Entity Component System:

To comment on this doc: https://docs.google.com/document/d/1TWZyl9Nr5ii92jd6bw1VOE8NC7DGHQZwGXGCsQo5DE0/edit?usp=sharing

BerylEntity:

Entity is a concrete class that wraps the components together.

Should have the following:

* Scene - scene
* String - name
* int - id
* Transform - transform
* Set<BerylRC> - renderComponents
* Set<BerylUC> - updateComponents

BerylComponent

The abstract superclass of BerylUC and BerylRC which allows the two classes to share code involving the interaction with the parent Entity. Every subclass will have to supply an Entity to attach this to. If the entity is not null then this component will be added to it. If the entity given has a scene set, then this component will be added to the scene’s respective map of BerylUCs and BerylRCs.

BerylUC

An abstract subclass of BerylComponent, held within the Entity and Scene, and updated through the Scene.

Implements Updatable and thus requires the implementation of:

* onInit()
* onUpdate()
* onLateUpdate()

In every concrete subclass of this class.

Implemented UCs:

* Physics
* Collider (abs.)
* BoxCollider
* SphereCollider

BerylRC

An abstract subclass of BerylComponent, held within the Entity and Scene. At the end of the frame after all of the late updates have been processed, the rendering process begins. Any active BerylRC will be read and evaluated in the MasterRenderer. GUIs will be batched and processed after the 3D elements.

Supported RCs:

* Light (abs.)
  + PointLight
  + DirectionalLight
* Camera
* Skybox
* Mesh3
* Mesh2
* Transform - 2/3
* PostProcessingEffect (abs.)
  + ContrastEffect
  + HorizontalGaussianBlur
  + VerticalGaussianBlur
  + GaussianBlur
  + CompoundGaussianBlur
  + BloomEffect
  + BrightnessEffect

Scene & Application System:

The scene, renderer, and application classes will handle the actual implementation of the game loop and rendering. The application will call updates in the scene for the game logic, then pass the data held within the scene to the renderer in order to display it. The two systems (game logic vs rendering) should be separated into different threads.

The Application should implement Updatable and Playable

GUI Library:

NavigationSystem:

The ability to transition between NavigationSets, which specify certain Mesh2s to render. The NavigationSystem also incorporates the ability to play animations as it transitions between NavigationSets.

Mesh2RC:

The most basic gui element. It extends the BerylRC class as another abstract class to extend in order to render 2D elements.

Able to store the pos/scale as

* Pixels from TL of screen
* Pixels from TargetPoint of the parent
* Percent from TL of screen
* Percent from TargetPoint of the parent
* Screen space coordinates

To calculate the screen space coords, must take into account the parent and the offsets defined as the pos. Should also be able to choose a TargetPoint on the GUIElement to get the screen position of.

To render, need to return a transformation matrix and material for the GUIShader. Text will also be implemented and integrated into the GUI Library, so we will need to handle the text children with a bit of a different render process. Text will return itself to the render system as it is required to Lookup the ModelData and VAO associated with the string.

The render method will have to recurse down the children rendering the element itself, then calling the render method for the text children and then the GUIElement children.

Rect:

The most basic implementation of the GUIElement with no extra functionality.

Button:

Essentially the same as a Rect, but should be used in lieu of the rect if an onClick method is defined.

FullScreenRect:

A simple Rect extension that instantiates itself as covering the entire background.

CustomTargetRect:

Overrides the contains method and uses the contains method of a PlaceholderRect to redefine the targetability of this GUIElement.

PlaceholderRect:

Is not rendered, but still allows for a position and scale to hold children.

Slider:

A slider with a cursor that moves as the mouse drags it. The slider defines bounds to be 0 and 1 with linear interpolation, but each can be overridden. There is also support to add ticking which ticks the slider from one point to the next as the bar has been divided into.

Extra functionality has been added to support text above the slider, text below the cursor displaying the value, and text on each side of the bar with the min and max bounds.

GridBox:

Comprised of a VBox with HBox elements to create a grid. The GridBox also allows the definition of a min/max scale of itself and per element it contains.

HBox:

A box that holds multiple elements and grows horizontally.

VBox:

A box that holds multiple elements and grows vertically.

ProgressBar:

A simple Rect with a child Rect targeting from the CL and scaling with the value set. Text above, and below or on the right may be nice to visualize what the bar is for, or what the exact value is in the progress bar currently.

ParentPosControl:

A targetable Rect that tracks the mouse’s position and offsets the parent’s position accordingly. It can also reorder the parent to the front of its parent’s children list.

ResizableRect:

A Rect that has a PlaceholderRect that defines the bottom left corner that drags the Rect larger. Problems will arise with simply increasing the scale, because the position may not be from the top left, so there must be some way to translate the TargetPoint that this GUIElement is created by to keep the same TL corner.

Extras:

TargetPoint:

* TL, TC, TR
* CL, CC, CR
* BL, BC, BR

Notes:

* Want to develop a menu navigation system, or simply an animation navigation system so calls to it can play groups of animations and such. A current example is the GUI BerylUC in the game.components package.

Rendering Patterns:

The goal with creating a rendering pattern is to allow for custom shaders. The rendering path is as follows:

* The Skybox is rendered.
* MasterRenderer iterates over Mesh3s and batches Mesh3s into Shader -> RawModel -> Mesh3.
* The MasterRenderer then iterates over the shaders, binds the shader, and loads any BerylRC specified. It then loads each unique RawModel and iterates through the Mesh3s drawing each. (Consider creating instances using a ModelView Vertex Buffer Object)

Animation System:

Editor System:

Input System:

Extra Systems:

Mesh Editing:

Rigging:

Mesh Animation:

Physics System:

Collision System: