**Author: John Shows**

**Software Design and Engineering**

**Lab Document**

|  |  |
| --- | --- |
| **High Level Purpose Statement:** | I’ll be using Gradle to handle dependencies for external libraries. The goal is to learn how to use Gradle to package my project so that it can be distributed to others as an easy to use executable file. |
| **Experimental Design:** | My project will be an experimental game created with the LibGDX framework. Because this framework as many other dependencies, I think it will be a perfect case to study for Gradle. |
| **Resources Available:** | Documentation for Gradle Documentation for IntelliJ. YouTube will likely have plenty of videos outlining out to use Gradle with LibGDX. Stack Overflow will also likely be useful to finding solutions to common problems that will arise. I will also use the LibGDX website, which has extensive documentation related to using Gradle with their framework. |
| **Time Estimate:** | I will reuse much of the project from the previous Maven lab to see how the change to Gradle affects the project 1:1 as much as possible. |
| **Experiment Notes:** | I ran into many issues while converting the project to Gradle, mostly due to my inexperience.  Firstly, I installed Gradle via APT on Linux.  Starting the conversion process to Gradle was easy, I simply ran in terminal: gradle init.  This generated all of the necessary files for the Gradle project structure. I still had to manually delete the .pom file and other Maven related files.   I also needed to set the build tools in IDEA to use Gradle instead of Maven:  File > Settings > Build, Execute, Deployment > Gradle  (continued below)  This changed the file structure of the project, as well as open the Gradle tool pane. The dependencies were automatically setup in the build.gradle which I found to be Gradle’s .pom equivalent, only much easier to use. The syntax uses an easier to read syntax similar to what you would see in JSON or even a standard .Java file. Adding additional dependencies , plugins, and repositories is a breeze. The Maven .pom file with multiple nested layers XML syntax is much harder to read.  After building the project and removing old Maven files, I ran into a bit of a snag. I kept getting errors about certain methods in the build.gradle not being recognized. Upon further inspection, I found out the version of Gradle I had installed from the repositories used by APT was 15 years old. Thus, all of the Gradle related files that were generated were deprecated. After removing those, I updated Gradle and reran gradle init.  This time, every ran perfectly. The only thing left I had to was add the ShadowJar plugin to the build.gradle file which allowed me to package all the dependencies with my project into an executable .Jar file. |
| **Results:** | The results were a success. As far as I can tell, the project made a transition over in a 1:1 state with minimal work, besides just researching how Gradle works. |
| **Consequences for the Future:** | I found Gradle to be much easier to use than Maven, especially with it’s much friendlier way of handling dependencies, plugins, and other project configurations. It also happens to be the preferred method for LibGDX projects, which means going forward, I’ll have plenty of documentation from the LibGDX team and help from their community. |