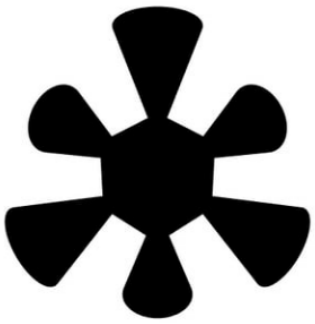

RUNTIME EDITOR

V1.3.1 04.05.2017

Table of contents

Introduction

Runtime Editor is a set of scripts, which will help you to implement runtime scene/level editor. Package divided into several parts, could be used together or independently: RTEditor, RTHandles, RTGizmos, RTSaveLoad, UIControls.

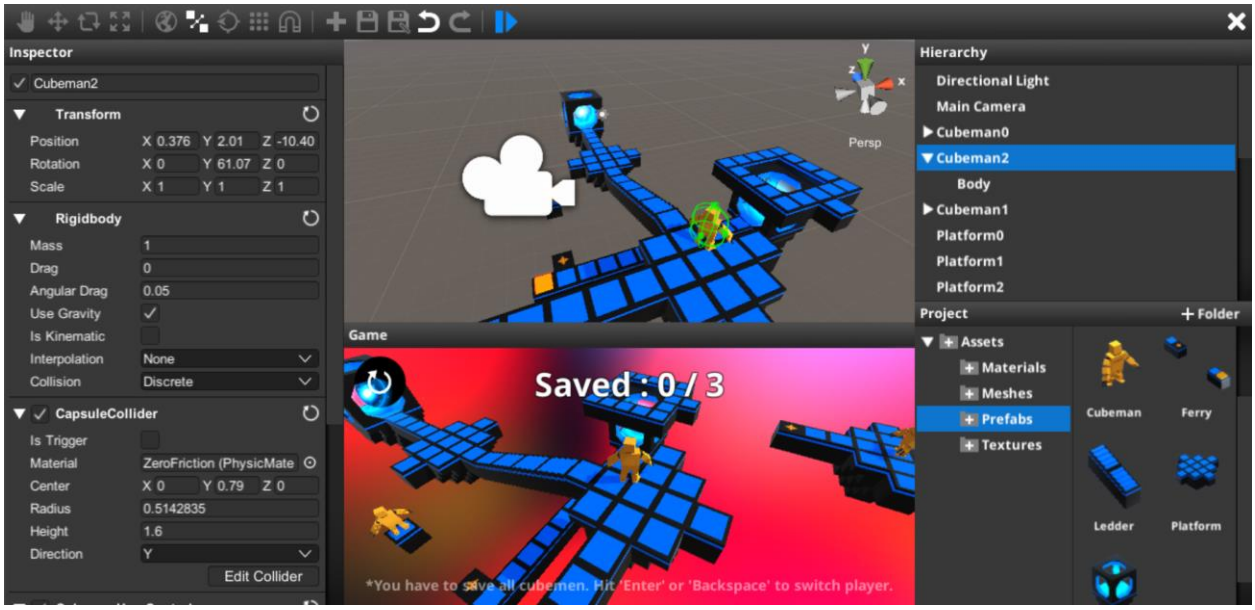

Fig.1 - Runtime Editor

Features

- Positon, Rotation, Scale handles, Grid, Box Selection, Scene gizmo
- Gizmos for Colliders, Lights and Audio Sources'
- Global & Local coordinates, Local & Center pivot point modes,
- Scene view navigation;
- Snapping (Grid, Ground, Unit, BoundingBox)
- Orthographic & Perspective view;
- Game View
- Play & Edit mode;
- Undo & Redo;
- Hierarchy Window, Project Window, Inspector Window
- Component & Material Editors, Property Editors;
- Ability to edit various component types including custom scripts,
- Asset bundles support & ability to create dynamic assets.

Package Structure

Runtime Editor located in **Assets/Battlehub/RTEditor folder,**

Transform Handles in **Assets/Battlehub/RTHandles,**

Gizmos for Colliders, Lights and AudioSources in **Assets/Battlehub/RTGizmos**

Save & Load subsystem in **Assets/Battlehub/SaveLoad,**

TreeView and ListBox in **Assets/Battlehub/UIControls,**

Common classes located in **Asset/Battlehub/RTCommon**,

Helper classes in **Assets/Battehub/Utils,**

Demo scene in **Assets/Battlehub/RTEditor/Demo**

Each folder organized as following:

/Scripts – for runtime scripts

/Scripts/Editor for editor scripts

/Prefabs for prefabs

/Shader/Resources for shaders

/Demo (if present) contains everything related to demoscene

1. Menu

1.1 Runtime Editor Menu

Runtime editor menu is very simple. There are four menu items:

1. **Create menu item** creates runtime editor using prefab located in Battlehub/RTEditor/Prefabs/RuntimeEditor.prefab
2. **Expose to Editor** can be used make game object, prefab or resource (material, texture, etc.) visible to editor
3. **Hide from Editor** can be used to hide game object, prefab or resource from editor
4. **Configuration** opens configuration window.

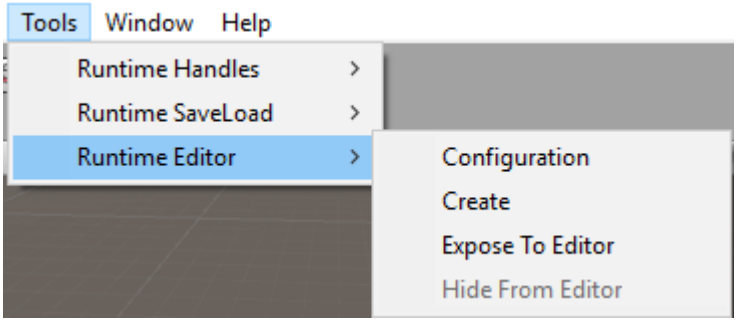

Fig 1.1 Runtime Editor Menu

1.2 Runtime Save&Load Menu

Runtime Save&Load menu allows you to configure Save&Load subsystem. There are following menu items:

1) **Build Resource Map** creates or updates mapping between objects (prefabs, resources, special scene objects) and unique identifiers. These identifiers are required to make Save&Load subsystem work correctly. Project's Resource map is saved to **Battlehub_ResourceMap prefab** located in **RTSaveLoad /ResourceMaps/Resources/** folder**. Create Resource Map** menu item also creates or updates resource maps for each asset bundle in project. Resource maps for asset bundles saved outside of **Resources** folder. Name of resource map for asset bundle have following format:

ResourceMap_<bundle name>_<guid> where <bundle name> is name of asset bundle and <guid> is string representation of arbitrary System.Guid

2) **Build Type Model** will precompile TypeModel wich will be used for serialization at runtime. Result of execution of this command will be located in **Assets/Battlehub/Deps/RTTypeModel.dll**

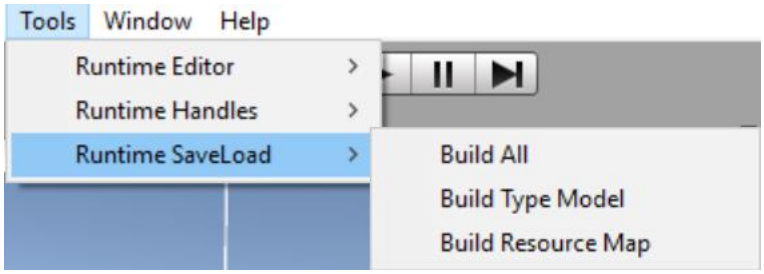

Fig.1.2 Runtime Save&Load menu

3) **Build All** will build ResourceMap first, then it will build TypeModel

1.3 Runtime Transform Handles Menu

If you need something lightweight instead of Runtime Editor, you could use runtime handles package. Runtime Transform Handles have separate menu.

