Emblem of Legends Developer's Notes

Brian Forbes, Hongyu Wang

January 15th, 2015

ICS4U1

Ms. Strelkovska

Description of the Game

Emblem of Legends is a turn-based strategy game that draws inspiration from popular games such as Fire Emblem [3] and League of Legends [2]. We created an online-multiplayer version that can be played from any two computers, as long as they are connected through the internet. In order to do this, we used technologies such as SOAP servers, SQL databases (and Java Database Connection, JDBC), and Swing GUI. This document is a guide to the code, sweat and tears that went into this huge project, broken down package by package. There are a total of 238 classes used in this project.

Folder 0: TurnBasedGame

In this folder is all of the picture files (All of the game art, excluding the background images in the client, was created by our classmate, Molly Li), the config file ("config.txt"), the map files ("swampy.map", "legacy_swampy.map"), and 3 runnable jar files ("game.jar", "gameLauncher.jar", and "createAccount.jar"). First, verify the config file is correct. There should be one line saying "url=http://sxtesta.sys-x.com:8080/BrianSoap/services/BrianSoap". If this line does not exist, replace the previous "url" line with it. If there is another "url" line, delete it. The other important line in this file is "sound". If you wish to play with sound, ensure it is set to true. If not, set it to false. You should not edit the map files, as they will not properly affect the game if they are changed. "game.jar" is an old version of the game and should be ignored, but the two important files here are "gameLauncher.jar" and "createAccount.jar". In order to play the game, you must first go to "createAccount.jar". You do not need to worry about your password being accessible - it is stored as an encrypted byte array, and is never sent or stored in plain text - not even to create your account. Once your account is made, you can proceed to "gameLauncher.jar", login, and play .

Folder 1: TurnBasedGame/Sound

This folder contains all of the sound files needed to play the game. There's no need to play these yourself, as they can be easily accessed through the game.

Folder 2: TurnBasedGame/src

This is the main folder that contains the source code for most of the game. In it you will find many packages:

- **Character (Main Developer: Hongyu Wang):**

This package contains all of the client-side characters. Many of their methods are currently unused, however if we were to change to game to include a single-player, they would become much more important. In general, each character has one main file and four spell files. The main file sets their stats and images, and the spell files set their spells. Each separate character has it's own subpackage.

client.login (Main Developer: Brian Forbes)

The classes in this package each deal with logging the user in and starting the game. CreateAccount takes a string username and password from its GUI interface, and sends them to the server to make the account. Before sending the password, it uses the SHA-256 to hash it, so the plain-text password is never sent, saved, or stored. The Login has a very similar interface, but returns a key if the login is successful. In the same way, the username and password are sent to the server to be verified. Again the password is hashed before sending and compared to the stored password hash for the given username. GameGenerator simply creates a GameWaitThread (TurnBasedGame/workers), which is discussed in detail later in the notes.

- **Client (Main Developer: Hongyu Wang)**

Backbone

The Backbone package contains several key classes. These are like mini-frames that don't really hold any real content. Main is the panel that has client and center. The center panel is a card layout setup controlled by the extensions to my button in the tabs file. The tab panel is a super class for each individual panel. ImageButton and MyButton are basically the way that I control the card layout in center.

Armory

The armory package should in all honestly be renamed to Almanac. Originally, we had decided to directly incorporate items into the game that you could purchase after games within the client. However, time was short, and this functionality was never achieved. Instead, however, we decided to make this package contain a general character description for each character, along with each of their specific spells. Armory is divided into two main class sections, Left and Right. The Left panel is a null layout that has inside all the character images. These character images then control the Right panel. Inside the Right panel is a variety of Spell panels, which control a Character panel which has a paintComponent method that basically draws each of their spell descriptions.

Empty

This is just nifty background home page that you can access. It contains a single JLabel that pops out that is controlled by a timer. The background is controlled by the profile panel.

Instructions

This is a singular panel that basically tells a person how to the play the game. This reads from an instructions.txt file, and is displayed to the screen using a bit of bulletproofing.

Profile Panel

Profile Panel is a Grid Bag Layout that has a few elements inside of it. Each element draws information fed by the server into the client. The first thing that it draws is the username of a person. Inside the profile panel is also a button that can change the main theme of the client. There is a win losses panel that draws info from the client, as well as an in queue indicator.

Shop

The Shop Panel is a grid bag layout that has two main panels. There was a planned item system as I stated earlier, but that was eventually scrapped as time constraints prevented us from effectively implementing that. The item panel will create item objects and the item shelf panel basically stores an Icon along with a JCheckBox to determine if the item is bought. This code also accesses the user arrayList to update it for easy server implementation. The Confirm Panel will basically add everything to the user’s item arrayList.

TeamSelection

The teamselection section of the code has two main sections. The first section is the top part which displays each individual icon. The second section contains a variety of character icons. If a character is in the team selection panel, clicking on their portrait will remove it. Otherwise, it will add the character. This is all controlled by the HeroShelf class which sends information to the Top to display using 10 JLabels. The save button saves the team in the desired order into the user class, which can then be written into the server.

- **game (Main Developer: Brian Forbes)**

Game contains the files core to the battle system.

AoEDmg, Poison, DeathMark, DamageModifier, SigilOfSilence,SuperHeal, and UnAffectable are all subclasses of EffectTimer, which is used to make temporary effects. When an EffectTimer is made, it affects stats on a character. Every turn, a method is called on the Character class, which in turn calls a method on all of their EffectTimers. When an EffectTimer runs out of time, it resets all changes to the character.

Blank, Fort, Grass, Rock, and Swamp are all subclasses of Terrain. They are the different map tiles that exist in the game. Each has it's own images, defence values, and effect the characters in a different way.

Character is the main file for characters in the game. All main character files in the Character package are subclasses of Character. It contains all main methods needed for characters, including their stats.

Battle is the basic class to run a battle. It has all the core features, including holding the map class, the characters, and the ability for characters to interact with each other.

Map is the last main class in this package, and it holds the map the game is played on. It consists mainly of two arrays, an array of Characters and an array of Terrain. This terrain array is the base map, and on top of it are the characters in the Characters array. Movement is done through this map class. ***Recursion*** is used for the getPossibleMoves and getPossibleAttacks methods.

- **gui (Main Developer: Brian Forbes)**

Gui contains all of the files for the game window.

The main class here is Game, which holds all the main methods for running the game. It holds the battle and is the main interface between the GUI and the rest of the code.

Root is the main frame that holds all of the other components.

CharaList is the list of characters on the left side of the game GUI. It updates with health bars and when a character is killed, they are removed from the list.

TopBar is bar on top that contains all menu items. It only contains a few small items, but contains more space to put more items, were they required.

StatsWindow is the window in the bottom right where the stats of the selected character are show. Whenever a new character is selected, a method is called on it to update the selected character for the panel. It also contains four SpellButton instances. These update to the spells of the selected character, which can be cast by pressing them and selecting a target.

ConsoleBox is a simple console that can be found on the top right of the game GUI. It outputs all the text needed for the game, mostly getting information to print from the server to update the user on effects of their actions.

- **server (Main Developer: Brian Forbes)**

The server folder consists almost entirely of generated code. It is generated by the Apache Axis “wsdl2java” tool, which, given a wsdl file, generates client side or server side code. The wsdl file in question is also generated (this time, by the “java2wsdl” tool) from a previously created java file. This can be found here [1]. The implementation of all server code can be found in the “src (server)” folder.

- **spells (Main Developer: Brian Forbes)**

This folder contains the base spell class. All other classes in this packages were used for testing and are either implemented somewhere else (spells.DamageMeleeOne) or unused (FakeSpell).

- **workers (Main Developer: Brian Forbes)**

Workers contains all the thread subclasses used in this project. SoundThread plays a sound, and can loop given an amount of time. In order to loop, after the given time it resets the sound file being played.

GameWaitThread is used when the user becomes in queue. It waits until the server returns true for the hasOpponent method, and then gets the characters for the battle. After that, it creates the Game window.

BattleThread is the main connection between the server and the clients. It has two queues (one incoming and one outgoing). Method calls on the thread either return items from the incoming queue or add items to the outgoing. For each item in the outgoing queue, it asks the server if it is possible, and then sends the action if it is.

Folder 2: TurnBasedGame/src (server)

- **server (Main Developer: Brian Forbes)**

This folder contains all of the server code. The root package server contains mostly the generated code (through the “wsdl2java” tool again), although the most important class is ServlerImplementation. This code is the only non-generated file in the package, and it contains all of the important methods for the server. Some of the methods are directly implemented there, and others in server.serverSide.workers.BattleThread. Most important are the database methods, which use SQL to retrieve and store data. The most obvious example is in ServerImplementation.createAccount, which uses three different SQL statements to check if an account is creatable, one to insert the account into the Users table, and one to add the user to the Teams table. ServerImplementation also contains two Hashtables: keyToUser and keyToBattle. keyToUser stores all logged in users, while keyToBattle contains the keys of the users who are in battle mapped to their respective BattleThreads. When a user logs in, they are given a key. This key stays with them for their entire session, which is used to access all main methods through these two hashtables.

- **server.Beans (Main Developer: Brian Forbes)**

The classes here are mainly auto-generated by “wsdl2java” based on previous bean classes I wrote. Their job is to take information from the server/database and package it to be sent to the client. MapB was intended to be for sending maps, but was never used. The required methods for converting a Map to and from MapB were implemented, but we did not have enough time to store maps on the server and implement choosing between maps. User packages information from the Users table, mainly used to send information to the server about a given user (or store them in one of the servers two main Hashtables). Profile is used to send this information to the client. Team stores a team of characters in the form of a string array. When the Team variable is used to send battle information to the client, the string array includes data about the id and position of the character.

- **server.Database (Main Developer: Brian Forbes)**

The only class in Database is DBAccess (standing for Database Access), which uses JDBC to create a connection between the server and the database (housed on the same server). It has very simple methods and is mostly used for creating the connection.

- **server.serverSide (Main Developer: Brian Forbes)**

server.serverSide consists entirely of server-side versions of code originally written for the client (for example, server.serverSide.Character contains the server-side versions of all of the characters). The key difference between the client and server-side codes is the calls to getThread in most of the classes. This refers to the associated BattleThread (server.serverSide.workers) for the object. This BattleThread is, similar to the client-sided one, however mostly works in reverse. It also has two queues, except in this case, they correspond to the “A” team and the “B” team in each battle. Each action is sent to both queues, which are picked up by each corresponding client.

- **General Notes**

Brian: The hardest part about this project for me was probably my inability to notice some of my mistakes until it was too late. When I first wrote some of the code, I did some weird things, such as using "==" for strings instead of .equals. It (somehow) worked until very late in the project, when I had to go back and change them all. Another big example of this was my client-sided BattleThread, which was not a legitimate thread until very recently. I implemented the two-queue threaded system to prevent the blocking which was caused by the time taken to call methods on the server. The server-sided BattleThread is still not a 'real' thread, but there was no reason to implement a two(really four, two queues per team) queue system as it would not seriously affect efficiency. This project as a whole has been the most difficult, but the most fun coding task I've ever attempted, and I'm incredibly proud of the fact that I saw it through to completion.

Hongyu: The most difficult part of this project for my section of the code would lie in the Team Selection panel. In the original way that I did the team selection panel, I would remove the Icon from top. This however was incredibly inefficient, and was extremely laggy even on my home computer. As a result, I ended up changing the code to just simply reset the JLabel to a different Icon. Another very difficult part was the creation of the Armory panel. This section took a large amount of effort and time in order to create well due to the sheer nature of the way that the Left half will control the right side. I quite enjoyed working on the project and seeing it come together at the very end, and gave me a great sense of accomplishment.

- **Reference URLs:**

1. http://tinyurl.com/pna6bfc
2. http://tinyurl.com/nwh8hm
3. http://tinyurl.com/2fay3r