Appendix: Game Design Overview {#gdd}

An Except from "Macromedia Director Game Development" available from Amazon.com

Approaching Game Development

Whether you are an experienced game developer, a master of video games, or even new to computers, a great computer game will offer you entertainment and often some type of competition. Over the years, basic games have evolved to encompass numerous genres such as role-playing, first-person shooter, side-scrolling, strategy, educations, and simulations. Role-play games involve a main character or characters thrust into a heroic plot. A first-person shooter, however, involves adrenaline-fed killing and destruction. Although simulations are generally designed to be as realistic as possible, most side-scrolling, strategy, and educational games tend to be a bit more simplistic. With the wide variety of games available today, the average person will have no trouble finding a game that suits their desires. As you develop you game-design skills, avoid simply mimicking existing games. In most cases, however, any game you develop will fit clearly into a specific genre.

Focusing Your Game

When you begin the process of designing a game, you must first decide how to approach the task. Determining the focus of your game is the best way to begin. You should decide on a topic, purpose, and theme for your game. Listing objectives that you want to accomplish through your game is often a smart way to begin. You might have objectives that direct the activities you will build into the game and a different set of objectives for the person who plays the game to accomplish. Through your thinking process, you will determine exactly what your game is about and how you should go about creating it. The manner in which you focus your game will affect all the decisions you make later in the design process.

Developing A Storyline & Characters

The focus of your game will affect the development of the storyline. A first-person shooter, for example, does not require very much depth. The player wants to shoot things, not listen to a story. However, you can never completely ignore the character and plot. Even the simple arcade classic Pac-Man had a story. Over the course of the game and its two sequels, Pac-Man apparently fell in love, got married, and had a kid, all while avoiding pixelated ghosts. Even the bloodiest and most mindless adrenaline-based shooter game could always use some justification for the rampant killing. If the game is of a different genre, such as a role-playing or strategy, plot and characters will become much more important, as these types of games use storytelling to capture and maintain player interest.

Generally, the plot is not all that advances during play. To attract the player and make the game more interesting, the game designer must determine the events that brought the conflict that the player faces to existence, known as the background story. You can effectively present the background story in a brief introductory scene of the game. If the story is a bit complicated, you can go into more detail in a manual introductory scene of the game. If the story is a bit complicated you can go into more detail in a manual accompanying the game. Even if you decide not to go into detail about the storyline, you should at least have some idea why the game's events occur. The story and setting provide ideas for designing the artwork such as characters and scenes and the interface through which the player interacts with the game.

Designing The Interface

Perhaps the most logistically important feature of your game is the interface. The interface is the layout through which the player interacts with the game. The design is important to the theme of the game. In addition, the interface must offer clear navigation. Your player must never have to fight the interface in order to get something to work. When you design your interface, therefore, you should keep several things in mind.

First, keep the interface simple. Your interface should be straightforward enough that someone who has never used a computer before will be able to pick up your game and intuitively be able to use the controls with little or no awkwardness. The controls should allow the player to play the game effectively without even having to open the instruction manual.

Next, make your interface appropriate for the type of game you design. Control elements are classified as either virtual or physical. Physical controls are hardware such as keyboards, mice, and joysticks that the player actually makes physical contact with in order to play the game. Physical controls are the first part of your interface with which the player will actually interact. Virtual controls are the interface features of your game, such as buttons and drop-down lists, that the player uses to control certain aspects of the game.

The types of controls you use in your game depend largely on the game's focus. A fighting game that involves quick reflexes should include a minimal amount of virtual control and consist almost entirely of physical control. Because that player must use physical controls to access

virtual controls, too many virtual controls can make your game seem cluttered. A strategy game, however, with many menus and interactive charts and maps, should implement a great deal of virtual control.

The next consideration in designing the interface is delivering feedback to the player. Feedback is a term that encompasses all information that the player receives from the game. Feedback is an important part of the interface, but you must control and shape it depending on the necessity of using it in the game. For a simple side-scrolling game, you would probably only need to provide the player with the level number, the number of extra lives remaining, the player's score, and the time left to complete the level. In a strategy game, you would have to provide the player with troop counts, reconnaissance and intelligence reports, weapons and armament statistics, as well as demographics and troop morale, for example. In short, the more thought you game requires, the more feedback you should include. On the other hand, if your game is more fast-paced, you should tone down the feedback as it will distract the player and is probably unnecessary.

Formulating Good Gameplay

Good gameplay never gives the user a chance to get bored. Poor gameplay will result in an unbalanced game that is too hard, too easy, or simply awkward and dull. Balance is the most important aspect of gameplay. There are several ways to make your game balanced and therefore fun play.

Often the best way to balance a game is to prevent the player from having too much power. Good balancing tools often come from real life. For example, when developing a shooting game, you should limit the player's ammunition supply. If the player never runs out of ammunition, they have no reason to ever stop firing at all. Therefore, pressing the shoot button simply becomes a chore, and the user becomes more likely to lose interest.

Next, there should always be a good conflict working against the player that they must overcome in order to proceed with the game. The level of difficulty should not be easy, nor should it be impossible. It is all right for certain parts of a game to be so difficult that it takes the player many tries to overcome them, but the game should not be so difficult that the average player can never make any progress. The average player probably can't beat all 256 levels of PacMan before the game runs out of memory and crashes, but at least they can have fun trying.