There are three menu items and one submenu:

1. **Create** menu item creates simple selection controller, box selection component and 3 handles (positon, rotation, scale)
2. **Enable Editing** makes game object or prefab visible to selection controller
3. **Disable Editing** makes game object or invisible to selection controller
4. **Demo->Create Editor** creates demo editor
5. **Demo->Expose Prefab** makes prefab visible to demo editor
6. **Demo->Hide Prefab** makes prefab invisible to demo editor

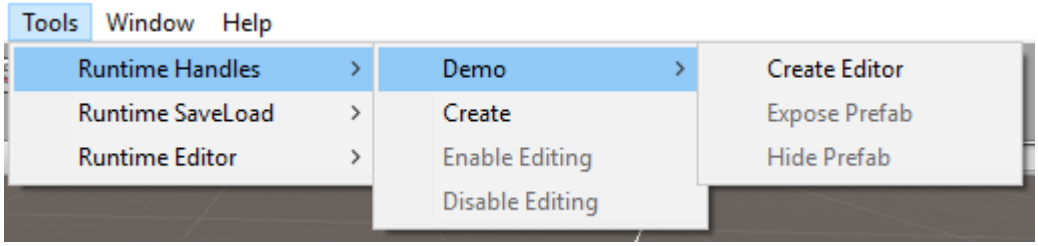

Fig.1.3 Runtime Handles menu

2. Getting Started

Whole process is shown in this video: https://vimeo.com/192127888

There are several easy streps to get started:

1. Create editor using **Tools->Runtime Editor->Create**

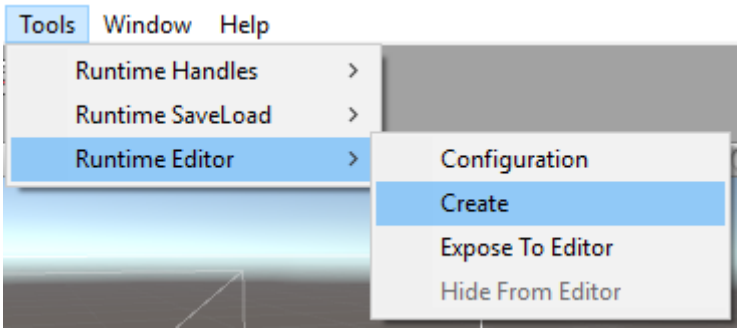

Fig. 2.1 Create Runtime Editor

2. Select scene objects you want to make available to editor and click **Tools->Runtime Editor->Expose To Editor** or just attach **ExposeToEditor** script to selected game objects.

