

You now have three structures in hand: the I/O structure, the game structure, and the program structure. You are satisfied that all three structures will work and that they are compatible with each other. The next stop in the design phase is to evaluate the overall design for the most common design flaws that plague games. The first and most important question is: does this design satisfy my design goals? Does it do what I want it to do? Will the player really experience what I want him to experience? If you are satisfied that the design does pass this crucial test, proceed to the next test.

Examine the stability of the game structure. Remember that a game is a dynamic process. Are there any circumstances in which the game could get out of control? For example, if the game has money in it, could a situation arise in which the player finds himself the owner of ridiculously large amounts of money? In short, does the game structure guarantee reasonable upper and lower bounds on all values? If not, re-examine the game structure carefully with an eye to structural changes that will right the situation. If you have no other options, you may be obliged to put them in by brute force (e.g., "IF MONEY > 10000 THEN MONEY 10000")

Now probe the design for unanticipated shortcuts to victory. A player who can find a way to guarantee victory with little effort on his part will not derive the full benefit of your game. Insure that all unintended shortcuts are blocked so that the player must experience those processes that you want him to experience. Any blocks you place must be unobtrusive and reasonable. The player must never notice

that he is being shepherded down the primrose path. An example of obtrusive blocking comes from the game WAR IN THE EAST (trademark of Simulations Publications, Inc). This wargame deals with the Eastern Front in World War 11. The Germans blitzed deep into Russia but their advance ground to a halt before Moscow. To simulate this the designers gave the Germans an overwhelming superiority but also gave them a supply noose whose length was carefully calculated to insure that the Germans would be jerked to a dead halt just outside Moscow. The effect was correct, but the means of achieving it were too obvious, too obtrusive.

The last and most crucial decision is the decision to abort the game or proceed. It should be made now, before you commit to programming the game. Do not hesitate to abort the game now; even if you abort now you will still have I earned a great deal and can say that the effort was worthwhile. A decision to give up at a later stage will entail a real loss, so give this option careful consideration now while you can still do it without major loss. Abort if the game no longer excites you. Abort if you have doubts about its likelihood of success. Abort if you are unsure that you can successfully implement it. I have in my files nearly a hundred game ideas; of these, I have explored at length some 30 to 40. Of these, all but eight were aborted in the design stage. Top

PRE-PROGRAMMING PHASE

If the game has made it this far, you are now ready to commit your ideas to paper. Until now your documentation has been sketchy, more along the lines of notes and doodles than documents. Now you are ready to prepare your complete game documentation. First, commit all of your design results from the previous phase to paper. Define the I/O structure and the internal game structure. The tone of this documentation should emphasize the player's experience rather than technical considerations. Compare this first set of documents with your preliminary program structure notes; adjust the program structure documents if necessary. Top

PROGRAMMING PHASE

This is the easiest of all the phases. Programming itself is straightforward and tedious work, requiring attention to detail more than anything else. Seldom has a game failed solely because the programmer lacked the requisite programming skills. Games have failed to live up to their potential because the programmer did not expend enough effort, or rushed the job, or didn't bother to write in assembly language, but in few cases has talent or lack of it been the crucial factor in the programming of a game; rather, effort or lack of it is most often the responsible factor. If you place all of your self-respect eggs in the programming basket, I suggest that you get out of game design and work in systems programming. Otherwise, write the code and debug it. Top

PLAYTESTING PHASE

Ideally, playtesting is a process that yields information used to polish and refine the game design. In practice, playtesting often reveals fundamental design and programming problems that require major efforts to correct. Thus, playtesting is often interwoven with a certain amount of program debugging.

Sometimes playtesting reveals that the game is too seriously flawed to save. A nonfatal, correctable flaw is usually a matter of insufficiency or excess: not enough color, too many pieces, not enough action, too much computation required of the player. A fatal flaw arises from a fundamental conflict between two important elements of the game whose incompatibility was not foreseen. You must have the courage to trash a game with such a fatal flaw. Patching after the game is programmed can only achieve limited gains; if the game is badly deformed, abortion is preferable to surgery.

