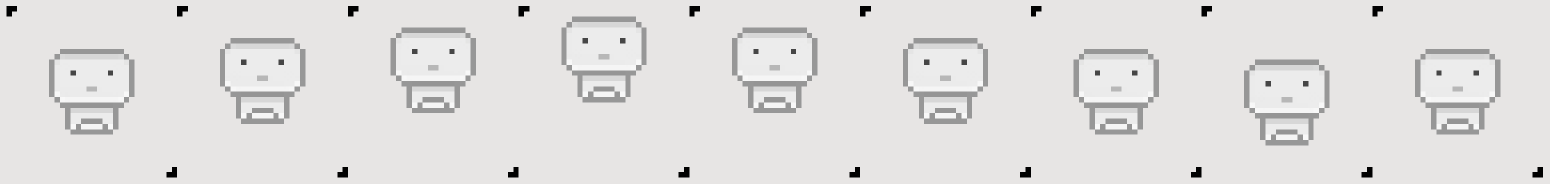
THE SPRITE PROJECT

SPRITE

‘Sprite’ is a term which is usually used to describe 2D art. There are different types of sprites - some are transparent, others are animated. This engine tries to cater for most type of sprites. The Sprite Project aims to facilitate development of games which use sprites without using shaders.

STRUCTURE

A Sprite Engine governs and controls a number of Sprite Libraries - collections of Sprites. while Sprites consist of a single sprite sheet (image containing a number of sprites with different frames for animations) along with a number of attributes which affect how a Sprite behaves. The Sprite Engine is responsible of going through each and every Library and its Sprites and checking whether any updates to the animation are necessary. Having multiple Sprite Libraries would let you remove a certain type of Sprites easily and without affecting other Sprites.



Certain variables, such as the width of a sprite ,are calculated automatically. Take the above sprite as an example. There are 9 frames. Each frame has a width and height of 32px. In total, the sprite sheet has a width of 288px. Therefore when passing the number of frames, the engine checks the width and takes care of the rest for you.

HOW TO USE

To create a sprite, three objects must be created. Firstly, a sprite... To create a Sprite, the following parameters must be passed:

String imageLocation:	relative path to the sprite image
String name:	sprite's name
AssetManager assetManager:	the game's Asset Manager
boolean animated:	'true' if the sprite is animated, 'false' if it isn't
boolean transparent:	'true' if the sprite has transparent parts, 'false' if it is opaque
int frames:	number of frames one row in the spritesheet has
int rows:	number of rows in the sprite sheet (currently the engine only displays the first row)
float timeSeparation:	the time (in seconds) separating each frame
String onEnd:	what to do with the sprite when its animation finishes. Valid values:
Loop:	the animation restarts from the beginning
NoLoop:	the animation stops upon reaching its end. To replay animation, the setPaused method is called, with 'false' as a parameter
Reverse:	the animation keeps reversing whenever the last or first frame is reached
Scroll:	the sprite side-scrolls through all the frames. When it reaches the end, it restarts from the beginning
String onResume:	what to do with the sprite once it resumes after being paused. Valid values:
Continue:	the animation continues from where it left off
Start:	the animation restarts from the beginning

Next up, we need to create a Sprite Library. To create one, the following parameters must be passed:

String name:	the library's name
boolean staticLibrary:	a boolean indicating whether the library contains any animated sprites. Set to true if there are none

Lastly, create a Sprite Engine. One Sprite Engine is enough in one project since its job is simply to update sprites and store sprite libraries. Creating the engine couldn't be simpler:

```
SpriteEngine engine = new SpriteEngine();
```

The node onto which the sprites will be attached is then sent to the Library class to be shared with all libraries. Normally, the node provided would be the guiNode. When this is done, the Sprite you created is added to the library:

```
SpriteLibrary.l_guiNode = guiNode;
library.addSprite(sprite);
```

The next step would be to add the library to the engine:

```
engine.addLibrary(library);
```

The last step is to call the update method in the simpleUpdate. The perimiter you should provide is 'tpf' - a float which is used to compute animations internally:

```
      engine.update(tpf);
```

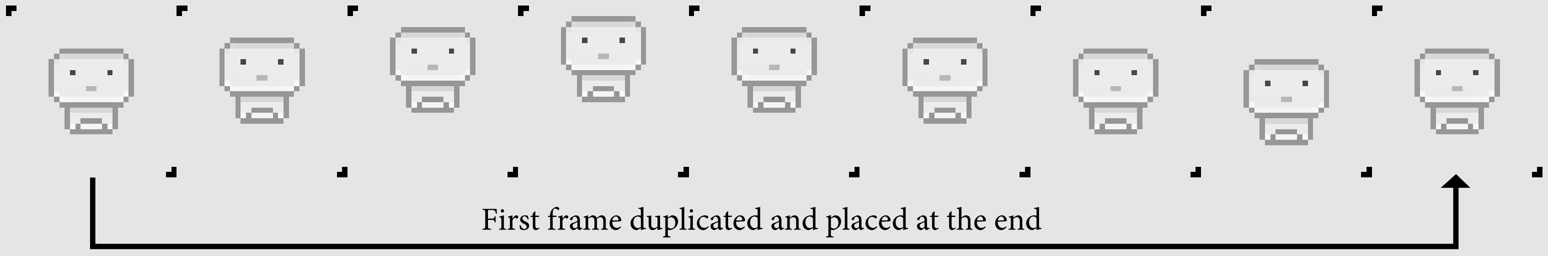
A sample program would look like this:

```
package mygame;
import com.jme3.app.SimpleApplication;

public class Main extends SimpleApplication {

      static Main app;
      public static void main(String[] args) {
            app = new Main();
            app.start();
      }

      static SpriteEngine engine = new SpriteEngine();
      @Override

      public void simpleInitApp() {
            Sprite sprite = new Sprite("Textures/Sprite.png", "Sprite 1", assetManager,
                                    true, true, 9, 1, 0.15f, "Loop", "Start");
            SpriteLibrary.l_guiNode = guiNode;
            SpriteLibrary library = new SpriteLibrary("Library 1", false);
            library.addSprite(sprite);
            engine.addLibrary(library);
      }

      @Override
      public void simpleUpdate(float tpf) {
            engine.update(tpf);
      }
}
```

Easy, huh? There are a number of other methods included each class which lets you change the previously-mentioned parameters. These include pausing or resuming a sprite's animation, adding or removing a sprite from a library, and adding or removing a library from the engine.

ADDENDUM - SCROLLING

When working with scrolling sprites, there is just one small change from the normal procedures. As can be seen in the following sprite sheet, the first frame is duplicated and its copy is placed in the last frame. This ensures a smooth scrolling animation. Note that when providing the number of frames in a scrolling sprite, the number represents by how much the sprite's width will be divided. For example a sprite sheet of width 288px divided by 9 would yield a sprite having a width of 32px.



OTHER FEATURES

There are also some other methods included in the Sprite class which lets you manipulate sprites. These include:

- move(int x, int y) - displace the sprite by a number of pixels as provided in the parameters

- moveAbsolute(int x, int y) - position the sprite at the coordinates provided in the parameters

- setOrder(int z) - increase the sprite's rendering order by the value provided

- setOrderAbsolute(int z) - change the sprite's rendering order to the value provided

- rotate(int x, int y, int z) - rotate the sprite by the angles (in Degrees) provided in the parameters

NYPHOON