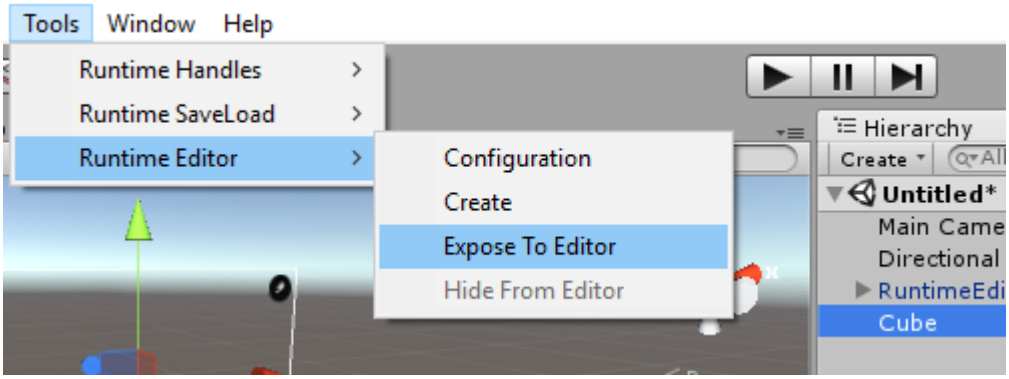

Fig. 2.2 Expose “Cube” to editor

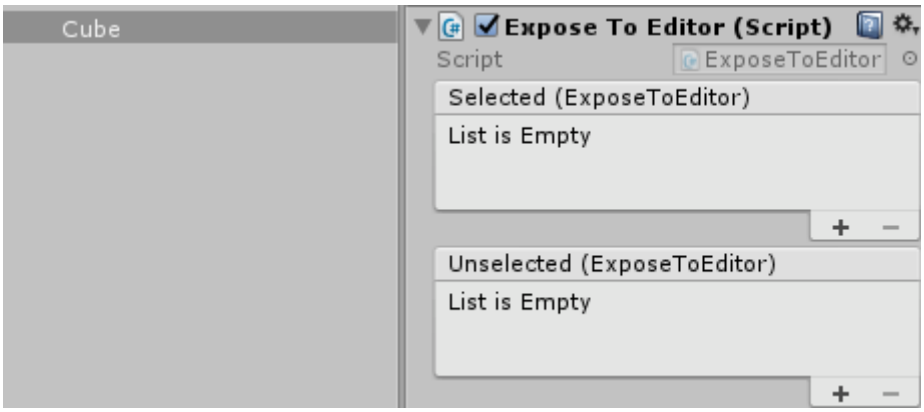

Fig. 2.3 ExposeToEditor script

3. Select resources or prefabs you want to expose to editor and click **Tools->Runtime Editor->Expose To Editor**

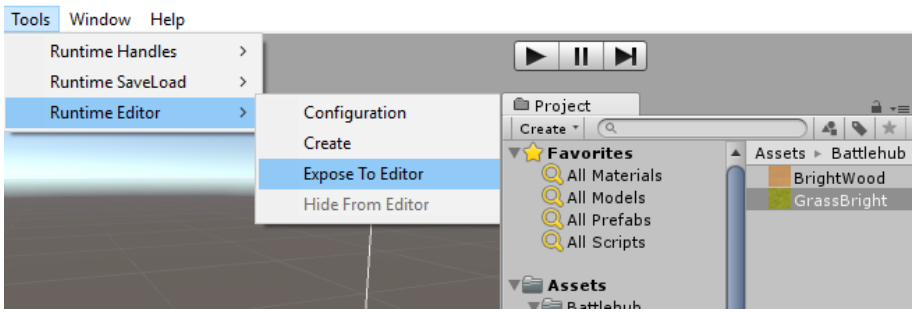

Fig. 2.3 Expose Texture to editor

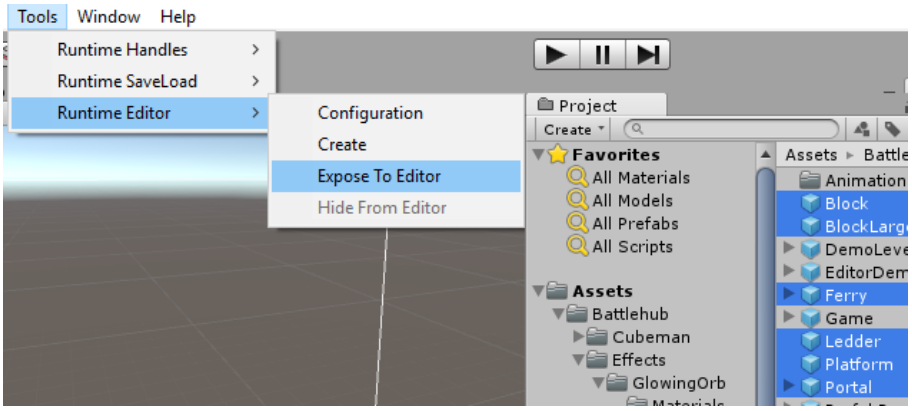

Fig. 2.4 Expose Prefabs to editor

4. Create Resource Map and Type Model using **Tools->Runtime SaveLoad->Build All menu item**

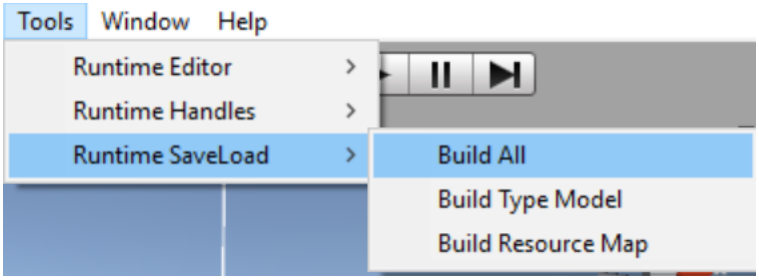

Fig. 2.5 Creating Resource Map

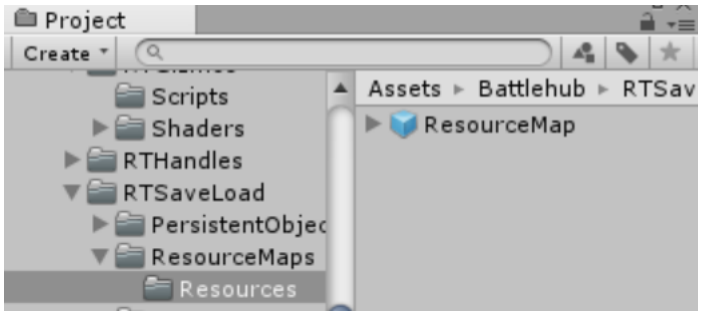

Fig. 2.6 Resource Map

5. Build & Run **File->Build & Run**

Fig.2.7 Open editor button

6. Open Editor

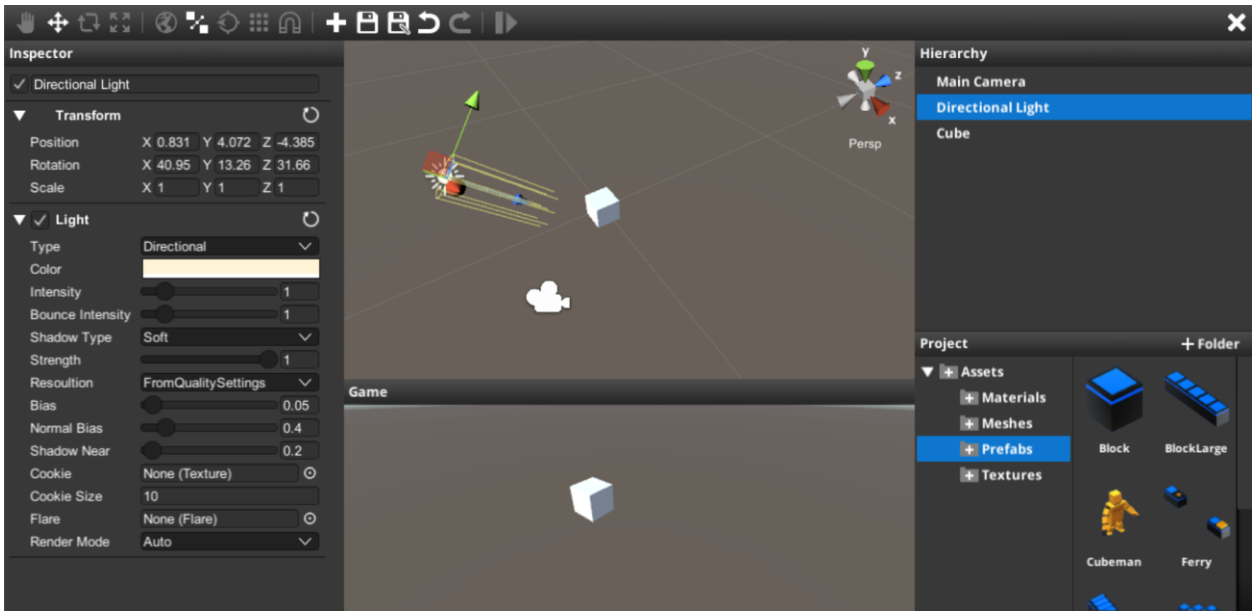

Fig.2.8 Opened editor with available objects and resources

3. Transform Handles

There are three transform handles included in this package: Position, Rotation and Scale. They behaves almost identical to their equivalents in unity editor. Transform Handles, Scene Gizmo, Grid, rendering classes and all required shaders can be found in **Assets/Battlehub/RTHandles folder.**

Position Handle, Rotation Handle and Scale Handle scripts allows you to choose Raycasting Camera, Selection Margin (in screen space coordinates), Target objects, Grid Size, and key which will switch position gizmo to “Snapping mode”

Scene Gizmo script let you to choose Scene Camera, Pivot Point (to rotate Scene Camera around), size of Gizmo.

Scene Gizmo could raise following events:

- Orientation Changing;
- Orientation Changed;
- Projection Changed;

Note: Scene gizmo always aligned to the top right corner of the screen

3.1 Getting Started With Transform Handles

Create editor using **Tools->Runtime Editor->Create**

1. Click **Tools->Runtime Handles->Create** menu item

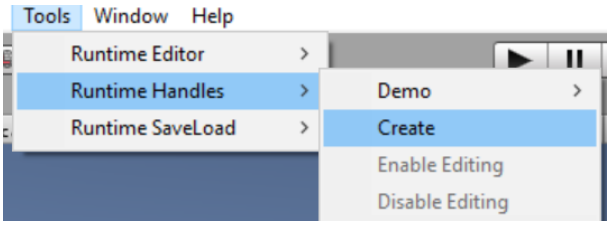

Fig.3.1 Create RuntimeTransform handles

2. Create several GameObjects and select them

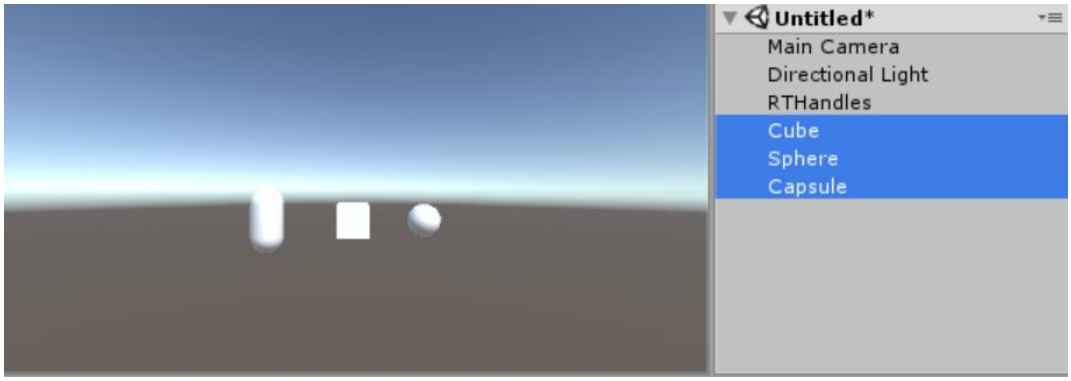

Fig.3.2 Several GameObjects selected

3. Click **Tools->Runtime Handles->Enable Editing** menu item

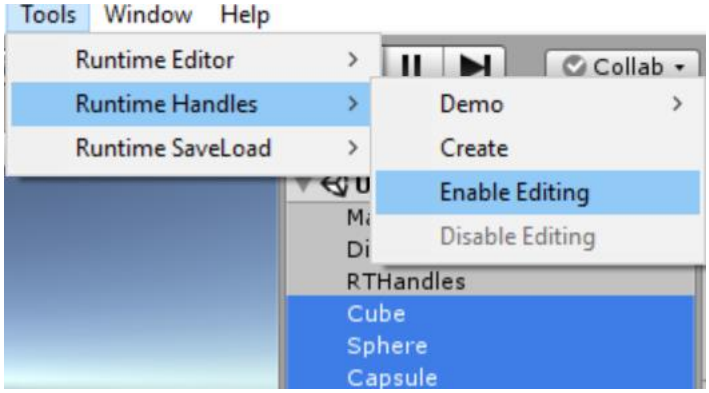

Fig.3.3 Enable objects editing

4. Press Play

You will be able to select objects using box selection, or by clicking them using left mouse button, or multiselect them by clicking LMB while holding Shift or Ctrl.

You could switch between postion, rotation and scale handles using Q,W,E,R buttons, switch pivot point mode between center & local using Z button and switch pivot point rotation between local object space & world space using X button.

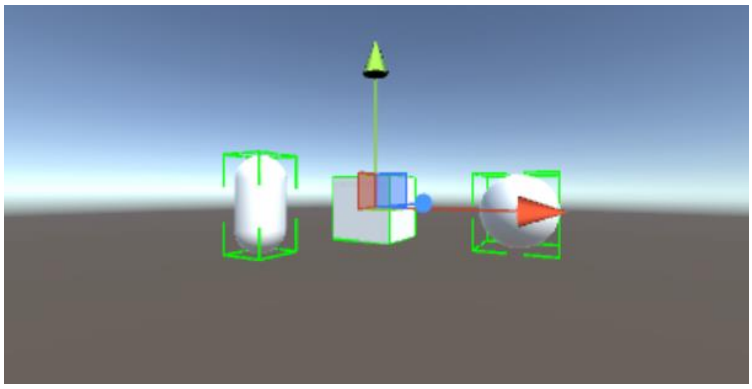

Fig.3.4 Runtime editing of object using position handle

3.2 Base Handle

Base handle is base class for Position, Rotation and Scale handles

All handles have following settings:

- **Highlight On Hover** - highlight axes on mouse over;
- **Enable Undo** - write records to RunitmeUndo;
- **Unit Snap Key –** used to activate unity snapping mode (default – Left CTRL)
- **Scene Camera** - used for raycasting and rendering;
- **Selection Margin** - area around each axis to make selection easier, measured in screen space;
- **Targets** – objects affected by handle

3.3 Position Handle

The procedure of creation of position handle (like all other transform handles) is quite simple:

1) Create Empty **GameObject**;
2) Assign **Assets/Battlehub/RTHandles/Scripts/PositionHandle.cs** script to it.