If playtesting reveals serious but not fatal problems, you must very carefully weigh your options. Do not succumb to the temptation to fall back on a quick and dirty patch job. Many times the problem that is discovered in playtesting is really only a symptom of a more fundamental design flaw. Be analytical;

determine the essence of the problem. Once you have determined the true nature of the problem, take plenty of time to devise a variety of solutions. Don't rush this process; sometimes the ideal solution comes from an unexpected angle. Choose a solution for its promise of furthering the faithfulness of the game to your goals. Do not opt for the easiest solution, but the solution that best meets your goals.

For example, while designing EASTERN FRONT 1941, I ran into a severe problem with unit counts: there were far too many units for the player to control conveniently. After wasting much time trying to devise ways to shrink the map or directly reduce the number of units, I eventually stumbled upon zones of control, a standard wargaming technique that extends the effective size of a unit. The inclusion of zones of control in the game not only solved the unit count problem; it also made the logistics rules more significant and gave the game a richer set of strategies. I set out with the narrow goal of reducing the unit count, but I found an improvement with much broader implications.

If your initial design was well-developed (or you are just plain lucky) the game will not face such crises; instead, the problems you will face will be problems of polish. All of the little things that make a game go will be out of tune, and the game will move like a drunken dinosaur instead of the lithe leopard you had envisioned. Tuning the game will take many weeks of work. For the short term you can scrimp on the tuning while you are working on other problems, for tuning the game requires delicate adjustments of all the game factors; any other changes will only throw off the tune. Therefore, defer final tuning work until the very end of the polishing stage.

There are actually two forms of playtesting. The first is your own playtesting done in the final stages of debugging. The second form comes later when you turn over the game to other playtesters. The salient-difference between the two lies in the nature of the bugs exposed. Your own playtesting should reveal and eliminate all program bugs (arising from flaws in the program structure) and many of the game bugs (arising from flaws in the game structure). The game you give to the playtesters should be free of program bugs; they should discover only bugs in the game structure. There is no point in showing an incomplete game to playtesters, and indeed there is a danger in contaminating their objectivity by showing them a version of the game too early. But the time will come when you feel that the game is very close to completion, and your own stock of ideas for improvements is dwindling. This is the time to show the game to a few select playtesters.

Playtesters must be selected and used with great care. You cannot simply grab a few friends and ask them what they think of the game. You need playtesters who possess a deep familiarity with games, playtesters who can analyze and criticize your game with some basis of experience. Ideally the playtesters would themselves be game designers, for they would then share your appreciation for the trade-offs essential to good game design. You should also know the player well, both his personality and his game taste. You should never use more than five or six playtesters. A surplus of playtesters only insures that you will not be able to assess carefully the reaction of each playtester.

A variety of other systems have been used for playtesting. Most rely on gathering large groups of "real people" and assessing their reactions to the game. I have little respect for such systems. Although they are scientific, objective, and democratic, they seldom yield useful design information, for consumers make lousy critics. The suggestions they make are inane and impractical; they don't know enough about computers or games to make practical suggestions. Such methods may well work with detergent and shaving cream, but I very much doubt that any great movie, book, or song was created through market research of this kind. I will concede that such methods can prove to be a useful way to guide the mass production of cheap games by designers of limited talents; this book is not directed to persons of such a mentality. The playtesters will need a preliminary manual for the game. It need not be a finished product any more than the game itself---just enough orientation information to get the playtester going with the game. Make sure that there is enough in the manual that the playtester doesn't waste time critiquing problems of the game that will be solved by the manual. Do not sit down with the playtester

in advance and coach him through the game; you will only contaminate his objectivity. The playtester's first reaction to the game is your best feedback on the success of the manual . Let the playtester experiment with the game for perhaps a week before you meet with him. Do not ask the playtester to keep lengthy written records of play performance; he won't do it. Instead, include in the manual a few suggestions about potential problems that worry you. The most for which you should ask in writing is a simple record of game options selected and subsequent scores.

Schedule along interview with the playtester after he has had enough time to digest the game. Come to the interview prepared with a set of standard questions that you ask all playtesters. Do not lead the playtester's answers and don't solicit praise. Your job is to find flaws; accolades come later. While it is more scientific to use a third person to conduct the interview (thereby assuring more honest answers), this imposes a middleman between you and your playtesters. I prefer to get the information directly from the playtester. I also prefer to take a very negative tack during the interview, encouraging the playtester to criticize the game along with me and to suggest means of improving it.

