Development Report: Maze Game

By Peter Mitchell (mitc0271)

1 Introduction

This document outlines a general overview of the different points of change that had to be made to convert the maze game that was developed for minor assignment A on the Android platform. The game was successfully ported to Windows Phone and appears to provide the same appearance of response to the different stages of the game. The changes have been grouped by the types of changes that they were. These include: class diagram changes, language changes, data type changes, general graphics changes, database changes, miscellaneous changes, handling of sensor inputs, and a comparison of development environments. In terms of general development is was found to be particularly easy to change between the platforms, although this was likely particularly due to the original design being built with the intent of changing to XNA in this second project.

2 Class Diagram Changes

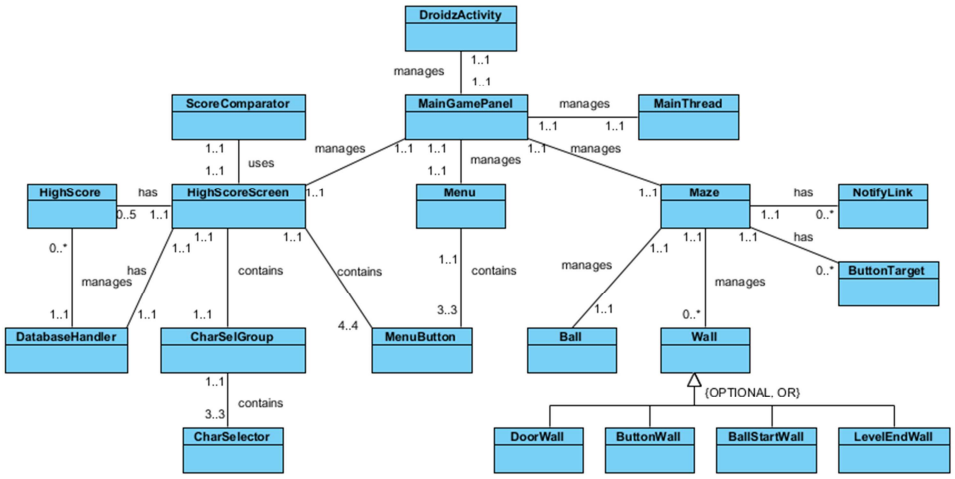

***Figure 1**: Original class diagram for the Android version.*

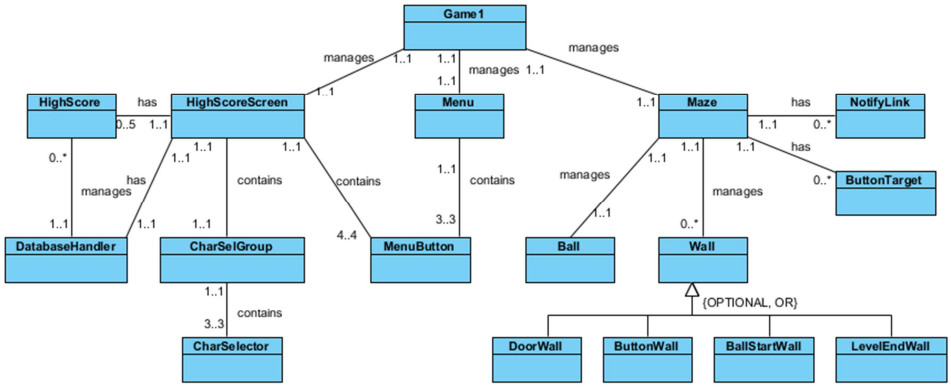

***Figure 2**: Modified class diagram for the XNA based Windows Phone version.*

As can be seen from observing the two diagrams on the previous page there are a very limited number of structural changes that occurred. XNA managed the entry point code that was in DroidzActivity and MainThread, so the core code between the original set of three classes was combined into the single Game1 class. The code was always designed with this specific structure in the Android version so that the change would be so simple when moving the code to XNA.

The only other class structural change is that ScoreComparator as a class was dropped. This was because the representation of a custom collection sort is handled differently in C#. This particular change in sorting algorithm is discussed later in section 4.5.

3 Language Changes

A large part of the low level changes when converting the application from Android to XNA for Window phone required language syntax specific changes. This was largely to do with the structure of creating classes and methods. A summary of some of the key differences that need to be changed follow.

3.1 Inheritance

In Java Inheritance is specified by using the “extends” keyword, but in C# this is specified by using simply a “:”. Another difference is the use of the base or super class’s constructor. In Java this is specified as a method call inside the child constructor with super(….) where the dots represent any relevant parameters. In C# this is specified outside the block of code for the constructor and is instead specified by following the method definition with a “:” and then a base(….) method call to run the appropriate constructor’s code.

