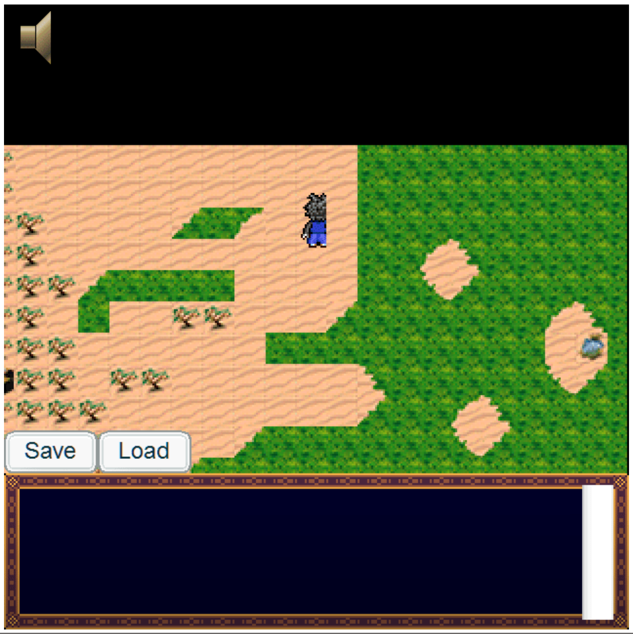
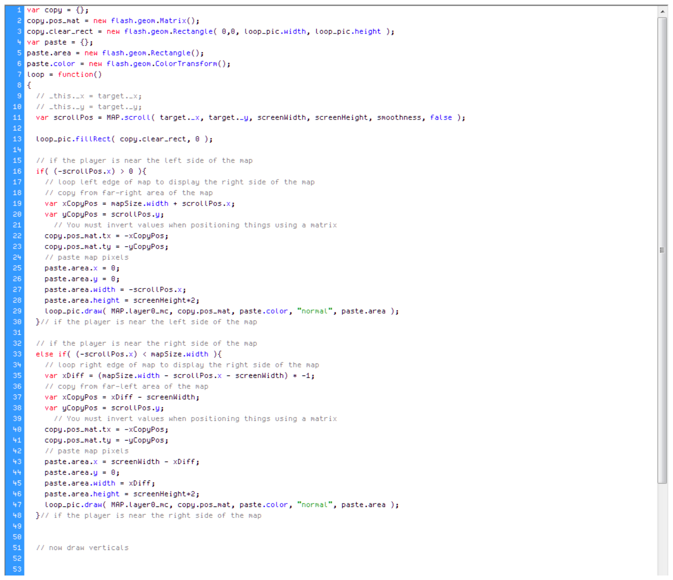
This article was originally posted on twitter as a series of posts while this project was still being made. I approach these blogs this way so people can see me experiment and discover things in-the-moment while I work on projects.

Attempt 1

I'm making looping maps in my game. I have some code that takes a snapshot of the other side of the map and draws it on the edge you're near. I've only added looping for the left and right edges so far. It's tricky to program because of all the matrix math.



