Proceedings of GREAT Day

Volume 2011 Article 6

2012

Free Radical - Interactive Game Design

Marcos Davila
SUNY Geneseo

Free Radical - Interactive Game Design

Marcos Davila

When learning the basics of computer programming, there is no finer language to start with than Java. It's a language with a lot of potential which is expanded upon with the inclusion of libraries – dynamically loadable code that can be called at runtime. To a person looking to craft a video game simulation in Java, the most important libraries to be concerned with are the ones that handle graphical user interface and input from the user. With complex interaction between the user and the computer, very simple games can be created in a relatively small period of time. Free Radical, a game simulation programmed entirely by an undergraduate Computer Science major at SUNY Geneseo, would be one of countless such games.

There's a lot more to creating a video game – and programming in general – than what meets the eye. The hardest part is always the starting point. Without an idea and a roadmap, there's no way to achieve the end result. Start with a roadmap and no idea and the resulting program will be little better than alphabet soup – methods thrown together with no idea as to why they're needed or what their overall purpose is. Omit the roadmap and start with an idea and the result will be lines and lines of code that work improperly and are riddled with bugs and errors. There are a few things that need to be worked through before programming could begin. The first question that needed an answer was the simplest: what kind of video game was Free Radical going to be?

I decided on making it a space shooter game, modeling it after *Space Invaders*™ but with a faster pace and a steeper difficulty curve. This immediately called for the creation of a panel class, classes to take care of ships and projectiles and enemies. These classes naturally gave birth to necessary methods and data structures such as movement, storage of enemy ships in the memory of the computer and hit detection between projectiles, ships, the edges of the "playing field" and ships and projectiles. The inclusion of necessary methods gave birth to the inclusion of useful methods – methods that wouldn't have an effect on the overall game but would make the end result look much more cohesive. This led to the inclusion of a scrolling background, a counter to keep track of "points" that the player earns, lives for the player, power-ups, background music, detailed sprites for the projectiles, ships and power-ups, and so on.

The first question was the most fundamental question, but the second was just as important: how big is Free Radical going to be? Is it going to be one level, two levels, ten levels, or some number in between? This question poses some more obstacles for a programmer to overcome. If there are going to be additional levels, how will it be handled? Will the current panel refresh? Will a new panel pop up in its place and discard the old one? How will the player transfer progress from one level to the next? Plenty of possibilities were available, each more useful in certain situations than others. After eventually settling on programming for two levels, I decided to create an object that upon collision would teleport the player to the next level.

The final major issue of any aspiring game programmer would be that of portability. The goal is to create as much content as possible while keeping the program at a manageable size. The first few demos of Free Radical ranged from 150 to 200 MB in space, which was a large amount of space for such a small game. The final product sits at 30 MB. The large reduction in storage space stems largely from the implementation of the background music. Java comes with the ability to play back WAV files – lossless files that promise high quality audio fidelity in exchange for hogging up a ton of space on your hard drive. The most common format of playback stems from MP3 files, which Java does not natively support. Now I was at a crossroads – do I want background music and a large program, no background music and a small program, or could I have both the background music and a small program size?

This problem highlights the concept of revision. A programmer must be content with knowing that whatever they're writing now will probably need to be rewritten within the next hour. Hardly any code that is ever written at the beginning will remain as is when the program is ready to be published. A programmer must also become fast friends with their search engine of choice, as they will be surfing the Internet time and time again looking for better ways to approach a problem, or looking for a bug fix so they can progress onwards, or looking for an explanation of

what a method does within a program. I eventually found a way to include both the background music at a smaller file size, but the steps to go about it were not intuitive. This highlights one final and very important idea – a programmer must understand how the algorithm works. There are plenty of ways to approach a problem but different solutions are more evident to certain programmers than others. Very often, you'll find that someone else has a better way of approaching a problem and adopting that is fine *as long as you understand what's being done and why*. The only thing worse than debugging lines of code that you wrote is debugging lines of code you don't understand.

The major misconception about programming video games is that it's 90% fun and flashy stuff and 10% boring details that no one wants to get into. It's a fluid, dynamic process that grows upon itself where every step is equally as important as the next and no parts can be skipped and no corners can be cut. The secret to programming is that it's not about the end product – it's about how you get there.

Bibliography

Horstmann, Cay S. and Gary Cornell. *Core Java™ Volume I – Fundamentals Eighth Edition.* Stoughton: Courier. 2009.

Lorimer, R.J., "MP3: Play MP3's from Java with Javazoom." *Javalobby*. http://www.javalobby.org/java/forums/t18465.html (8 Jul 2011).