Since this was the first time either of us had worked on game programming in Java, this assignment was poised to be a great learning experience. Some of the main difficulties were making sure that everything was synced up with the frame rate, handling the different screens that were required, creating and tracking the “evil” ships, and making sure that the gameplay felt right. The last, being the most ambiguous was the most difficult to get right, but the easiest to implement.

We chose to make the game in an applet, so we were given a few methods to draw objects on the screen (The most important of which was the paint method). At times though, the paint method would take less time to run than it took to move to the next frame. This caused some of the objects to be in random locations at times look as if they were flashing. After debugging this issue for some time, we turned to google to see if others had experienced similar issues. Thankfully, this seems to be a fairly common issue and all that was required to fix it was a Thread sleep. It took a bit more time to figure out exactly how long of a sleep the thread required, but with some tinkering we figured it out.

As stated above, neither of us had ever programmed a game in java before. With that, we figured out how to draw the first frame with ease as we had gone over this in class. But when it came to drawing the high score or options screens, we were at a loss. We ended up taking two different routes for these screens. For the options screen, a JDialog was chosen. The popup provided us the opportunity to allow the user to utilize the mouse without having to write the logic ourselves. It also allowed us to keep everything drawn in the game. At first, he JDialog was another big question mark. Most of the examples we had found in java docs suggested that we use a JOptionsPane, but it turned out to not be customizable enough for our needs. A JDialog provided a highly customizable, fully functional pop up and met our needs very well. The high score screen on the other hand only was shown when the game was over. Because of that, we didn't have to worry about keeping all of the bullets, ships, etc. drawn and we could simply use the paint method with a boolean flag. Also, since the game wasn't technically paused, we were also able to have a restart game key. This played well into making sure our gameplay felt right.

The evil ships posed an interesting problem for us, as they essentially required some very basic artificial intelligence. The rogue ship was the easier of the two to implement as it just moved and shot randomly. The movement was a throwback to our trigonometry classes of middle/high school. It took some research to find the appropriate functions to use, but once we did things worked pretty quickly. The alien ship on the other hand required us to shoot at a player ship. Again, finding the appropriate angle between two points required some research but after that it was just a matter of figuring out which ship to shoot at and firing away.

The final programming challenge that we overcame was getting the game play to "feel" right. Being that this is so ambiguous, it took quite a bit of fiddling around with constants to get right. We had to consider various aspects of the game such as how much the gravitational object should effect ships, how often the evil ships would fire at the players, how much to increase object speeds as levels increase, etc. The gravitational object effects were especially difficult because we essentially had to program a very basic physics engine. This seemed like a much more daunting task when we started out, but again it just required some trigonometry research and some basic programming.

In hindsight, it would have been good if we had spent more time planning out the various classes before starting to implement them. This is because of how the classes ended up being structured. Pretty much every class ended up extending the Drawable class and implementing the AsteroidsObj interface. It turned out to be a fairly clean implementation, but we didn’t add the Drawable class until the Ship and Bullet classes were already completed. Thankfully things worked out well in the end, but we learned that sufficient planning is equally as important as clean code to a good program.

As far as improvements for the project, it would have been useful to have a couple meetings with the TA (or professor) to discuss our thoughts on implementation. Google is a wonderful resource for getting questions answered, but things like best practice or opinions are not easy for google to resolve. Also, a decent amount of time could have been saved if we were given the trig functions that we would need.