CSCI 4237 Project Final Report

Gem Spinner

10th December 2011

Sharvani Tota

Introduction and Motivation

I am a gamer originally. I travel back home from campus daily in metro. One day when I was travelling back home during the last semester, I happen to see a lady playing game called GemSpinner enthusiastically in her Iphone. This game fascinated me very much. The next moment I bought the game on my Iphone and started playing. On the Day 1 of my Software Handheld Devices course, when professor asked us to think about our final project, the first thing that came into my mind was, "Why don't I try to develop this GemSpinner game in Android". This thought of mine led to development of this project.

Gem Spinner is a Match-3 Style game. The game board consists of different colored gems embedded into pieces of different Shapes. The player is allowed to swap the gems - only within a piece, long press a gem to rotate a piece 180 degrees. Once there is a match, matched gems disappear and new gems replace them from the top.

Start Up

As I started thinking how do I start development, my Project Proposal document came handy to me. As I stated in the proposal document, I started with developing the layout of my application. My first requirement was to design the shape of my gem. I decided i will use a circular shaped gem for my application. Then, I started researching for drawing shapes in android.

Developing a Gem Class

As I was going through the developer guide of developer.android.com, I realized that android.graphics.drawable library has ShapeDrawable class which can be used to draw different kind of geometric shapes which was mentioned in Project proposal. The code snippet below shows the construction of a gem Object.

```
public Gem(Context context, AttributeSet attrs) {
      super(context, attrs);
           int x = 10;
           int y = 10;
           int width = 30;
           int height = 30;
           gemColor = colorArray[randomColor];
        mDrawable = new ShapeDrawable(new OvalShape());
        mDrawable.getPaint().setColor(gemColor);
        mDrawable.setBounds(x, y, x + width, y + height);
      }
```

mDrawable is a ShapeDrawable Object which is used to render an oval Shape of the gem.getPaint() method is used to set the color of the gem. The colors of the gems are randomly generated using a Random object as shown below.

```
Random random = new Random();
int randomColor = random.nextInt((colorArray.length));
```

Gem Class extends the View class which is a basic building block of the UI components in android. The onDraw method renders the drawing of the Gem Shape.

```
protected void onDraw(Canvas canvas) {
           mDrawable.draw(canvas);
           }
```

Arranging Gems into Different Pieces

I have used the help of xml files to embed gems into a piece. I have embedded the gems into a Linear Layout and bordered the layout to render the shape of a piece. The code to arrange a piece shape with two gems vertically is shown below.

```
<merge xmlns:android="http://schemas.android.com/apk/res/android">
 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/layout2a"
    android:orientation="vertical"
    android:layout_width="31dp"
    android:layout_height="66dp"
    android:background="@drawable/my_border" >
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="31dp"
    android:layout_height="33dp">
      <game.gemspinner.Gem
```

```
        android:id="@+id/secondgem1a"
        android:layout_width="30dp"
        android:layout_height="30dp" />
</LinearLayout>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="31dp"
    android:layout_height="33dp">
     <game.gemspinner.Gem
       android:id="@+id/secondgem2a"
       android:layout_width="30dp"
       android:layout_height="30dp" />
</LinearLayout>
</LinearLayout>
</merge>
```

<game.gemspinner.Gem> tag of the above xml allows to embed a Gem into the linear layout. The above xml is stored in second.xml file under layout directory. The generated piece is as shown below.

As you can see the above xml consists of a <merge> tag. When we include this xml layout in the main layout, merge tag helps to reduce the number of levels in a view tree. It is used to complement the <include> tag which is used to include the above xml into the maib layout xml. Different shapes can be formed by arranging gems into different vertical and horizontal layouts.

Layout of the Game Board

I have arranged different pieces onto the game board using the relative layout which provides constructs for laying out the views relative to the position of the main layout. Different pieces arranged in Linear layouts using the merge tag are included into the main xml relative layout file using the <include> tag. The code for including the above second.xml piece using <include> tag into main relative layout file is as shown below.

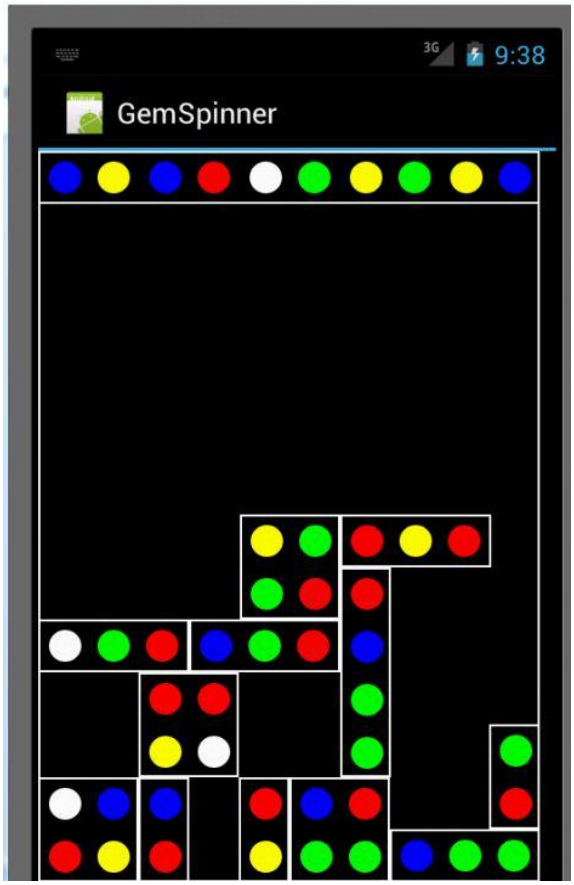
```
<?xml version="1.0"encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="310dp"
```

```
   android:layout_height="fill_parent"
   android:id="@+id/whole"
   android:background="@drawable/my_border">
<RelativeLayout android:id="@+id/relativesecond1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentBottom="true"
            android:layout_toRightOf="@+id/relativefirst1" >
    <include android:id="@+id/second1" layout="@layout/second"/>
</RelativeLayout>
</RelativeLayout>
```

This main xml file is included into the activity of the android in the following way.

```
    @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
     }
```

Final layout of the game board after including all different shapes into the layout is as shown below.



Swapping Gems

Next functionality is to swap the gems only within a piece. As explained in the project proposal, I used event handlers to know which two gems are being clicked. If the two gems selected are within the same shape, the colors of the two gems are swapped or else the swap does not take place. The code below shows the method for comparison.

```
public void compare(Gem g,View v) {
                if(Global.counter==1) {
                   a_x = v.getLeft();
                   a_y = v.getTop();
                   parentId1 =(LinearLayout)v.getParent();
                   grandParentId1=(LinearLayout)parentId1.getParent();
                   a=g;
                   Global.color1 = a.gemColor;
                }
                    if(Global.counter==2) {
                        b_x = v.getLeft();
                        b_y=v.getTop();
                        parentId2 =(LinearLayout)v.getParent();
                        grandParentId2=(LinearLayout)parentId2.getParent();
                        b=g;
                        Global.color2 = b.gemColor;
                        if (grandParentId1==grandParentId2) {
                        a.changeColor(Global.color2);
                        b.changeColor(Global.color1);
                        }
                        Global.counter=0;
                    }
}
```