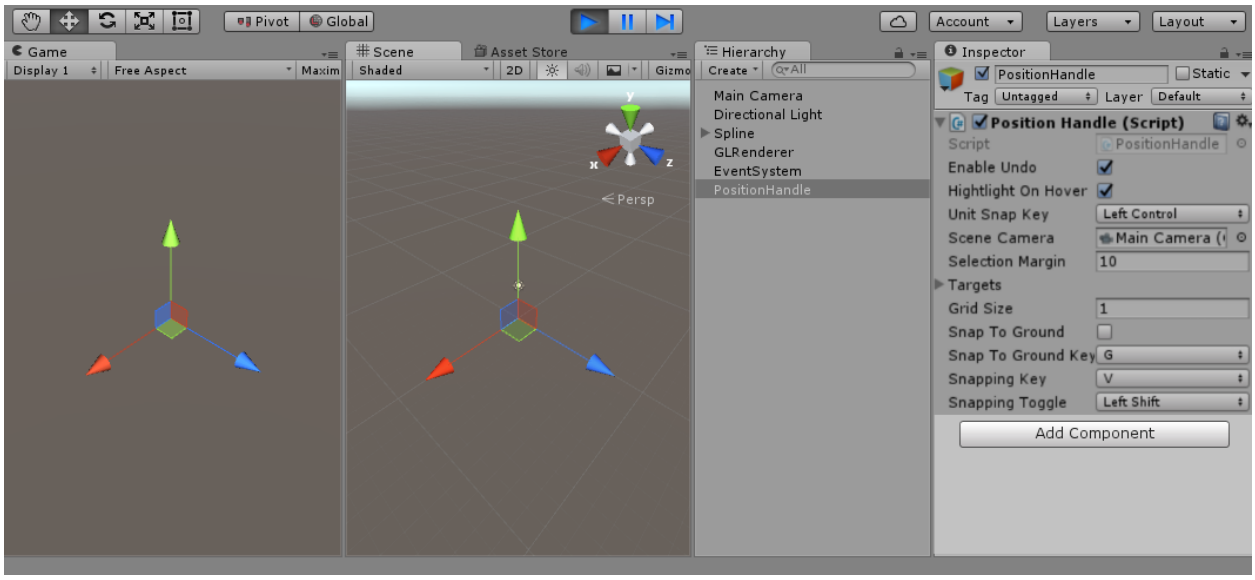

Fig.3.5 Runtime Position Handle

Position Handle has following settings:

- **Enable Undo** - write records to RunitmeUndo;
- **Highlight On Hover** - highlight axes on mouse over;
- **Unit Snap Key** – key used to activate unit snapping mode;
- **Scene Camera** - used for raycasting and rendering;
- **Selection Margin** – size of area around each axis to make selection easier, measured in screen space;
- **Targets** – objects affected by position handle
- **Grid Size** - used in unit snapping mode;
- **Snap To Ground** - snap target objects to ground if possible, activated using G button by default;
- **Spap To Ground Key –** key which is used to activate snap to ground mode
- **Snapping Key** – key which is used to enable vertex snapping or bounding box snapping mode;
- **Snapping Toggle** – hold this key and press “Snapping Key” to toggle between normal and snapping mode.

3.4 Rotation Handle

To use rotation handle do following:

1) Create Empty **GameObject**;
2) Assign **Assets/Battlehub/RTHandles/Scripts/RotationHandle.cs** script to it.

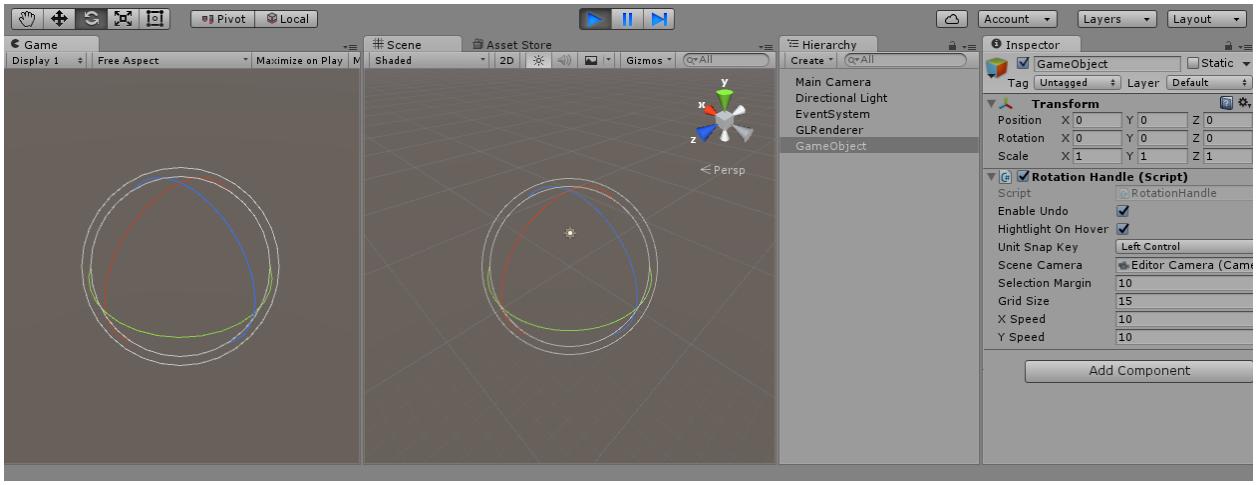

Fig.3.6 Runtime Rotation Handle

Rotation handle has following settings:

- **Enable Undo** - write records to RunitmeUndo;
- **Highlight On Hover** - highlight axes on mouse over;
- **Unit Snap Key** – key used to activate unit snapping mode;
- **Scene Camera** - used for raycasting and rendering;
- **Selection Margin** – size of area around each axis to make selection easier, measured in screen space;
- **Targets** – objects affected by rotation handle
- **Grid Size** - used in unit snapping mode, measured in degrees;
- **X Speed, Y Speed** - mouse sensitivity in free rotation mode;

3.5 Scale Handle

To use scale handle do following:

1) Create Empty **GameObject**;
2) Assign **Assets/Battlehub/RTHandles/Scripts/ScaleHandle.cs** script to it.

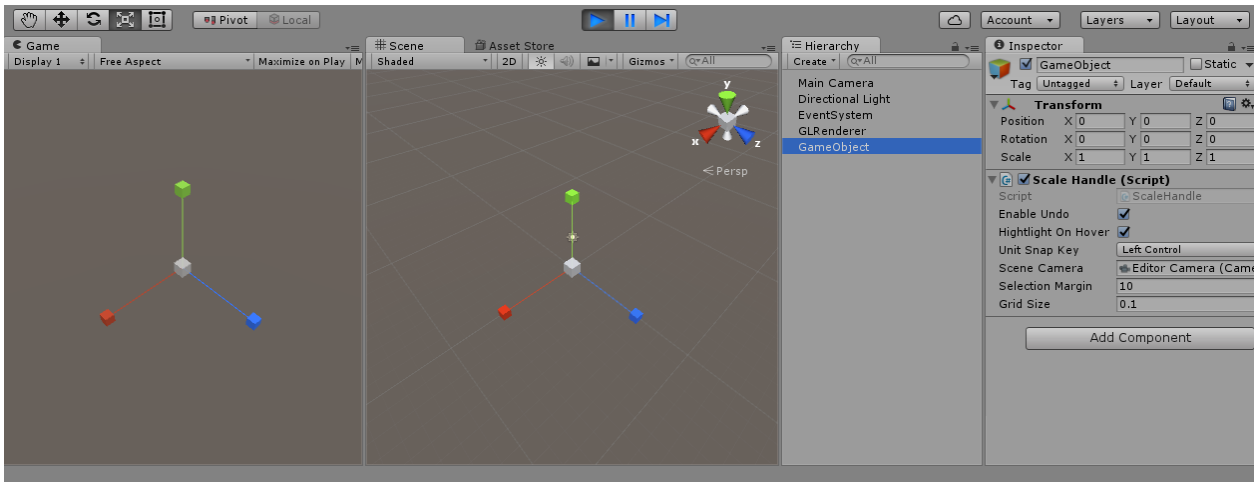

Fig.3.7 Runtime Scale Handle

Scale handle has following settings:

- **Enable Undo** - write records to RunitmeUndo;
- **Highlight On Hover** - highlight axes on mouse over;
- **Unit Snap Key** – key used to activate unit snapping mode;
- **Scene Camera** - used for raycasting and rendering;
- **Selection Margin** – size of area around each axis to make selection easier, measured in screen space;
- **Targets** – objects affected by scale handle
- **Grid Size** - used in unit snapping mode, measured in fractions of initial scale, activated by LeftControl key;

3.6 Scene Gizmo

To use scale gizmo do following:

1) Create Empty **GameObject**;
2) Assign **Assets/Battlehub/RTHandles/Scripts/SceneGizmo.cs** script to it.