When creating nay, you should try to make it easy to learn but hard to master. The game should not be so cryptic and complicated that the average beginning player would not be able to play with relative ease. Some power must be reserved so that to truly get good at the game, you must continue playing and practicing for some time. Games that require little or no skill can be just as annoying as those that are impossible even for the most adept gamers to master. The skill level of your game must fall somewhere in the middle.

When developing single-player games, you might face a dilemma. Without good artificial-intelligence routines, the player will always beat the computer in an equal match. Although good artificial intelligence allows decent competition against the player, the computer characters still sometimes lack the spark of human smarts, leaving them predictable and exploitable. In this case, it is necessary to give the computer a few extra advantages, such as omniscience or a large supply of weapons.

Developing The Logic

Although developing the logic of the game is perhaps the least fun, it is the stonework of the bridge between the design and the birth of the actual game. You should plan the methods you will use to create your game before you begin typing in the code. Break out the old-fashioned paper and pencil and turn off the computer for now. The logic should come first.

Divide your game into various sections, and simplify until you have isolated a single task. Pretend for a moment that you are programming a falling-blocks puzzle game in the spirit of Tetris. When the player presses the left- or right-arrow keys, you want the block to rotate to the left or right respectively. Now that you have isolated a single task, break it down and write it out in "pseudo-code." Pseudo-code is simply a rough draft of a program's logic, not the actual programming language, so it has no real rules. Just write you pseudo code so that it makes sense to you. Do not worry about syntax right now. The pseudo code for the block rotation algorithm could appear as follows:

A>If the left-arrow key is pressed then
A> Rotate the current Block 90 degrees to the left
A>End if
A>If the right-arrow key is pressed then
A> Rotate the current block 90 degrees to the left
A>End if

Notice how the if statements are written in plain English and the conditions for the key presses are express as literal words, instead of ASCII code. Do not worry about syntax or programming-languages specifications right now. Just think it out and put it down on paper.

If you are already familiar with Lingo, you may notice that the preceding code looks similar to the verbose syntax used in the previous version

of “Director” that is now becoming archaic. Do not even think about the similarities between pseudo code and actual programming languages. Your pseudo code can look however you want. Not until you actually write the code will you need to worry about details.

Writing The Source Code

For hard-core game programmers, writing the code is the most fun part of the entire project. For those concerned only with design, however, actually programming the code can seem like a chore. But the fact remains that programming is the most necessary part of game development. Even the best-designed game gets nowhere if you do not sit down and write the code for it. Consequently, programming is usually the most time-consuming part of the project.

However, writing the code is not as hard and scary as many people think it is, providing you have a good grasp of your programming language. When writing a program, you are fighting an endless battle against code entropy. As you write your game, you will notice that it will become increasingly complicated and messy. You can avoid unnecessary clutter by taking advantage of a technique called compartmentalization.

Compartmentalization is the process of breaking your code up into as many small reusable parts as possible. You should try to make your functions as generic as possible so that you can use the same basic function for several similar purposes throughout your game development. For example, when developing a “falling-block” algorithm, you should assign falling speed and block share as parameters into a function. Parameters, or arguments, allow you change how a function behaves by specifying different values in different situations.

By using compartmentalization, You can organize your code to make finding and isolating bugs easier. The sloppily written code causes several problems First, determining the cause of a bug can become extremely difficult. In addition, changing code can become awkward and difficult. If you write generic functions instead of countless sections of scattered code, you will know exactly where to look when you run into a specific problem, and the entire program will be easy to fix.

To be an effective programmer, you should also take advantage of objects. The use of objects differs between languages, but the idea is generally the same. Objects allow you to assign various properties and functions to a single item. For example, an object named “car” might include a “model” property, a “year” property, and a “drive function. By creating properties and functions specific to objects, programming becomes a much more organized and less complicated task.

To keep your games clean and organized, you should always comment sections of ambiguous code. A comment is a line of programming code completely ignored by the compiler. The main purpose of a comment is to keep your thoughts organized and make your code easier for other programmers to understand. Comments can also be useful when debugging your games. You can keep the compiler from seeing specific lines of code to help you find potential bugs.

Checking For Errors

Now that you have written your programming code, you should check it to make sure you have not overlooked any errors in syntax or logic. Syntax errors cause your program to crash or often prevent your program from even compiling. If you understand the programming language, syntax errors are easy enough to fix. However, logic errors, commonly called bugs or glitches, are much harder to catch, because your compiler will never even be aware that the problem exists. You must test your work after each finished aspect of your program to find bus as they come and be able to deal with them one at a time. Never let bugs build up, or you will have to deal with them later, and they may become so overwhelming that you find yourself unable to continue. Whenever you find a bug, try to fix it immediately, or you might create even more bugs.

Preparing Your Game For Distribution

Preparing and packaging your game is the final step of game development. First, you must compile your program into an executable file and organize it, along with any other files required to run your game, into logically named folders. After your game is completely finished, you must decide on the method of distribution you will use. You might decide to give your game away as shareware or even freeware depending on your motivation for creating the game. If you plan on selling your game, you should probably find a publisher who will market and distribute your game and give you a share of the proceeds. If you choose to market the game yourself, you will be limited to selling only as many copies as you can burn onto CDs. Most likely, without any advertising or large-scale distribution, only a limited market of consumers will be aware of your game. If you plan to give your program away, it will be easiest to upload your game to a website where people can download or view it.

Read more?