Malcolm Wetzstein
3/13/14
AP Computer Science

Independent Java Project Narrative

It's probably good to first begin with the context behind why I chose to work on the project that I did. The reason I started learning to program back in 8th grade was because I was interested in game design. I wanted enough about programming so that I could develop games as a hobby. Back then, I knew I would have to spend a long time learning the basics of programming before I would be able to program anything close to a game. What I didn't know is that between school and other extracurricular opportunities I wouldn't be able to find much time to actually pursue my desired hobby of making games. Four years later, I've been able to do some small projects here and there, but not enough to really say that game programming is one of my hobbies. I saw this independent project as an opportunity to (force myself to) find some time to actually pursue my interest in game programming.

I've always fancied the idea of building a game from the ground up, but I knew that for the time frame of this project, I'd have to find a game engine with most of the lower level graphics rendering code already implemented for me. After doing some research, I came across a 2D game engine called JGame, which seemed simple to use and had enough features to make the creation of a game plausible in a month's time; I decided to use this game engine for my project (Independent Project Proposal). In my proposal for my project, I stated that I would build a framework on top of the JGame game engine (I wanted some of my code to be reusable) and create a full fledged high score game from it (Independent Project Proposal). As I was fleshing out what would go into the framework portion of my project, I started to get a lot of ideas. I soon realized that the framework I wanted to build might prove to be a comprehensive, month long project in and of itself. After realizing this, I spoke with Mr. Kiang and asked if I could focus my project on the framework and perhaps create a demonstration game or tech demo just showing some of its features. This alteration to my proposal was okayed as long as the framework actually took as long to create as I thought it would (and it did).

When I began working on my actually project, I decided it might be a good idea this time to try and make a UML diagram because of all the interacting classes that might be involved in my framework, which I decided to call GameFrame. I spent the first week or so of my project diagraming my project using an app called HandyUML on my phone (Commit #1, Commit #2, Commit #3). Some of the things I decided to include in my framework were things that I thought might prove universally useful for programming 2D games that were not already present in the JGame game engine. Examples of this were classes for managing the creation of game objects and also classes for creating specialized game objects as opposed to the generic JGObject that JGame supported, some of these specialized objects being things like pickups and projectiles. I also planned to include some classes that improved upon features already present in JGame. I didn't like the way JGame handled keyboard and mouse input, so I planned to build my own class on top of JGame's methods for collecting input. The last type of classes I decided to include in my framework were classes that helped to simulate simple physics (Commit #2). JGame comes with an API called JBox2D, which is meant to simulate physics with 2D objects.

When I started to look into the documentation, I realized that it went a little too overboard with making the physics as realistic as possible, with support for things like torque and joints joining together rigid bodies. A lot of these super realistic features were things that would be undesirable for the aesthetics of most 2D games. JBox2D was also missing things, just as ways of implementing force fields. It also seemed incredibly complex and hard to use. I decided to make my own classes in my framework for implementing optional physics that better suited the effects and aesthetics that might be found in most 2D games.

As I began to finish my UML diagram, I began to transfer what was in my UML diagram into code in an Eclipse project (Commit #2, Commit #3). The UML diagram that I created helped me to very quickly lay out the structure of all my code and then start simply filling in methods (Record of Thinking #1). I made sure to comment all of my classes, instance variables, and methods as I fleshed them out and filled in the code (Commit #5, Commit #6, etc.). As I started to fill in methods for all of my framework classes and see how they actually interacted, I realized that there were a lot changes that had to be made from what I originally planned in my UML diagram (Record of Thinking #2). I still think though that, without the organization that my UML diagram laid down, my framework would never have come together as well as it did, with most of my classes taking care of their own interactions with each other.

Another thing that I noticed however as I was working on my code was that I had a lot of confusion between Java and C++ as programming languages. When I first began learning Java, I thought it was very similar to C++, and this led me to a lot of confusion about how references and allocating memory for objects worked in Java (Record of Thinking #1). At first, I tried to work around writing any code that dealt with any tricks with references. I decided along the way however that it was a better idea to probably stop and try and figure out the answers to my questions about how references work in Java. I did some research on the internet and experimented to try and do some of the things I remembered being able to do in C++ in my Java code. After awhile I was able to wrap my head around references in Java and the differences between Java and C++ (Record of Thinking #2).

I was not able to test or debug my framework until it was completed in its entirety. Once I finished writing the code for the framework, I started to create a tech demo showing the features of the framework. As I started to use my framework to create a tech demo, I realized that there were quite a few bugs in my framework (Commit #21 to Commit #25). These bugs happened to be due to several misunderstandings about how the engine I was using, JGame, worked. One of my biggest misconceptions was about how collision IDs worked with tile collisions. I thought that the method for handling tile collisions was called for every tile that an object was overlapping with during a frame when really it was only called once. The collision ID parameter passed to the one call to that method was not the collision ID of one tile, but the AND of all the collision IDs of tiles that an object was overlapping with (all the collision IDs added together). I had to change a lot of code to accommodate this. If I could do this project over again, I would have spent some time experimenting with the JGame engine to figure out exactly how it worked before building an entire framework on top of it.

Eventually, I was able to finish both my framework and the tech demo. When I say finish, I mean that I added all the content to the framework that I had planned to when I first set out on this independent project that I had planned to do by the time the project was due. I still plan to add more features to my framework, as I feel it would make creating games from it easier and faster in the long run.

I learned a lot from this project about how programming games works. I often used to wonder as a kid how the games that I played were coded, and I feel now that I have some answers to those questions after having used an engine like JGame and then coding the GameFrame framework myself. I also learned about the kind of time and effort that goes into programming games and the kind of foundational code that can make that process easier. I also learned a lot about simulating physics in a computer program; my project was a fun application of some of the things that I have learned in my physics class.