Technical Overview WHITEPAPER

By Michael Angstadt, www.mangst.com, 8/30/10

Introduction

The following document describes some of technical details and development notes concerning Bemuled!.

Libraries

Trident[2]
This is an animation library that can be used with Swing and SWT components. It provides a simple API which allows you to do things like make the background color of a button fade into different colors. I used this for changing the sizes of the cell images, as well as for the "red flash" effect that you see when you try to perform an invalid move.

MigLayout[3]
This is a layout manager (like Swing's BorderLayout and FlowLayout). It allows you to do pretty much anything and the API is dead simple to use.

Apache Commons Codec[4]
Used solely for its base64 functions.

Trident callback problem

One problem I ran into was with the Trident animation library (which worked 100% perfectly the rest of the time). The library allows you to define a callback method which will be called when an animation completes. However, in two separate cases, this wasn't quite working for me. It was acting as if it was calling the callback method one or two frames before the animation was actually complete. The first case was solved by simply calling the repaint() method on the component I was animating. But this did not work for the second case. I ended up figuring out the following hack: I changed my Parallel scenario into a Rendezvous scenario, called a rendezvous() after adding my animations, and then added an "empty" animation. This is demonstrated in the following code sample:

```
TimelineScenario.RendezvousSequence resetScenario = new
TimelineScenario.RendezvousSequence();
resetScenario.addCallback(new TimelineScenarioCallback() {
      //...
});
```

```
//add animations using resetScenario.addScenarioActor()
//...

//wait until all the above animations complete
resetScenario.rendezvous();

//run this "empty" animation for 100ms
Timeline emptyTimeline = new Timeline();
emptyTimeline.setDuration(100);
resetScenario.addScenarioActor(emptyTimeline);

//play the sequence
resetScenario.play();
```

Cell image resizing

If you hover the mouse over a cell, you'll notice that the image increases in size slightly, to give the player a better sense that the game is responding to his actions. I've found that scaling the images before-hand and caching them into memory greatly speeds up the fluidity of these kinds of animations. Scaling the images on the fly can result in a choppy animation if multiple animations are executing concurrently. So, I make scaled copies of each image from 10 pixels in size to 60 pixels in size, for a total of 50 images per image.

```
//generate the scaled copies of each cell image
scaledCells = new ArrayList<ImageIcon[]>(originalCells.size());
for (ImageIcon imageIcon : originalCells) {
      Image image = imageIcon.getImage();
      ImageIcon scaled[] = new ImageIcon[51];
      for (int j = 0; j < scaled.length; ++j) {
            int max = j + 10;
            int scaledWidth, scaledHeight;
            if (image.getWidth(null) > image.getHeight(null)) {
                  scaledWidth = max;
                  scaledHeight = image.getHeight(null) * max / 
image.getWidth(null);
            } else {
                  scaledHeight = max;
                  scaledWidth = image.getWidth(null) * max / 
image.getHeight(null);
            }
            scaled[j] = new ImageIcon(image.getScaledInstance(scaledWidth, 
scaledHeight, Image.SCALE_SMOOTH));
      }
      scaledCells.add(scaled);
}
```

High Score list

The high score list is stored on my mangst.com website. Bemuled! retrieves it using a HTTP GET request, and submits a new high score using a HTTP POST request. Both requests go to the same URL--http://www.mangst.com/bemuled.php. The requests are handled by a PHP script, which uses a MySQL database to store the high scores. It also records a count of how many times the high score list was retrieved.

Launching

Java WebStart is used to launch the application. The way this works is as follows: The user downloads a JNLP file, which is an XML file that contains details such as where the application's JAR files are located and what permissions the application requires.