Playtesters' criticisms are difficult to evaluate. Most criticisms must be rejected for a variety of reasons. Some are incompatible with your goals; some are not achievable in the-memory space remaining. Some are reasonable, but would require major software surgery incommensurate with the gains offered. Do not hesitate to reject 90% of the suggestions made. The remaining 10% are right; waste no time implementing them. How do you tell the good 10%? This is the stuff of wisdom; I certainly don't know.

The final stage of the design cycle is devoted to polishing the game. The polishing stage is actually concurrent with the later stages of playtesting and may involve several iterations with the playtesters. This stage is critical; the designer has been working on the game for a long time by now and the luster of the new design has worn off. It is now only a big job that should have been finished months ago. The playtesters love it, the publisher loves it and wants it right now, and the designer is sick of it. The urge to dump the damn thing is overpowering. Resist this urge; press on relentlessly and polish, polish, polish. Keep testing the game, fine-tuning it, and adding tiny embellishments to it. Once it's out the door, it's gone forever. Every single game I have done has followed the same pattern: I polished the game until I was sick of it and never wanted to see it again. When at last I sent the game out, I rejoiced; I was free of that dog at last. Within a month I was regretting my impatience and wishing I could have a chance to clean up that one embarrassing bug that I had never noticed. Within three months my regret had turned into shame as I discovered or was told of many more bugs. I have programs out there whose patrimony I hope never becomes widely known.

One of the last tasks you must perform before releasing the game is the preparation of a game manual. Manuals are frequently given short shrift by just about everybody associated with computer games. This is a serious mistake, for the manual is a vital element in the overall game package. A computer has many limitations; some can be overcome with a good manual. Much of the static information associated with a game can be presented in a manual. The manual is also an excellent place to add fantasy support elements such as pictures and background stories. Finally, a well-written manual will clear up many of the misunderstandings that often arise during a game.

You must write your own manual for the game, no matter how poor a writer you are, and even if a professional writer will prepare the final manual. The attempt to write your own manual will increase your respect for the skills of the professional writer, making it more likely that you will have a productive relationship with the writer. Writing your own manual will also provide feedback on the cleanliness of the game design. Clumsy designs are hard to describe, while clean designs are easier to describe. Finally, your own manual will be a useful source document for the professional writer. You

should be prepared for the writer to throw out your manual and start all over---a good writer would rather create a new manual than polish an amateur's crude efforts. You must cater to the writer's needs, answering all his questions as completely as possible. Only a close and supportive relationship between designer and writer can produce an excellent manual. Top

POST-MORTEM

Once the program is out, brace yourself for the critics. They will get their filthy hands on your lovely game and do the most terrible things to it. They will play it without reading the rules. If it's a strategic game, they will castigate it for being insufficiently exciting; if it's an S&A game, they will find it intellectually deficient. They will divine imaginary technical flaws and speculate incorrectly on your deep psychological hang-ups that led you to produce such a game. One critic of mine concluded that TANKTICS was obviously slapped together on a rush schedule; actually, the time between first efforts and final publication was five years and two months. Another roasted ENERGY CZAR (an energy economics educational simulation) because it wasn't as exciting as his favorite arcade game. Don't let these critics affect you. Most critics are far less qualified to criticize programs than you are to write them. A very few critics with the larger publications are quite thoughtful; you should pay attention to their comments. With most critics, though, you should pay heed only to views shared by three or more independent critics. Remember also that even a good critic will roast you if your goal is not to his taste.

The public is another matter. If they don't buy your game, you lose two ways: first, you or your employer make little money on the game; and second, you don't reach as many people with your message. It doesn't matter how beautiful your message is-if nobody listens to it, you have failed as an artist. One failure is nothing to worry about; every artist bombs occasionally. Two failures in a row are bad; three should initiate a serious reconsideration of artistic values. Are you willing to be a noble and starving artist, or a somewhat wealthier artisan? Look within your heart, long and hard. If deep down inside you know that you met your goals, then ignore the critics and the public.