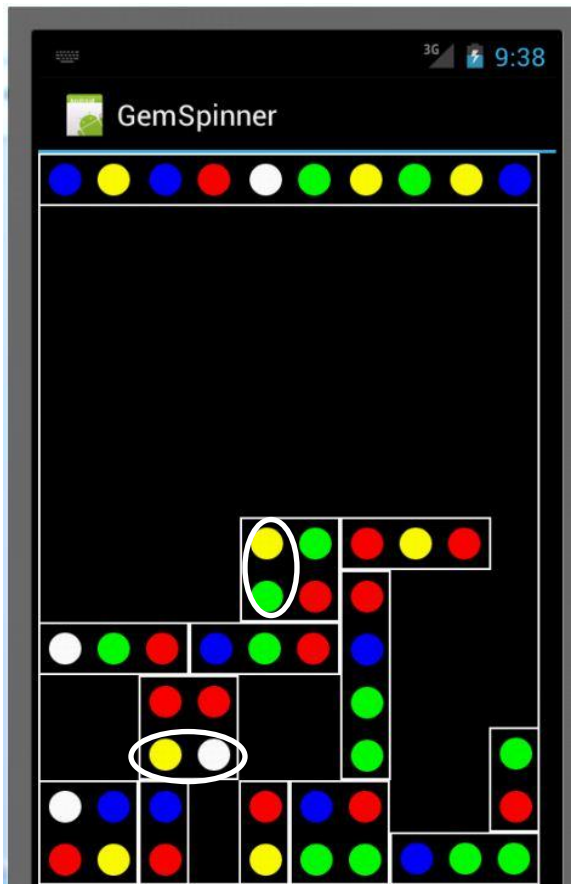
When the first gem is clicked, Global.counter is set to 1 and the first if condition is executed. When second gem is clicked, Global.counter is set to 2 and the second if condition is executed. The OnClickListerner of a gem is implemented as follows.

```
Gem.OnClickListener gemClick1a    = new Gem.OnClickListener(){
        public void onClick(View v) {
                Global.counter++;
                        compare(firstgem1a,v);
                }
        };
```

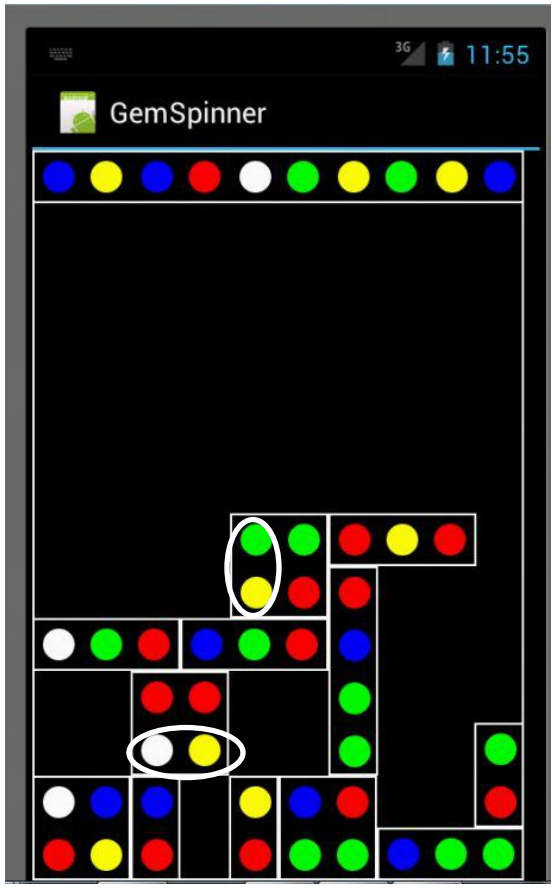
changeColor(int changedColor) method of the Gem class is invoked to swap the color of the two clicked gems.

```
protected void changeColor(int changedColor) {
                int x = 10;
                int y = 10;
                int width = 30;
                int height = 30;
                gemColor=changedColor;
                mDrawable.getPaint().setColor(gemColor);
                mDrawable.setBounds(x, y, x + width, y + height);
                invalidate();
        }
```

Before Swap:



After swap:



Piece Rotation

Then I have included rotation of a piece using the android.view.animation library. First the type of animation and its properties are well defined in the xml file under anim directory as shown below.