It wouldn't be so bad if I didn't have to program separate looping code for all 4 directions and each of the 4 corners. I should try breaking down the math steps more to see if I can find common code I can re-use.



```
var copy = {};
copy.pos_mat = new flash.geom.Matrix();
copy.clear_rect = new flash.geom.Rectangle( 0,0, loop_pic.width, loop_pic.height );
var paste = {};
paste.area = new flash.geom.Rectangle();
paste.color = new flash.geom.ColorTransform();
loop = function()
{
  // _this._x = target._x;
  // _this._y = target._y;
  var scrollPos = MAP.scroll( target._x, target._y, screenWidth, screenHeight, smoothness, false );

  loop_pic.fillRect( copy.clear_rect, 0 );

  // if the player is near the left side of the map
  if( (-scrollPos.x) > 0 ){
    // loop left edge of map to display the right side of the map
    // copy from far-right area of the map
    var xCopyPos = mapSize.width + scrollPos.x;
    var yCopyPos = scrollPos.y;
      // You must invert values when positioning things using a matrix
    copy.pos_mat.tx = -xCopyPos;
    copy.pos_mat.ty = -yCopyPos;
    // paste map pixels
    paste.area.x = 0;
    paste.area.y = 0;
    paste.area.width = -scrollPos.x;
    paste.area.height = screenHeight*2;
    loop_pic.draw( MAP.layer0_mc, copy.pos_mat, paste.color, "normal", paste.area );
  }// if the player is near the left side of the map

  // if the player is near the right side of the map
  else if( (-scrollPos.x) < mapSize.width ){
    // loop right edge of map to display the right side of the map
    var xDiff = (mapSize.width - scrollPos.x - screenWidth) * -1;
    // copy from far-left area of the map
    var xCopyPos = xDiff - screenWidth;
    var yCopyPos = scrollPos.y;
      // You must invert values when positioning things using a matrix
    copy.pos_mat.tx = -xCopyPos;
    copy.pos_mat.ty = -yCopyPos;
    // paste map pixels
    paste.area.x = screenWidth - xDiff;
    paste.area.y = 0;
    paste.area.width = xDiff;
    paste.area.height = screenHeight*2;
    loop_pic.draw( MAP.layer0_mc, copy.pos_mat, paste.color, "normal", paste.area );
  }// if the player is near the right side of the map


  // now draw verticals


```

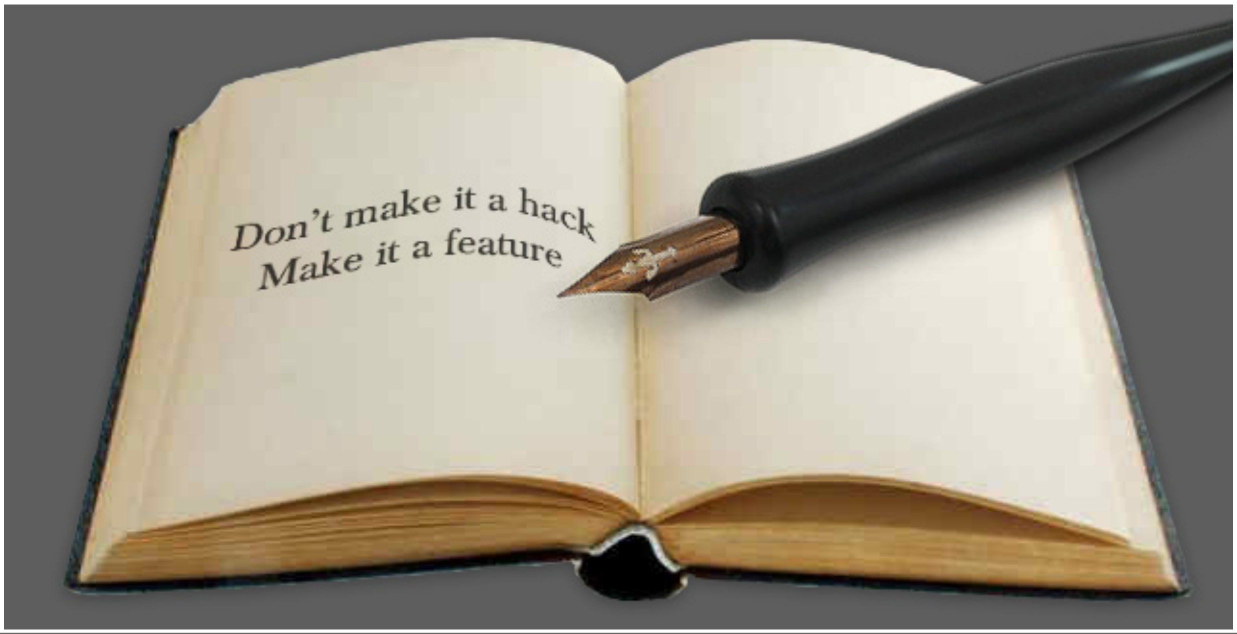
```
}// loop();
```

. . .

