THE LEGEND OF ADLEZ

My Contributions to The Legend of Adlez

Micah D.

Taking the a java Swing game template graciously provided to use by our professor, the first thing I did was refactor it to use different directories and upload it to GitHub, our source control provider. Shortly after this, I modified the game to remove the zombies and add destroyable “Goblins” a feature that would not stay in the game, but was ultimately refactored into an abstract EnemyAI class that all “bad” NPCs would use.

After a move to the Maven build system, done by my teammate Andrei, to make it easier for those using IDEs to build the project, I then started working on rending the map. We had opted to use *Tiled* as our map creation tool. *Tiled* primarily exports to two file formats: TMX (Its internal format) , and JSON. We opted to write a TMX parser, since TMX is just XML, and XML is fairly boring. While writing this code we had a lot of time to ruminate about Java’s unofficial motto: “Java: It’s like C# but worse.”

The first iterations of the game to include the map, only included it as a background image drawn by the graphics layer on every clear (instead of a white rectangle from the template). The background image was rendered from a list of tiles (represented by *ImageRefTo*’s, parts of images, that also have projected coordinates) once when the graphics system was instantiated. At this stage the map was only concerned with layers, which contains the tiles to use, and tile sets which gave more information about the tiles. In order to load these tiles, we had to load the tile sets as images first. The map was made to implement a *MediaInfo* interface and passed to the *GraphicSystem* on creation.

After we had the background layers rendering, we needed to resize the world to the size of the map. I defined an interface in the engine layer that to be implemented by the map. There were some problems with this and different screen sizes, however, so I created another class in the graphics layer to provide the screen size. The world info (as implemented in the map) could use screen info class to determine what parts of the world to show. Also during this phase I added an animated layer on top of the background that would change based on animations set in the *Tiled* Editor.

During the first two phases of Tiled map integration, the player was represented by a bush (or something that looked like a bush), because that was what happened to be in the top left-hand corner of one of our tile set images. The next step, however, was a little tricky because we needed a way to easily reference images that didn’t exist like in the way our tile set assets did, because they had to be stitched together from lots of different tiles. So I created the *VirtualImage* class, to hold a list of *ImageRef*’s (essentially tiles), and a name. These *VirtualImage*’s are rendered just like the background image is and can be referenced by their names (as opposed by path, like normal images). We load these virtual images from object groups that are defined in the TMX, and match them up to layers that have the same name. Having these *VirtualImage*’s allowed us to reference complex images normally, and allowed us to stop using a bush as our avatar. We graduated to using a rock instead.

Now that we had objects being read in from the map, we needed to start assigning them to objects in the game. Part of this was creating blank hit boxes (such as for trees or water) and some of it was objects that would need to make with their image move with them (such as NPCs). Since we had some rather large hit boxes for bodies of water, at this stage I modified the collision detection to work on either *RectangularGameObjects* or *CircularGameObjects*. This way objects that worked better as circles (such as tree bases), and those that worked better as rectangles (such as the house bases), could both be represented. I say tree bases instead of trees because I also introduced a foreground layer at this stage which the Trees and Houses are drawn onto. This gave the illusion of some depth.

Having completed most of the TMX import with the help of my teammate Sevde, I added a sword and sword swing to the map, and had the avatar draw it. The avatar was also changed to reflect is last x moment that he made. This required modifying how we used the *GraphicSystem.* Up until this point, objects were given to the graphics system, and the graphic system would draw them. That worked pretty well when we only had simple objects, but more and more complex objects (such as a player with a sword) were causing the graphics layer to be dependent on the game layer, rather than the other way around. By flipping things around and expanding the *GraphicSystem* interface to expose more primitive, but still high level, drawing functions, GameObjects could now draw themselves after being provided a *GraphicSystem*, cleaning up our graphics layer considerably. Many thanks to my teammate Ivaylo who initially suggested this.

After we had a swinging sword, I went on to draw and integrate a bow and arrow into the game. I also worked some on the primitive NPCs that would be come to be known as “Monsters”, and made the friendly NPCs move about randomly, which helped them seem “alive”. I showed this early version of the game to a friend outside our development group to get their opinion and integrated many changes they suggested including, adding an animation highlighting where you start, and making the animals follow you home in one of the quests.

In the final leg of development, I made a number of cosmetic fixes for the game. Including, drawing and loading a logo in the main menu, and rounding the corners of many parts of the interface. I also implemented the Boss fight, giving the Boss (Darth Avaj) the ability to spawn orbs that would hurt, teleport, and generally hinder the player in their goal of defeating him. In this phase also I worked with my teammate Josh to implement the game save and load, allowing you to re-spawn. Implementing the game save and load was a testament to how Object Oriented design often leads you down the wrong path of development.

In conclusion, I had a great time working on this Game with my team. I hope others have as much fun playing this game as we did making it.