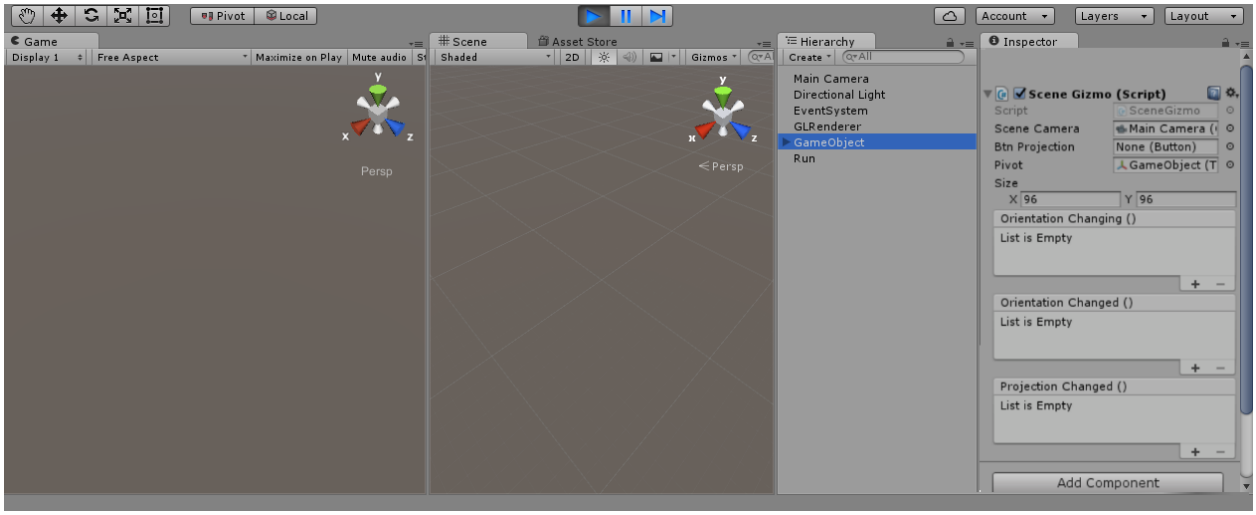

Fig.3.8 Runtime Scene Gizmo

Scene gizmo has following settings:

- **Scene Camera** - camera modified by scene gizmo;
- **BtnProjection** – optional reference to UGUI button used to switch between perspective and orthographic projections;
- **Pivot** – pivot object transform to rotate SceneCamera around;
- **Size** – size of scene gizmo
- **Orientation Changing Event** – raised before Scene Gizmo rotation started
- **Orientation Changed Event** – raised after Scene Gizmo rotation finished
- activated by LeftControl key;

3.7 Grid

To use grid do following:

1) Create Empty **GameObject**;
2) Assign **Assets/Battlehub/RTHandles/Scripts /Grid.cs** script to it

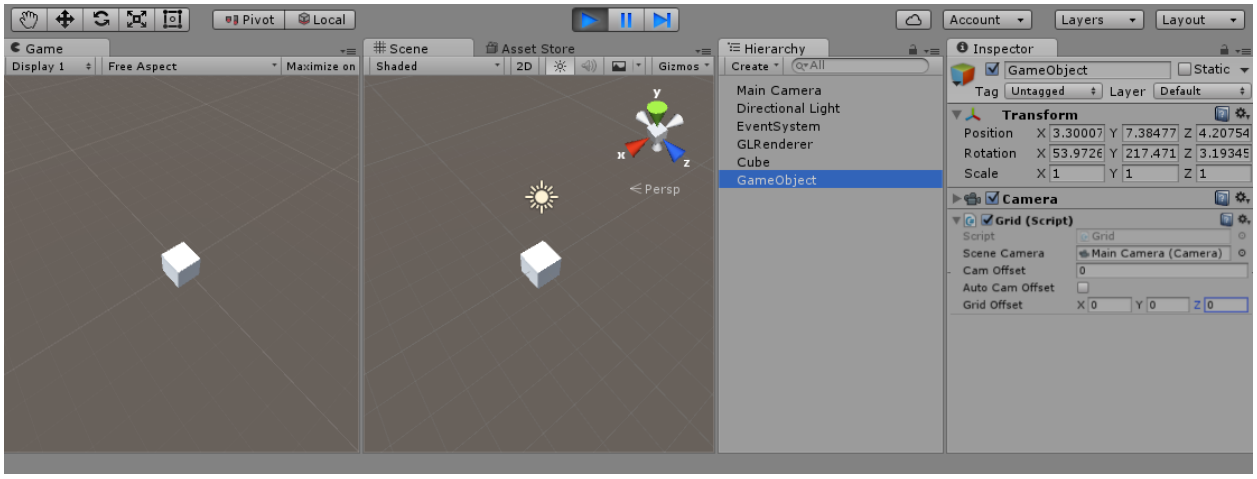

Fig.3.9 Runtime Grid

Grid has following settings:

- **Scene Camera** - camera using for grid rendering
- **Cam Offset** – control opacity of intermediate grid lines
- **Auto Cam Offset** – opacity of intermediate lines is controlled autmatically
- **Grid Offset** – grid transform component is controlled by Scene Camera. Use this property to control grid offset relative to camera

3.8 Lock Axes Script

There might be cases when you need to prevent certain axis from being modified by position, rotation and scale handles. To do this attach **LockAxes** script to game object and select axes you want to lock.

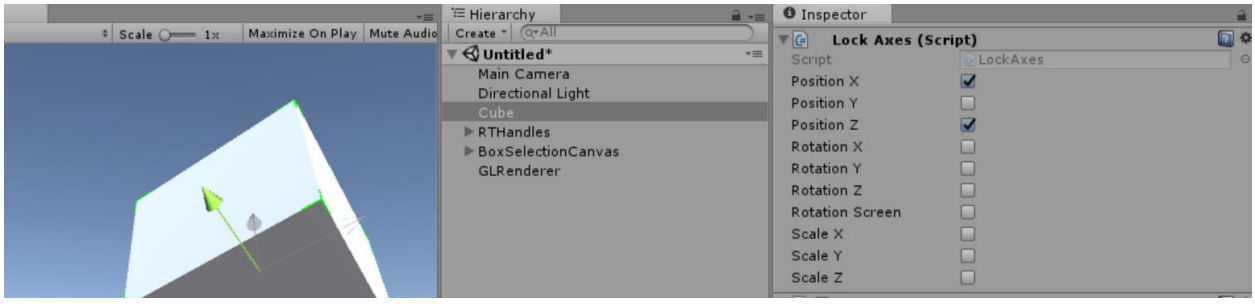

Fig 3.10 Position handle with locked x and z axes

3.9 Box Selection

To use BoxSelection do following:

1) Create Empty **GameObject**;
2) Assign **Assets/Battlehub/RTHandles/Scripts/BoxSelection.cs** script to it

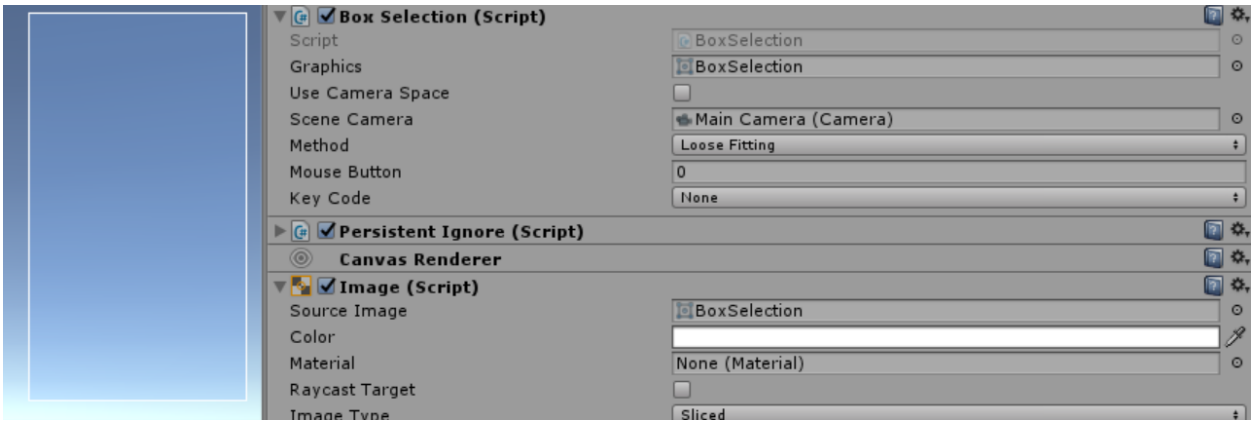

Fig 3.11 Box Selection

Grid has following settings:

- **Griphics** – sprite used to render box selection
- **Use Camera Space** – use camera space for rendering if selected, or screen space if unselected.
- **Scene Camera** - used for raycasting and rendering;
- **Method** – box selection method:
    - Loose Fitting – use renderer bounds and collider
    - Bounds Center – use bounds center
    - Transform Center – use transform center
- **Mouse Button 0 – left, 1 – right, 2 – middle**
- **Key Code** – key to activate BoxSelection (none by default means that no key needs to be pressed to activate box selection)

To get objects selected by BoxSelection you have to use RuntimeSelection class.

See: How to get objects selected by BoxSelection example for more info

4. Runtime Gizmos

There are nine gizmos included in this package: **BoxColliderGizmo**, **SphereColliderGizmo**, **CapsuleColliderGizmo**, **PointLightGizmo**, **SpotlightGizmo**, **DirectionalLightGizmo**, **AudioSourceGizmo**, **AudioReverbZoneGizmo** and **SkinnedMeshRendererGizmo**. Each gizmo could be used to control certain properties of corresponding components. Runtime Gizmos behaves almost identical to their equivalents in unity editor. There are also several base classes which might be helpful for implementation of additional gizmos. All gizmos, their base classes, rendering classes and all required shaders can be found in **Assets/Battlehub/RTGizmos folder.**

Each Gizmo script allows you to choose raycasting camera, selection margin (in screen space coordinates), target object, size of grid, line color, handle color and selection color as well as the key which will switch position gizmo to “Unit Snapping mode”

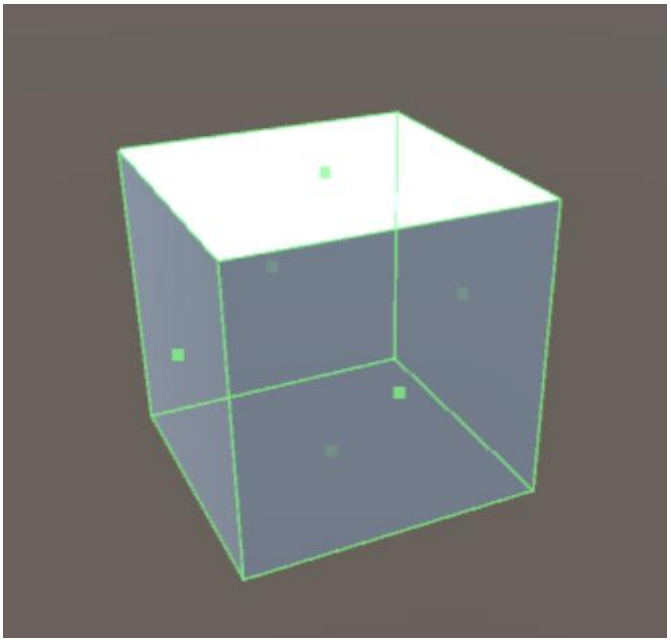

Fig. 4.1 Runtime Gizmo

4.1 Base Gizmo

Located in Assets/Battlehub/RTGizmos/Scripts/BaseGizmo.cs

BaseGizmo is base class for BoxGizmo, SphereGizmo, CapsuleGizmo and ConeGizmo. All gizmos have following settings:

- **Grid Size** – size of step in unit snapping mode (1.0 by default)
- **Line Color** –color of gizmo lines;
- **Handles Color –** color of handle

- **Selection Color** – color of selected handle.
- **Enable Undo** – if set to true then RuntimeGizmo will write all changes to undo stack (true by default)
- **Unit Snap Key** – key wich will switch RuntimeGimzo to unity snapping mode (Left Ctrl by default);
- **Target** – reference to target object

4.2 Box Gizmo

Located in Assets/Battlehub/RTGizmos/Scripts/BoxGizmo.cs

Base class for all gizmos that have box shape:

- BoxColliderGizmo,
- SkinnedMeshRendererGizmo

4.3 Sphere Gizmo

Located in Assets/Battlehub/RTGizmos/Scripts/SphereGizmo.cs

Base class for all gizmos that have sphere shape:

- Sphere Collider Gizmo,
- Pointlight Gizmo,
- Audio Source Gizmo,
- Audio Reverb Zone Gizmo

4.4 Capsule Gizmo

Located in Assets/Battlehub/RTGizmos/Scripts/CapsuleGizmo.cs

Base class for all gizmos that have capsule shape:

- Capsule Collider Gizmo

4.5 Cone Gizmo

Located in Assets/Battlehub/RTGizmos/Scripts/ConeGizmo.cs

Base class for all gizmos that have cone shape:

- Spotlight Gizmo

4.6 Box Collider Gizmo

Located in Assets/Battlehub/RTGizmos/Scripts/BoxColliderGizmo.cs

Box Collider Gizmo could be attached to object with BoxCollider

1) Create GameObject with BoxCollider
2) Assign BoxColliderGizmo script to it

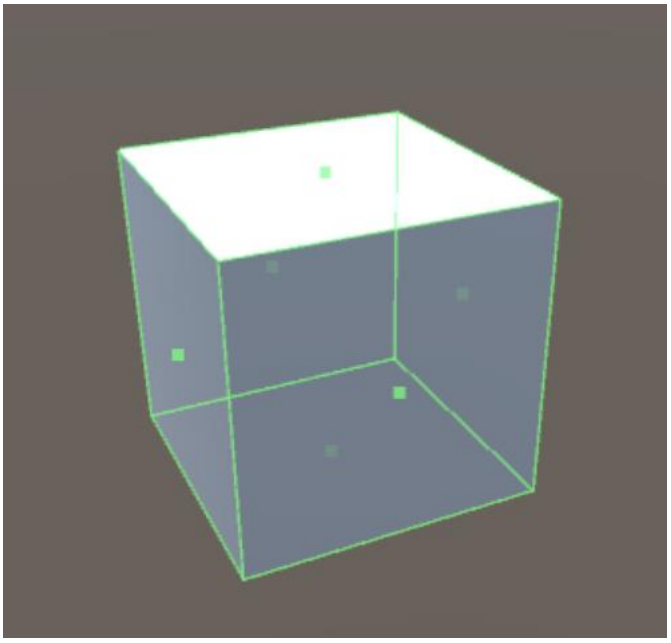

Fig. 4.2 Box Collider Gizmo

4.7 Sphere Collider Gizmo

Located in Assets/Battlehub/RTGizmos/Scripts/SphereColliderGizmo.cs

Sphere Collider Gizmo could be attached to object with SphereCollider

1) Create GameObject with Sphere Collider
2) Assign SphereColliderGizmo script to it

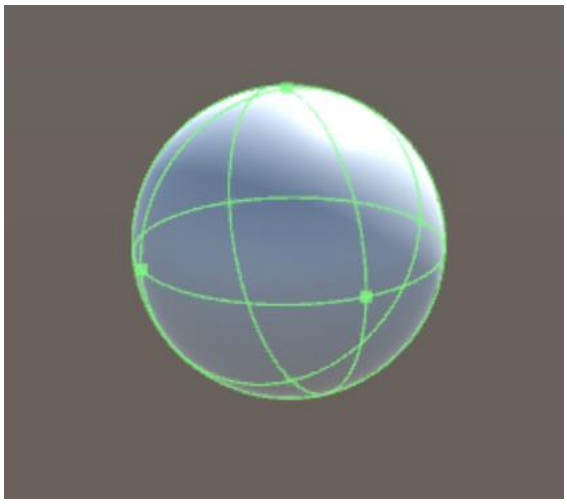

Fig. 4.3 Sphere Collider Gizmo

4.8 Capsule Collider Gizmo

Located in Assets/Battlehub/RTGizmos/Scripts/CapsuleColliderGizmo.cs

Capsule Collider Gizmo could be attached to object with CapsuleCollider

1) Create GameObject with Capsule Collider
2) Assign CapsuleColliderGizmo script to it

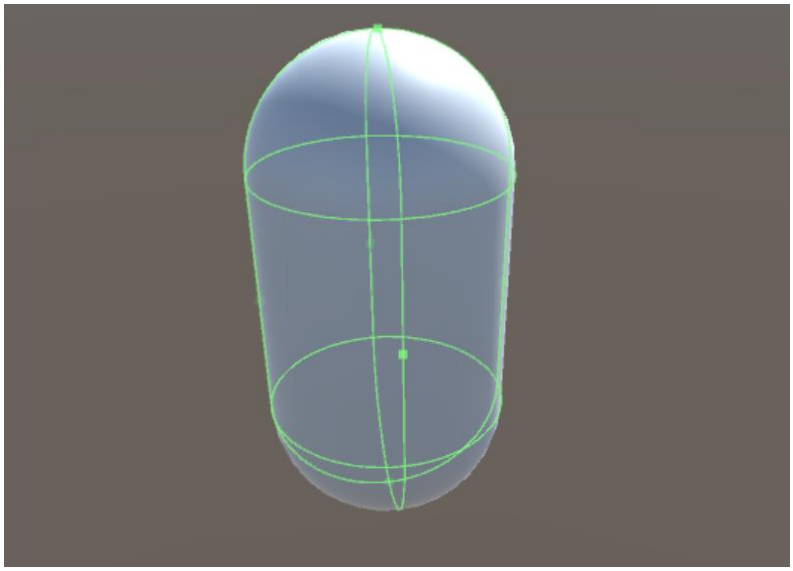

Fig.4.4 Capsule Collider Gimzo

4.9 Point Light Gizmo

Located in Assets/Battlehub/RTGizmos/Scripts/PointLightGizmo.cs

Point Light Gizmo could be attached to Point Light

1) Create Point Light
2) Assign LightGizmo script to it

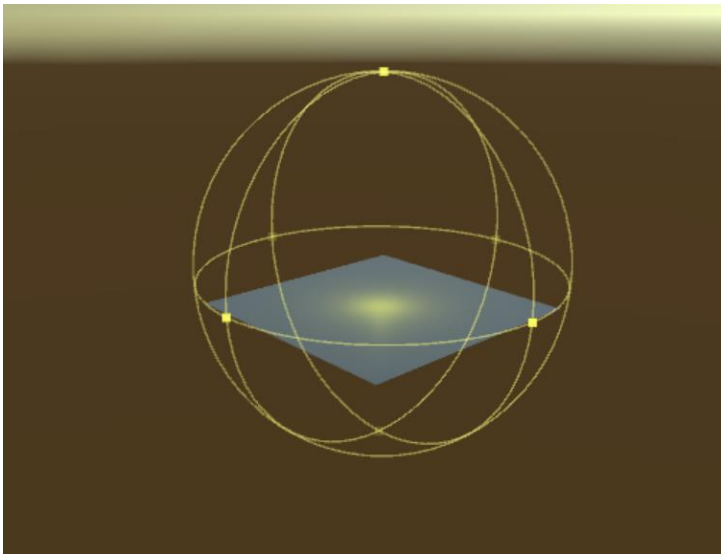

Fig.4.5 Point Light Gizmo

4.10 Spot Light Gizmo

Located in Assets/Battlehub/RTGizmos/Scripts/SpotLightGizmo.cs

Spot Light Gizmo could be attached to Spot Light

1) Create Spot Light
2) Assign LightGizmo script to it

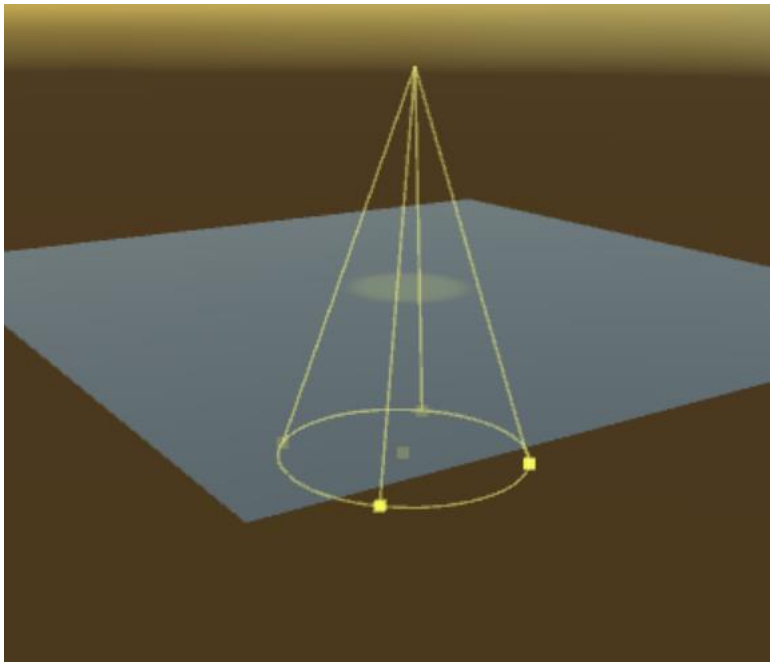

Fig.4.6 Spot Light gizmo

4.11 Directional Light Gizmo

Located in Assets/Battlehub/RTGizmos/Scripts/DirectionalLightGizmo.cs

Directional Light Gizmo could be attached to DirectionalLight

1) Create Directional Light
2) Assign Light Gizmo Script to it

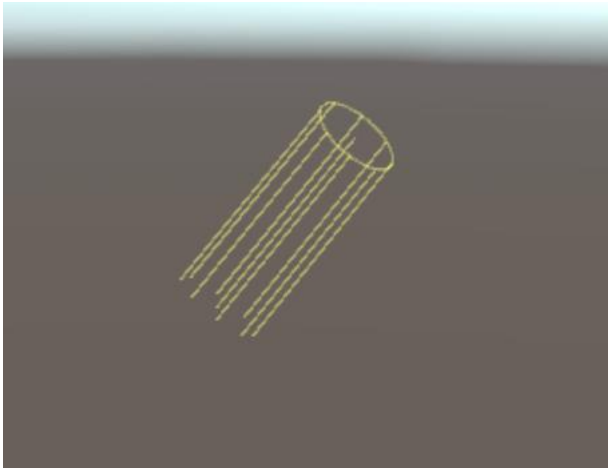

Fig.4.7 Directional Light Gizmo

4.12 Audio Source Gizmo

Located in Assets/Battlehub/RTGizmos/Scripts/AudioSourceGizmo.cs

Audio Source Gizmo could be attached to AudioSource

1) Create Audio Source
2) Assign Audio Source Gizmo Script to it

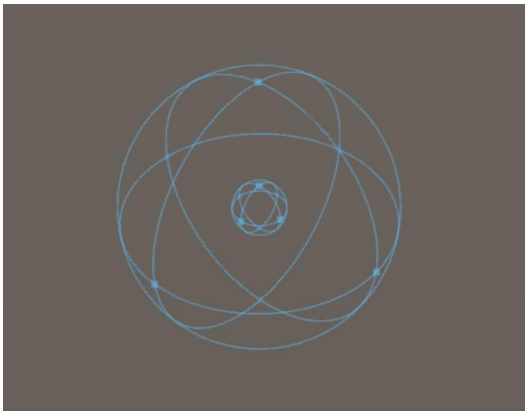

Fig. 4.8 Audio Source Gizmo

4.13 Audio Reverb Zone Gizmo

Located in Assets/Battlehub/RTGizmos/Scripts/AudioReverbZoneGizmo.cs

Same as AudioSouce Gizmo

4.14 SkinnedMeshRenderer Gizmo

Located in /Battlehub/RTGizmos/Scripts/SkinnedMeshRendererGizmo.cs

SkinnedMeshRenderer Gizmo could be attached to object with SkinnedMesh

1) Create GameObject with SkinnedMeshRenderer
2) Assign SkinnedMeshRenderer Gizmo Script to it

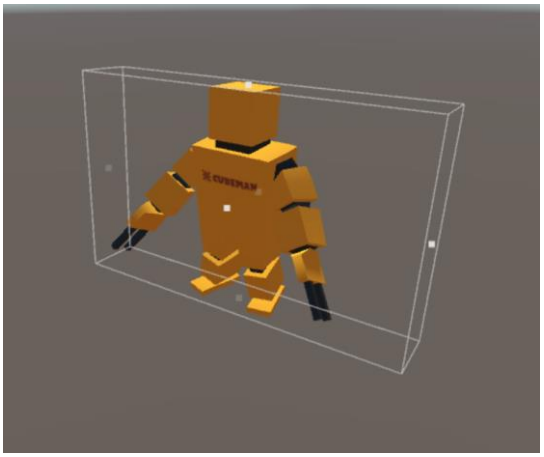

Fig. 4.9 Skinned Mesh Renderer Gizmo

5. Gizmos and Handles Rendering

5.1 IGL

Located in Assets/Battlehub/RTCommon/Scripts/GLRenderer.cs

Implement this interface to make object available to GLRenderer

```
public interface IGL {
        void Draw();
}
```

5.2 GLRenderer

Located in Assets/Battlehub/RTCommon/Scripts/GLRenderer.cs

GLRenderer is a singleton used by GLCamera script to render all registered objects. Register object for rendering by calling `public void Add(IGL gl)` method. Cancel object rendering by calling `public void Remove(IGL gl)` method

Example usage:

```
private void OnEnable() {
    if (GLRenderer.Instance != null) {
        GLRenderer.Instance.Add(this);
    }
}

private void OnDisable(){
    if (GLRenderer.Instance != null) {
        GLRenderer.Instance.Remove(this);
    }
}
```

5.3 GLCamera

Located in Assets/Battlehub/RTCommon/Scripts/GLCamera.cs

Attach this script to any camera and GL graphics will be rendered with this camera

```
[ExecuteInEditMode]
public class GLCamera : MonoBehaviour
{
    private void OnPostRender()
    {
        if(GLRenderer.Instance != null)
        {
            GLRenderer.Instance.Draw();
        }
    }
}
```

5.4 RuntimeHandles

Located in Assets/Battlehub/RTHandles/Scripts/RuntimeHandles.cs

This class contains rendering code and helper methods for all handles scene gizmo and grid.