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0+" codebase="http://www.mangst.com/projects/bemuled/"
href="bemuled.jnlp">
      <information>
            <title>Bemuled!</title>
            <vendor>Michael Angstadt</vendor>
            <homepage href="http://www.mangst.com/" />
            <description>Bemuled! is a game similar to the popular Bejeweled
games.</description>
            <icon href="bemuled-jnlp-icon.png" />
            <offline-allowed />
      </information>
      <resources>
            <j2se version="1.6+" />
            <jar href="bemuled.jar" main="true" />
      </resources>
      <application-desc main-class="michael.bemuled.Bemuled" />
</jnlp>
```

The JNLP file for Bemuled!

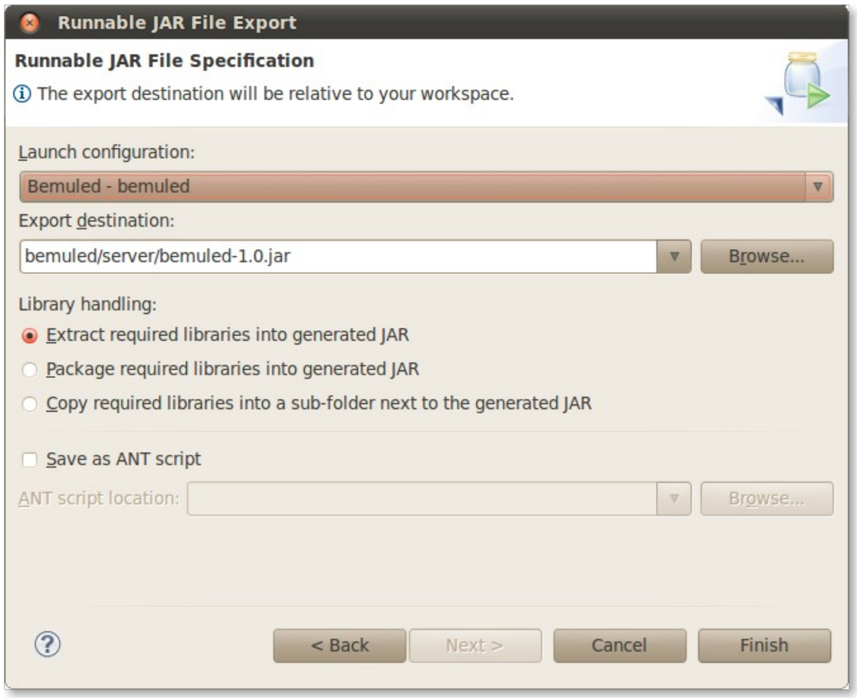
The JNLP file is then executed by the user's Java installation. It first downloads the necessary JAR files that are defined in the JNLP (or re-downloads them if they have been updated since the last time the application ran). If the application requires extra permissions, such as access to the local file system, then a warning dialog will appear asking the user whether he trusts the author of the application. If the user clicks "Yes", then the application will launch. Because Bemuled! does not require any of these permissions, the application is launched as soon as it is downloaded, without displaying this scary warning dialog.

Bemuled! sends HTTP requests (for the high score list), which normally would require these extra permissions. However, requests going to the same domain that the JAR came from are not restricted by this rule and so do not require the extra permissions.

The JAR must also be "signed" if these extra permissions are required. This is one less thing that has to be done to the Bemuled! JAR file, thus simplifying the deployment process.

Building the source code

The application was developed using the Eclipse IDE. It is packaged into a "Runnable JAR" using Eclipse's "Export" feature. The class files from all the dependencies are extracted from their corresponding JAR files and put directly inside Bemuled!'s JAR file using the "Extract required libraries into generated JAR" option. This makes running the application simpler and more maintainable. For example, if further development work were to be done on Bemuled! and a new library was used, the JNLP file would not need to be updated, because all dependencies are packaged inside of a single JAR.


Runnable JAR File Export

Runnable JAR File Specification

The export destination will be relative to your workspace.

Launch configuration: Bemuled - bemuled

Export destination: bemuled/server/bemuled-1.0.jar

Library handling:

- [✓ Extract required libraries into generated JAR]
- Package required libraries into generated JAR
- Copy required libraries into a sub-folder next to the generated JAR

☐ Save as ANT script

ANT script location:

The Eclipse dialog for the Runnable JAR File Export feature

I was unsuccessful in using the "Package required libraries into generated JAR" option. I would prefer this option over the former option because it puts the dependency JAR files directly inside of the main JAR instead of extracting their contents, thus keeping them separate from the application code. But WebStart apparently does not like this, as it refuses to launch the application without even displaying an error message. This may be because under normal conditions, Java does not allow JAR files to be accessed from inside another JAR file. Eclipse has to add special class files of its own to make this possible, which may interfere with WebStart.

Mac OSX Integration

Although Java is a cross-platform language, there are some extra steps you can take to better integrate your application into the Mac OSX environment. The JDK for Mac provides additional classes which allow for this integration. Reflection can be used to access these classes so that the application can be complied on any operating system. However, when the application is run from WebStart, a security exception is thrown, preventing this Mac-specific code from running (it will not allow you to access the Application class). This is not all bad however, as some of the integration functionality happens automatically. For example, it uses the icon file defined in the JNLP as the dock icon (what you see when you alt-tab). However, you are not able to do things such as creating an About or Preferences dialog. If the JNLP were configured to require extra permissions, then full access to this code would probably be allowed, but then the user would be confronted with the security warning dialog box, which could scare the user and prevent him from wanting to run the application.

Artwork

All of the artwork was found on the Internet through Google image searches (I'm not an artist, so did not create my own art). The "Bemuled!" title image at the top of this document was created using the free Xara 3D Heading Maker[1].

Links

[1] http://site.xara.com/referrer/headmaker_trial.asp
[2] http://kenai.com/projects/trident
[3] http://www.migcalendar.com/miglayout/
[4] http://commons.apache.org/codec/