Let's Re-think This

I've decided to re-think my approach to looping map scrolling. It turns out I've already programmed something like this before when I made the 3D map system.

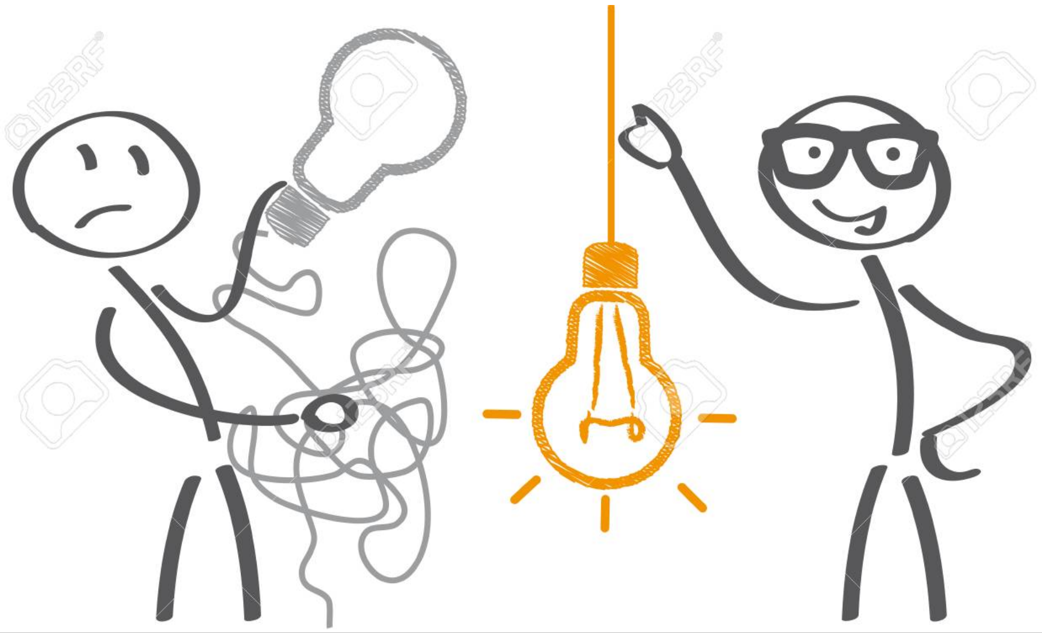
I have a rule of thumb:
"Don't make it a hack, make it a feature"



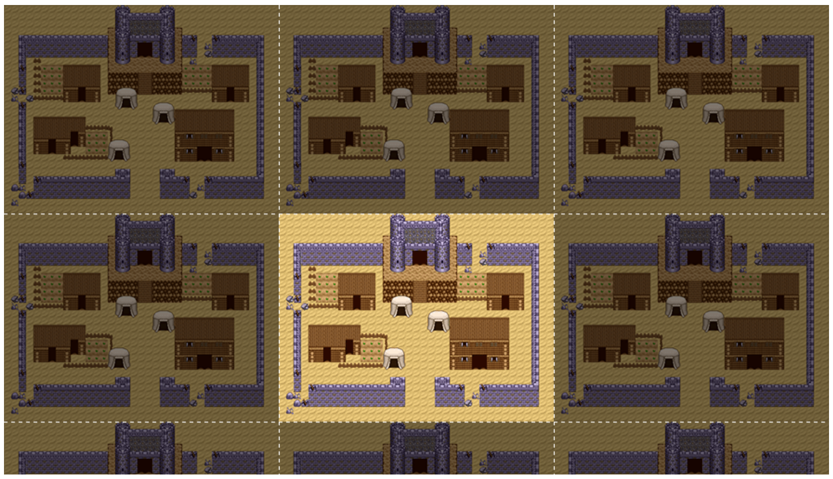
It means that when I change things in programming, I shouldn't alter something's behavior using outside code. I should just add it to the original code as a built-in feature. This might seem like feature-creep, but it means the new behavior isn't going to break when I change things later.