3.2 Methods

Method naming of core functionality differs between Java and C#. Java follows the convention of first letter of variables and methods as lower case and the start of following words in the variable or method as a capital. In C# developed by Microsoft their standards follow the same approach except the first letter is also a capital. Thus there are very similar methods for both languages, but it requires a lot of case changing when swapping between languages when there are a lot of framework based method calls. Overriding of methods can be applied more easily in Java, the inclusion of @Override is optional, and it is possible to override methods without them needing to be abstract or other methods of indicating this. In C# to override a method in an inherited class the method must include the keyword “virtual” in the version that can be overridden, and the keyword “override” in the ones that do the overriding.

3.3 Loops

The “for each” type loop structure is defined differently between Java and C#. In Java this uses the structure for(DataType temp : arrayVariable). This is comparable to the C# version that uses the structure foreach(DataType temp in arrayVariable).

3.4 Arrays

Defining arrays in Java and C# varies. To define a multidimensional array in Java the syntax is:
DataType[][] variableName = new DataType[rowCount][colCount];
In C# you would write this as:
DataType[,] variableName = new DataType[rowCount, colCount];
Then when using an initialiser list in Java:
DataType[][] variableName = { { v1, v2, v3, …. }, { v1, v2, v3, …. }, …. };
Then in C#:
DataType[,] variableName = new DataType[,] { { v1, v2, v3, …. }, { v1, v2, v3, …. }, …. };

3.5 Switch Statements

In Java the “default” clause is optional, but in C# it is enforced as a required statement.

4 Data Type Changes

4.1 Imports

Obviously all the core imports are different between XNA for C# and Java and had to all be changed to the appropriate includes.

4.2 Booleans

In Java the full “boolean” must be written, but in C# the variable is written in the shortened “bool”.

4.3 Rectangles

In Java for Android the variable of Rectangle used to represent the location in screen space is defined using the RectF data type. RectF represents four coordinates using four floating point values for the Top, Left, Right, and Bottom. These are the values that must be supplied when creating the Rectangle. Then when changing to the XNA C# variable of Rectangle the values are stored as Integers instead, and represent the x (left), y (top), width, and height. Therefore the main step to converting between these types of Rectangles in the definition is to remove the adding of the x and y coordinates from the width and height that would be used to define the right and bottom values.

4.4 General Data Type Changes

The Java Canvas has the equivalent SpriteBatch in XNA. The Java Bitmap has the equivalent Texture2D in XNA. The Java Context has the equivalent Content in XNA.

4.5 Sorting Collections

In Java the Comparator class is inherited as an interface and the compare function overridden to provide the sorting functionality. In C# for sorting collections the delegate function can be used to create the comparison function and pass that directly to the sort method. The code for each can be seen below.

Java sorting:

```
import java.util.Comparator;

public class ScoreComparator implements Comparator<HighScore>{

    @Override
    public int compare(HighScore score1, HighScore score2) {

        float rank1 = score1.getScore();
        float rank2 = score2.getScore();

        if (rank1 > rank2){
            return +1;
        }else if (rank1 < rank2){
            return -1;
        }else{
            return 0;
        }
    }
}
```

C# sorting:

```
scores.Sort(delegate(HighScore s1, HighScore s2) { return s1.getScore().CompareTo(s2.getScore()); });
```

5 General Graphical Changes

5.1 Fonts

In Java Strings can be drawn to the Canvas without needing to specify any font or anything particularly special. In XNA with C# however, to display test rendered using the DrawString method for the SpriteBatch a SpriteFont must be manually loaded in. This requires having additional font files. SpriteFonts typically don't upscale well either. So as the SpriteFont files that were included in this project were low in resolution, the output when up scaled to match the sizes in the Android version made them appear blurry.

5.2 Circles And Rectangles

In Java there are methods provided by the Canvas to draw basic shapes such as circles and rectangles. In XNA there is not this provision without using additional libraries. For this reason it was far simpler to just represent the circle and rectangles as sprites. This did require development of a generic rectangle and circle sprite, but then these were modified by changing the colour setting to match the colours used in the Android version.

5.3 Screen Orientation

Screen orientation was a large issue in terms of the visual effect for both devices. When the orientation changed the game would render half the content off the screen. So a fix in both cases had to be found. In the case of Android changes had to be made to the manifest file to allow the permissions to handle the orientation and to set the full screen property. Then additional code provided to set the options from within the activity onCreate method. In XNA this was far simpler. The default was full screen, and instead of having to modify all sorts of settings the orientation could be locked with a single line of code.

```
graphics.SupportedOrientations = DisplayOrientation.LandscapeLeft;
```

6 Database Changes

The Android SQLite database is a handy feature to have accessible. Although for this project only a single table was used with four different values (unique id, map number, score, and name). XNA does not directly support a SQLite database without applying additional features. Therefore based on the simplicity of the data a simple XML based database was built. Both database implementations can perform a “SELECT” type operation, an “INSERT”, and a “DELETE” too. This was all being handled by direct functions in the case of XNA to perform the appropriate actions.

7 Miscellaneous Changes

7.1 Time Tracking

In Java with Android there was no way to natively have the tracking of elapsed time except by calculating it manually using a function to get the time in milliseconds. XNA provides this through the ElapsedTime mechanism.

