Steel Engine 1.0 Documentation

Contents

Components

A component is a script containing a class that inherits from the 'component' class or a 'component' subclass. These classes can be instantiated and applied to any **GameObject**.

Functions:

Every component has a '**Tick**' function that must be implemented. This function provides one float parameter. The value of this parameter is always the **DeltaTime** (time between the last and current frame).

Every component also has a '**Init**' function. This function is protected and virtual. This means that only classes that are a subclass of the component class can access this function and that it *can* be overridden but it isn't required. This function gets called when the component is **initialised**. Do not get this confused with the component being **instantiated**.

GameObjects

A GameObject in Steel Engine is pretty much the same thing as a GameObject in Unity. It is an object in the scene and can have the following properties:

- Position (vec3)
- eRotation (vec3)
- qRotation (quaternion)
- Scale (vec3)
- RenderShader (RenderShader)
- ID (int)
- Name (string)
- Components (List<Component>)
- Parent (GameObject)

The **eRotation** is the rotation of the object in Euler angles.

The **qRotation** is the rotation of the object as a quaternion. Of course, the eRotation is prone to gimble lock so qRotation should be used when possible.

The **RenderShader** is a variable of type RenderShader specifying the type of shader to use when rendering the GameObject.

The **Parent** of the GameObject is not implemented as of this version but it will in future make the **Position**, **Rotation** and **Scale** of the GameObject relative to its Parent.

Functions:

- Void AddComponent(Component component)
- T GetComponent<T>() where T : class
- Void Tick(float DeltaTime)
- Static GameObject Instantiate(GameObject Original)
- Static GameObject QuickCopy(GameObject Original)
- Matrix4 GetModelMatrix()
- Void Rotate(Vector3 rotation)
- Void Rotate(Float x, Float y, Float z)
- Void SetRotation(Vector3 rotation)

- Void SetRotation(Quaternion rotation)
- Void LoadTexture(String name, String extension)
- Void LoadTexture(String path)
- Void LoadTexture(Texture texture)
- Void Load()
- Void Render()

The **AddComponent** function appends the given component instance to the GameObjects component list and calls the **component's Init** function. This should be used only in runtime otherwise the component maybe initialised twice.

The **GetComponent** function is a generic function that takes a component type and finds the first component instance of the specified type on the GameObject. If no instances are found, null is returned.

The **Tick** function is called every frame by the **SceneManager** if the **InfoManager's GameRunning** variable is true. It calls every **Component's Tick**

The **Instantiate** function takes a GameObject as an original and creates a new GameObject instance with the same properties. This function is very slow so it should not be used often.

The **QuickCopy** function is a sort of faster version of the instantiate function with one drawback. This being that it's simply creating a new reference type and not value type variable. This means that it's pretty much just a pointer to the original GameObject provided, rather than a fresh space in memory with the same values. But it is much **faster** than the **instantiate** function.

The **GetModelMatrix** function returns a Matrix4 representing the transformation of the GameObject (the model matrix).

The **Rotate** function has 1 overload. The base function takes a Vector3 value representing the desired rotation in Euler angles and applies it to **eRotation** and **qRotation** of the GameObject. The overload takes 3 floating point values representing the Euler angles X, Y and Z values and applies them to the eRotation and qRotation of the GameObject.

The **SetRotation** function has 1 overload. Both functions set the eRotation and qRotation values of the GameObject, but the base function takes the rotation as a Vector3 (Euler angles) rotation, and the overload takes the rotation as a quaternion rotation.

The **LoadTexture** function has 2 overloads. They all apply a texture to the GameObject. The base function takes the desired textures name and file extension, and assumes it exists in the Resources/Textures folder. The first overload takes a path to the texture including the extension. The second overload takes a **Texture**.

The **Load** function sets up the GameObjects shaders, vertices, and indices along with calling every attached component's Init function.

The **Render** function renders the GameObject to the viewport.

Scenes

A **scene** contains information about the **GameObjects**, **LightObjects**, and **Cameras** in the scene. These are all stored in three separate lists that are public.

A scene also has two more properties being the **Name** and the **Starting Camera ID**. These properties are self explanatory.

Functions:

- Void Load()
- Void AddLight(Vector3 position, Vector3 colour, float intensity)

The **Load** function is used to replace all GameObjects, LightObjects, and Cameras in the **SceneManager** with the GameObjects, LightObjects, and Cameras in the scene.

The **AddLight** function is used to add a LightObject to the scene. It takes all of the parameters used to create a LightObject and is not the most useful function, but it exists, nevertheless.

SceneManager

The SceneManager is a static class that stores and handles all information around scenes.

The SceneManager public properties:

- Scenes (List<Scene>)
- GameObjects (List<GameObject>)
- Cameras (List<Camera>)
- GameRunning (Boolean)
- GameTick (event – void GameTick(float DeltaTime))

Functions:

- Scene GetActiveScene()
- void LoadScene(int BuildIndex)
- GameObject GetGameObjectByID(int ID)
- Scene ConstructScene(string[] Lines)
- void Init()
- void Tick(double DeltaTime)
- SteelRay CalculateRay(Vector2 MousePosition)

GetActiveScene returns the currently loaded scene.

LoadScene uses the BuildIndex to load a scene.

GetGameObjectByID returns a loaded GameObject which matches the given **ID**.

ConstructScene creates a scene by using the lines of a scene file (.SES).

Init creates all the scenes by using the scene files in the Resources/Scenes folder.

Tick invokes the **GameTick** event when the **InfoManager's GameRunning** property is true.

CalculateRay creates a **SteelRay** from the mouse's screen position forwards from the camera into the world.

RenderShader

The RenderShader type is an Enum containing the following shader types:

- ShadeFlat
- ShadeLighting
- ShadeTextureUnit

The ShadeFlat shader is a simple shader that renders the object using just the colour of each vertex. This means that it wont react to lighting and doesn't support textures.

The ShadeLighting shader is in the current version slightly broken, but it will be improved as time goes on. This shader attempts to change the brightness of each vertex based off of the surrounding lights in the scene. It also only supports one light source in this version.

The ShadeTextureUnit shader is also a simple shader that uses the **Mesh's** UVs to determine how to display a texture onto the GameObject. This doesn't react to lighting and doesn't support vertex colours.