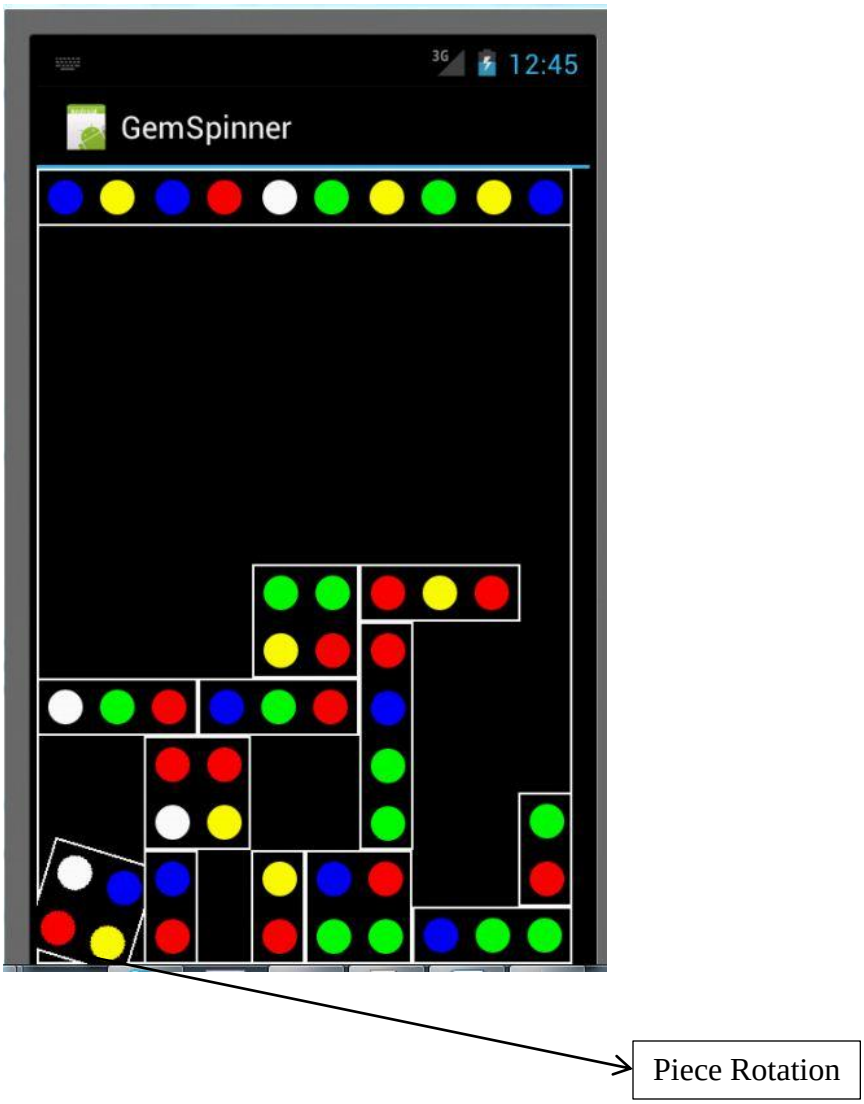
```
<?xml version="1.0" encoding="utf-8"?>
<rotate
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:fromDegrees="180"
    android:toDegrees="0"
    android:pivotX="50%"
    android:pivotY="50%"
    android:duration="3000" />
```

Then the animation is attached to the relative layouts of all the pieces on the board. Once the gem within a piece is long pressed, the parent relative layout of the particular gem is rotated by 180 degrees. This particular scenario as of now is implemented for only one gem as shown below. The OnLongClickListener of the first gem is configured to run animations when the gem is long pressed.

```
RelativeLayout tv;
tv = (RelativeLayout) findViewById(R.id.relativefirst1);
firstgem1a.setOnLongClickListener(new OnLongClickListener() {
            public boolean onLongClick(View v) {
                    RunAnimations();
           return true;
                    }
    });

private void RunAnimations() {
             an = AnimationUtils.loadAnimation(this, R.anim.rotate);
             an.reset();
             tv.clearAnimation();
             tv.startAnimation(an);
          }
```

The screenshot showing the rotation of the pieces when the first gem is long pressed is shown below.



Future Scope

I would like to implement the logic of replacing the gems with the gems of the top row when a match is established. I would like to replace gems of different colors with different gems of different shapes and colors. I would like to implement the drag and drop functionality to the pieces.