Attempt 2 (The Right Way)

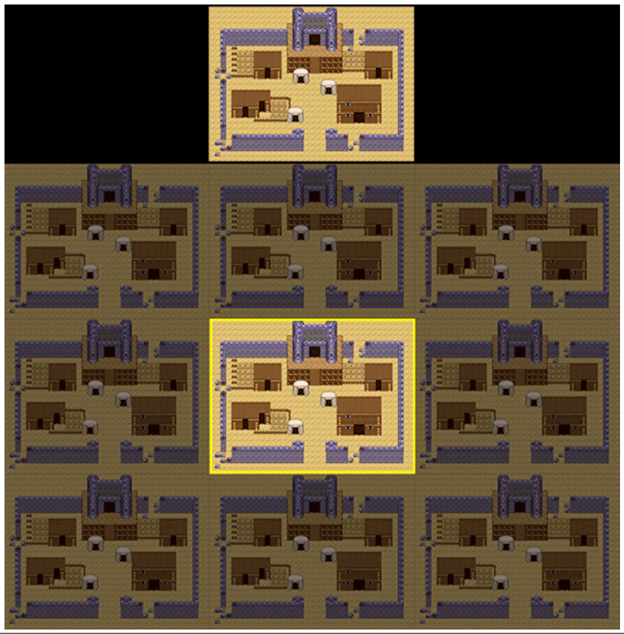
So instead of having some outside code making a bunch of draw() calls to paint additional screenshots of the map on top of the actual map system to simulate a looping scroll, it'll be better to have the map itself handle this, and it would actually be much simpler to do.



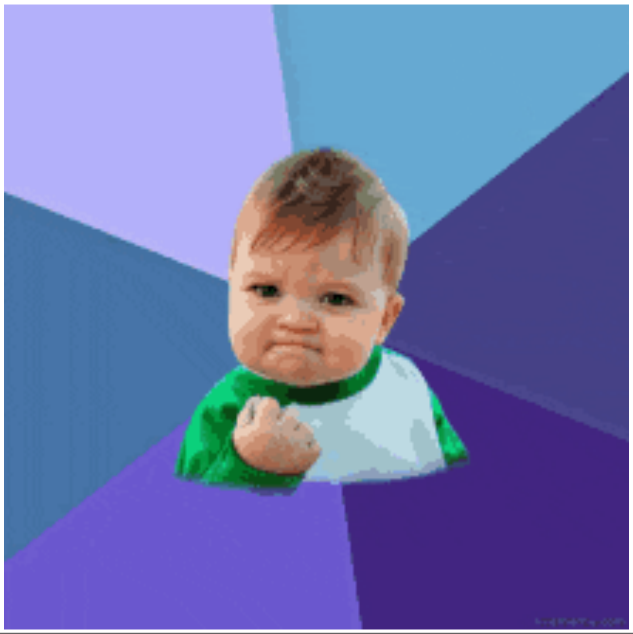
Just tell the map to re-display additional copies of its existing bitmaps on all of its corners. If I use the cacheAsBitmap flag this won't slow down render performance because Flash is just referencing the same data behind-the-scenes. I could display the same picture a thousand times and it wouldn't matter.



And I can continue using scrollRect for fast scrolling the same way I already do instead of making additional draw() calls, so the performance will stay exactly the same.



This new approach to looping scrolling is much simpler to do, performs much faster, and won't be a fragile hack. Win-win-win!



The Result

Aaaaaand... done! The only tricky part was changing the collision system to sometimes let the player walk outside the map. The scroll system instantly moves the player to the other side of the map and then tells the map to scroll to that spot. The effect is seamless.