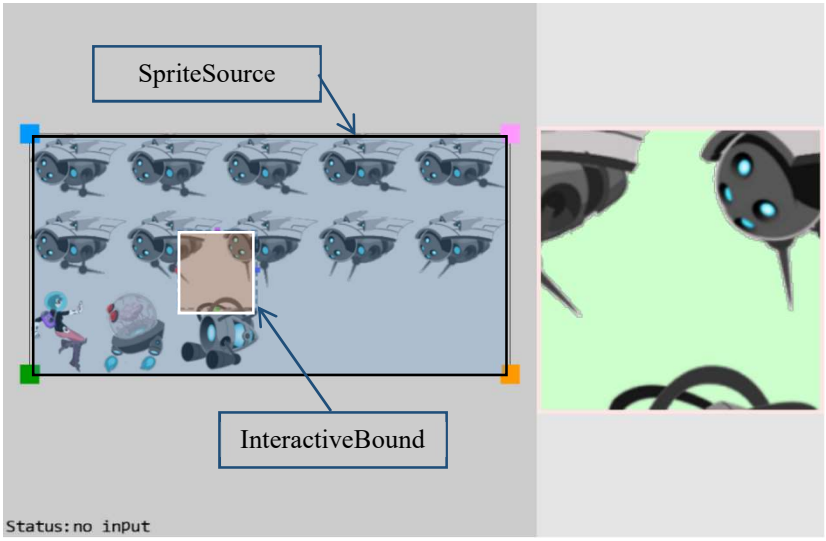
CSS 452: Programming Assignment #4
Working with Textures and Sprites

Due time: Please refer to our course web-site

Objective
In this programming assignment we will develop a mock sprite animation identification tool. In the process, we will review concepts relating to and practice programming with object oriented design, geometric bounds, and texture UV coordinates.

Assignment Specification:
Here is an example of the results from this assignment, there are two objects and two views:

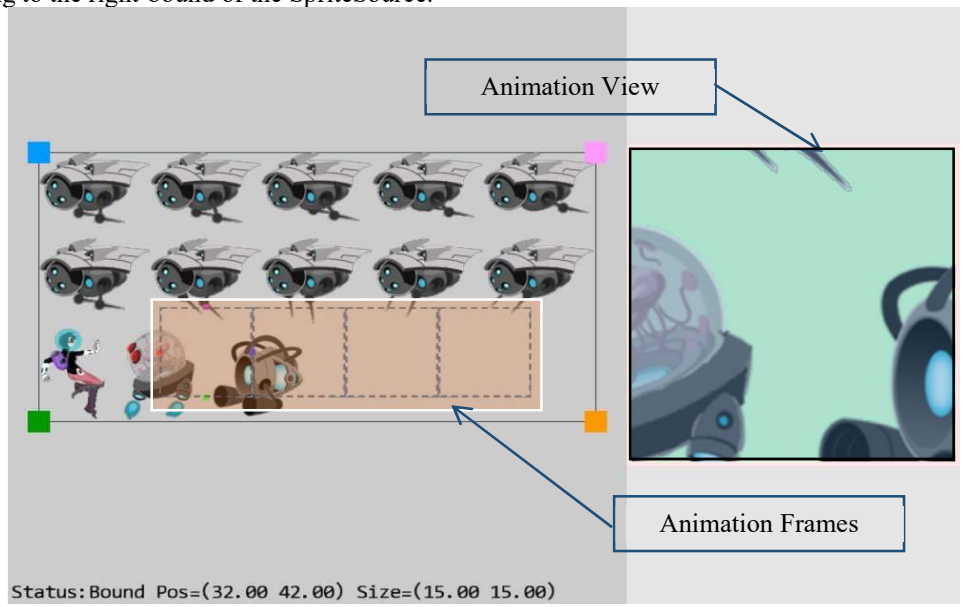
The Two Objects: the two objects are the **SpriteSource** and the **IntertactiveBound** as illustrated in the following figure.