7.2 Collisions (ball offsets)

Strangely with the exact same code for all the calculation of positions in the game the offsets for collisions of the ball that were inserted to fix the collision on Android were different to fix it for the Windows Phone. On android both the x and y coordinates had to have 1.4 times the radius subtracted to align the collision correctly. This can be compared to the value on the Windows Phone with XNA this was only 0.5 times the radius to provide the same effect. I suspect it’s because of an issue somewhere hidden in the collision code, but it would take more time than it is worth to find the issue at this stage.

8 Handling Sensor Inputs

8.1 Touch sensing

On Android the handling of touch events is handled via an event. On XNA this is handled like any other standard form of input. During the update method the following code was used to access all the touch locations (supports up to 4) and the touch location is passed to the relevant window. Checks can be done at this stage for moving touch points as well, although this was not needed for this application.

```
// Process touch events
TouchCollection touchCollection = TouchPanel.GetState();
foreach (TouchLocation tl in touchCollection)
{
	if ((tl.State == TouchLocationState.Pressed))
	{
		if (menuEnabled)
			menu.handleTouchEvent((int)tl.Position.X, (int)tl.Position.Y);
		else if (!levelFinished)
			maze.handleTouchEvent((int)tl.Position.X, (int)tl.Position.Y);
		else
			highScoreScreen.handleTouchEvent((int)tl.Position.X,
		(int)tl.Position.Y);
	}
}
```

8.2 Accelerometer

Handling the accelerometer was easy on both devices. To accomplish this in XNA the first step was to execute the following code.

```
accelSensor = new Accelerometer();

// Add the accelerometer event handler to the accelerometer sensor.
accelSensor.ReadingChanged += new
EventHandler<AccelerometerReadingEventArgs>(AccelerometerReadingChanged);

accelSensor.Start();
```

After this code has been executed the AccelerometerReadingChanged method fires at regular intervals with the arguments supplied to the program as double data types. The difference is that the values are not the same as the reading from the Android device. The reading in XNA provides a value between -1 and 1, but the Android provides one between -9.8 and 9.8. To simulate this same behaviour to ensure the program worked the same a multiplication by -9.8 was applied to correct the readings.

```
public void AccelerometerReadingChanged(object sender,AccelerometerReadingEventArgs e)
{
	accelReading.X = (float)e.X * -9.8f;
	accelReading.Y = (float)e.Y * -9.8f;
	accelReading.Z = (float)e.Z * 9.8f;
}
```

8.3 Other key inputs (back button)

On Android the back button is handled as an event using the onBackPressed() method in the relevant Activity. On XNA the buttons on the phone are considered to be buttons on a controller as if an xbox360 controller is in use. So to handle the back button it is just simply detecting if back is pressed on the controller for player one inside the update method. The default code actually provides interaction for this with the default of exit on the application as it would for any other XNA based project.

9 Development Environment

9.1 Installation complexity

Configuring the two development environments was relatively easy. The only real issue was that it took so long to install each of them. There were more manual selection of options and configuration steps to set up for developing on Android. Most of these were just following a guide though. Once installed for the development environment it was simple to install drivers for deploying to devices and applications worked on devices for Android. It was so simple that all the debugging and testing during development for Android was done directly on the device and the emulator was never used. For Windows after installing the SDK package there were a whole series of additional steps for account registration and installation of Zune. Even after all of that deploying to the device still failed for the Windows Phone from the installation and the emulator was used instead.

9.2 Development complexity

Eclipse is not preferable as an environment; a lot of the shortcuts that I tried to use were those from Netbeans. It was not difficult to use in general as most of the development was just writing the code into the text editor. In contrast the development with Visual Studio for the Windows Phone was native to most of the code I have been writing recently. So it was particularly easy to navigate for the Windows Phone development.

9.3 Debugging complexity

Debugging on the Android was reasonably easy. Developing by continually redeploying to the device was quite fast and the logs being fed back to LogCat were useful for determining sources of issues. As the emulator was not used I can't comment on how useful that particular feature was for debugging. The same is true for the inverse with the Windows Phone. I was unable to deploy to the phone, so I am unsure of the debugging capabilities it provides on the device. The emulator was particularly useful once I understood how to use it. Initially the accelerometer emulation seemed strange, but after a while of using it to test out the application it felt quite easy to use. As the core code was not being modified at all except for a conversion there wasn't really any need for logging of data in the output and the debugging was all visual. I assume that it would be similar to how console output would be shown in visual studio anyway.

10 Conclusion

A Windows Phone based version of the application was successfully converted from the Android version. There was not any particular difficulty is completing this task. Most of the general topics of elements that needed to be changed in the code have been identified in this document. In terms of general preference as a device I would rather be developing for Android as that is what I see myself continuing to use. As a development environment I would say that the Visual Studio development for Windows Phone with XNA is easier to work with. Particularly when having used XNA or other similar engines for games development.