- **SpriteSource**: This is a texture image typically with sprite elements. Notice that:
    - **Color corner markers**: Each corner of this object should display a unique color square (any visible color).
    - **Bounded sides**: Each sides of this object should be bounded by a visible, yet thin, bounding lines (in a dark color).
    - **Always textured**: Please refer to the asset.zip file in the assignment folder for the assets used. You must support the at least two different images as your sprite source. In my implementation, I used *minion_sprinte.png* and *Consolas-72.png* (shown in the following screen shot).

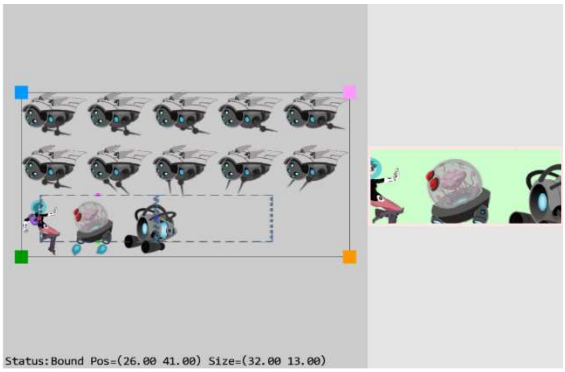
- **Switching of the SpriteSource image:** at run time your user must be able to enter "**Z**" and "**X**" keys to dynamically show at least two different sprite source images.

- **InteractiveBound**: This is a textured image with clear boundary indications and transparent throughout. Please refer to the assets.zip file this is a textured renderable using Bound.png as the texture. Make sure you **do not** draw the dotted lines as individual Renderables. Notice that:
  - **Bound markers**: Located at the middle of each bound is a colored square marker (e.g., middle of the left bound is a red square).
  - **Behaviors**:
    - **Manipulations:**
      - **WASD** keys: to move this object;
      - **Arrow keys**: to change the size of this object (left/right to increase/decrease width, up/down to increase/decrease height).
    - **Refined Manipulations:** 1% of the above effect when the space-bar is pressed.
    - **Bounded**: this object should never be moved/scaled beyond the bounds **SpriteSource** object.
    - **Animation Frames**: The Q-key shows the rest of the frames that will be part of the animation as illustrated in the following screenshot. As indicated, these are same-size **InteractiveBound** objects to the right that can be fitted into the **SpriteSource** object. Note, the number of frames shown here (5 frames) are results of the size of the SpriteSource, your implementation must support frames extending to the right-bound of the SpriteSource.



The Animation View: As illustrated in the above screen shot, the Animation View is the continuous and instantaneous view of the content of the Animation Frames.

- **Animation View**: This is the view located at the top-left corner. This view continuously sequence through all of the current *Animation Frames*, from left towards right, at a rate of one element per second.
- **Aspect ratio of the viewport:** Aspect ratio of the viewport is always proportional to that of the InteractiveBound, such that content shown via the view is always proportional. This is illustrated in the following two screen shots. The left shows a talk and narrow and the right shows a short and wide InteractiveBounds. Notice in both cases, the content displayed in the Animation View is proportional.



- **Size and Position of the viewport:** As indicated in the above example, the Viewport size must either cover the entire width or entire height of the given canvas area, **additionally**, the viewport must located in the middle of the given canvas area.

In the real-world, the above tool will output the UV (or pixel) values that define a sprite animation.

Hints:

1. My implementation is based on book Example-5.4 (*FontSupport*).

2. Implementation approach, this is a large and complex project that requires planning! This is the approach I took:
    a. Define the two game objects and the two views as separate JavaScript objects.
    b. All particular and behaviors are hidden in respective objects.
    c. These are the steps I followed, note that I did extensive testing after each steps to ensure correctness. You must tightly integrate testing into your development process!
        i. Step 1: The **MainView**. My **MainView** is a subclass of the **Scene** class. This is my interface to the game engine. My architecture is such that both of the game objects (**SpriteSource** and **InteractiveBound**) will be defined here.
        ii. Step 2: I defined a **mSpriteSource** as a **SpriteSource** object where the **MainView::draw()** is a simple call to **mSpriteSource.draw()**. I hide all the markers and border lines inside **SpriteSource** object, and make sure the drawing works correctly.
        iii. Step 3: I worked with the **InteractiveObject**. Once again, **MainView** defines **mInteractiveObject** as an instance of the **InteractiveObject**. All detailed definition of boarder markers is hidden inside the **InteractiveObject** such that the **MainView::draw()** can now include a simple call to **mInteractiveObject.draw()** to draw the object. An **update()** method is defined to accomplish all user manipulations, and this method is called from **MainView::update()**.
        iv. Step 4: Define the **AnimationView** class to include a camera that focuses on an instance of **SpriteAnimateRenderable**. The **SpriteAnimateRenderable** refers to the same sprite sheet texture as the **MainView::mSpriteSoruce** object. When the **MainView::mInteractiveObject** is manipulated (moved or scaled), the **SpriteAnimateRenderable::setSpriteSequence()** function is called to update the sprite animation sequence. The **SpriteAnimateRenderable** object simply continuously updates its animation.

    The above is a very rough description of my design. There are much details missing, e.g., how does **InteractiveObject** bound itself to within a **SpriteSource** bounds. Please do feel free to design and implement your system in whatever way you feel comfortable. The important thing is to start NOW, and to implement and test one small module at a time. My rough estimation is that this assignment will take 2 to 5 times the amount of time to implement when compared to the previous MPs.

3. **Important**: only call the **SpriteAnimateRenderable::setSpriteSequence()** function to update the sprite animation when required. Remember that this function call resets a sprite animation, if you call it at every update, the sprite animation will be constantly reset and you will not observe any animations.

4. To format floats, I call the **x.toPrecision()** function when turning a number into a string, e.g., if x is number (integer or float) with value of 33, **x.toPrecision(4)** will return this string: "33.00".

5. Modify the game engine. I modified the game engine whenever I need additional functionality. E.g.,
    a. Defined: **Camera::getWCWidth()** to access the width of a camera WC window.
    b. Defined: **SpriteAnimateRenderable::setSpriteSequenceUV()** to support specifying sprite animation sequence via UV values instead of pixels.

Please do start early, this assignment is challenging and demanding!

Credit Distribution

Here is how the credits are distributed in this assignment:

1.	SpriteSource object		30%
	a. Displays corner markers	10%	
	b. Displays boundary lines	10%	
	c. Proper aspect ratio (image not squashed)	10%	
	d. **Z** and **X** keys to load two different images	10%	
2.	InteractiveBound object		30%
	a. Proper texture (bounds and transparent throughout)	5%	
	b. **WASD** and **Arrow** keys to move/scale	10%	
	c. Refined manipulation at 1% with space bar	10%	
	d. "**Q**" key showing animation frames	10%	
	e. Manipulation bounded to SpriteSource	10%	
3.	Main View:		5%
	a. Proper display of the two objects	5%	
	b. Proper status display	5%	
4.	Animation View		30%
	a. Focus view on one sprite animation	10%	
	b. Animation rate: 1 frame per second	10%	
	c. Always viewing current Animation Frames	10%	
	d. Always proper aspect ratio	10%	
	e. Width or height covers the given canvas area	10%	
	f. Always located in the center of the given canvas area		
5.	Proper submission		5%

	a. Zip file names with **NO SPACES** b. No extra unused files/folders (E.g., Test folder) c. Styles (project name, variable names, etc.)	5% 5% 5%	

This programming assignment will count 16% towards your final grade for this class.

Creativity and Extra Credits: This is a tool, so it may be slightly more challenging to insert your own marks. But, please do try, e.g., setup your views differently than mine, use better color coordination, better web-page construction, etc.