Scale of position, rotation and scale handles:

```
public const float HandleScale = 1.0f;
```

Get Screen Scale factor for given world position and camera:

```
public static float GetScreenScale(Vector3 position, Camera camera)
```

Draw Position Handle with given world position, rotation and selected axis. Use snapMode to render position handle in SnapMode. Use lock object to change rendering of locked axes.

```
public static void DoPositionHandle(Vector3 position, Quaternion rotation,
        RuntimeHandleAxis selectedAxis = RuntimeHandleAxis.None,
        bool snapMode = false,
        LockObject lockObject = null)
```

Draw Rotation Handle with given world position, rotation and selected axis.

```
public static void DoRotationHandle(Quaternion rotation, Vector3 position,
      RuntimeHandleAxis selectedAxis = RuntimeHandleAxis.None,
      LockObject lockObject = null)
```

Draw Scale Handle with given world position, rotation, scale and selected axis:

```
public static void DoScaleHandle(Vector3 scale, Vector3 pos, Quaternion rotation,
      RuntimeHandleAxis selectedAxis = RuntimeHandleAxis.None,
      LockObject lockObject = null)
```

Draw Scene Gizmo with given world position, rotation. Selection specified using Vector3 selection arg, where Vector3.zero means “not selected”. xAlpha, yAlpha and zApha args are used to fade in and fade out corresponding axes. gizmoScale = 1 is used to render Scene Gizmo in 96x96 rectangle:

```
public static void DoSceneGizmo(Vector3 pos, Quaternion rot, Vector3 selection,
      float gizmoScale,
      float xAlpha = 1.0f,
      float yAlpha = 1.0f,
      float zAlpha = 1.0f)
```

Get far plane for grid rendering camera:

```
public static float GetGridFarPlane()
```

Draw Grid using grid offset relative to rendering camera and cam vertical offset to control intermediate lines opacity:

```
public static void DrawGrid(Vector3 gridOffset, float camOffset = 0.0f)
```

5.5 Runtime Gizmos

Located in Assets/Battlehub/RTGizmos/Scripts/RuntimeGizmos.cs

This class contains rendering code and helper methods for all runtime gizmos

Scale of dragable handles of runtime gizmos:

```
public const float HandleScale = 2.0f;
```

Draw Selected handle using following method:

```
public static void DrawSelection(
        Vector3 position, Quaternion rotation, Vector3 scale, Color color)
```

Following method is used to draw handles for Box, Sphere and Capsule Gizmos:

```
public static void DrawCubeHandles(
        Vector3 position, Quaternion rotation, Vector3 scale, Color color)
```

Following method is used to draw handles for Cone Gizmo (SpotLight gizmo):

```
public static void DrawConeHandles(
        Vector3 position, Quaternion rotation, Vector3 scale, Color color)
```

Draw Cone with specific height and radius:

```
public static void DrawWireConeGL(
float height, float radius, Vector3 position, Quaternion rotation, Vector3 scale, Color color)
```

Draw capsule along axis (axis: 0 – x, 1 – y, 2 – z)

```
 public static void DrawWireCapsuleGL(
        int axis, float height, float radius, Vector3 position,
        Quaternion rotation, Vector3 scale,  Color color)
```

Draw Directional Light gizmo:

```
public static void DrawDirectionalLight(
        Vector3 position, Quaternion rotation, Vector3 scale, Color color)
```

Draw Sphere:

```
public static void DrawWireSphereGL(
        Vector3 position, Quaternion rotation, Vector3 scale, Color color)
```

Draw Cube:

```
public static void DrawWireCubeGL(
      ref Bounds bounds, Vector3 position, Quaternion rotation, Vector3 scale, Color color)
```

6. Common Infrastructure

6.1 ExposeToEditor

Located in Battlehub/RTCommon/Scripts/Infrastructure/ExposeToEditor.cs.

Attach this script to any GameObject you want to make it available for runtime editing.

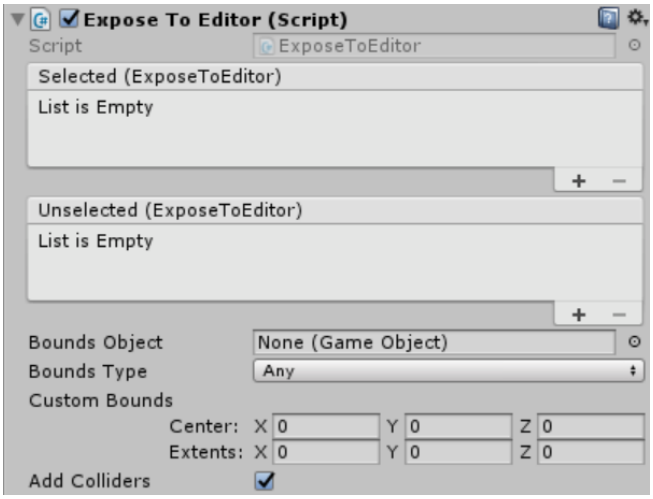

Fig.5.1 ExposeToEditor script

UnityEvents:

- **Selected Event**– raised when object with ExposeToEditor added to RuntimeSelection
- **Unselected Event**– raised when object with ExposeToEditor removed from RuntimeSelection

Delegates:

```
public delegate void ExposeToEditorEvent(ExposeToEditor obj);
public delegate void ExposeToEditorChangeEvent<T>(
            ExposeToEditor obj, T oldValue, T newValue);
```

Events:

- Raised when any of corresponding mono-behavor events occurred. See https://docs.unity3d.com/Manual/ExecutionOrder.html for details

```
public static event ExposeToEditorEvent Awaked;
public static event ExposeToEditorEvent Started;
public static event ExposeToEditorEvent Enabled;
public static event ExposeToEditorEvent Disabled;
```

- Raised before hierarchy items and colliders destroyed

```
public static event ExposeToEditorEvent Destroying;
```

- Raised after hierarchy items and colliders destroyed

```
public static event ExposeToEditorEvent Destroyed;
```

- Raised when object marked as destroyed by undo&redo subsystem

```
public static event ExposeToEditorEvent MarkAsDestroyedChanged;
```

- GameObject's name changed

```
public static event ExposeToEditorEvent NameChanged;
```

- position, rotation or localScale changed

```
public static event ExposeToEditorEvent TransformChanged;
```

- ExposeToEditor Parent property changed

```
public static event ExposeToEditorChangeEvent<ExposeToEditor> ParentChanged;
```

Fields & Properties:

- AddColliders – add extra colliders to enable selection.
- Bounds Object – which GameObject will be used to draw SelectionGizmo
- Bounds Type –
    - Mesh – MeshFilter.mesh.bounds will be used to draw SelectionGizmo
    - SkinnedMesh – SkinnedMeshRenderer.sharedMesh.bounds will be used to draw SelectionGizmo
    - Custom – user defined bounds
    - Any – any of the above
- Custom Bounds – used if BoundsType == Custom

6.1.1 Expose To Editor Usage Example

```
private void Awake()
{
    ExposeToEditor.Started += OnObjectStarted;
}
private void OnDestroy()
{
    ExposeToEditor.Started -= OnObjectStarted;
}
private void OnObjectStarted(ExposeToEditor obj)
{
    //implement your handler
}
```

6.2 HierarchyItem

Located in Battlehub/RTCommon/Scripts/HierarchyItem.cs

This class is used to create links between ExposeToEditor objects in hierarchy. Suppose you have two objects exposed to editor. One of them is grandparent of another. Object between them will be HierarchyItem. HierarchyItem will track all hierarchy changes between ExposeToEditor objects. HierarchyItems are created automatically. You don't have to assign them to GameObjects.

6.3 RuntimeSelection

Located in Battlehub/RTCommon/Scripts/RuntimeSelection.cs

This static class is limted equivalent of UnityEditor.Selection class

Delegates:

```
public delegate void RuntimeSelectionChanged(Object[] unselectedObjects);
```

Events:

```
public static event RuntimeSelectionChanged SelectionChanged;
```

Properties:

- Gets selected GameObject

  ```
  public static GameObject activeGameObject { get;  }
  ```

- Gets or sets selected Object

  ```
  public static Object activeObject { get; set;}
  ```

- Gets or sets selected Objects

  ```
  public static Object[] objects { get; set;}
  ```

- Gets selected GameObjects

  ```
  public static GameObject[] gameObjects { get; }
  ```

- Gets Transform of selected GameObject

  ```
  public static Transform activeTransform { get; }
  ```

6.3.1 Runtime Selection Usage Example

```
private void Awake()
{
    RuntimeSelection.SelectionChanged += OnSelectionChanged;
}
private void OnDestroy()
{
    RuntimeSelection.SelectionChanged -= OnSelectionChanged;
}
private void OnSelectionChanged(Object[] unselectedObjects)
{
   Object[] selectedObjects = RuntimeSelection.objects;
}
```

6.4 RuntimeSelectionComponent

Located in RuntimeHandles/Scripts/RuntimeSelectionComponent.cs

Add gameObject with this script to scene to enable box selection and selection using raycasting functionality.

6.5 RuntimeTool, PivotRotation, PivotMode

Located in Battlehub/RTHandles/Scripts/Infrastructire/RuntimeTools.cs

These enumerations defined as following:

```
public enum RuntimeTool
{
  None,
  Move,
  Rotate,
  Scale,
  View,
}

public enum RuntimePivotRotation
{
  Local,
  Global
}

public enum RuntimePivotMode
{
   Center = 0,
   Pivot = 1
}
```

6.6 RuntimeTools

Located in Battlehub/RTCommon/Scripts/RuntimeTools.cs

This static class is used to get or set following:

- Current Tool (CurrentTool);
- Currently Active Tool (ActiveTool – reference to actively manipulated object);
- Pivot Rotation – the rotation of coordinate system in which position, rotation and scale handles will be rendered (local or global);
- Pivot Mode could be either Center or Pivot. If pivot selected then objects will be rotated and scaled around individual centers. If center is selected then rotation and scale handles will rotate and scale objects around common center;
- Snapping mode which could be either BoundingBox or Vertex;
- ToolChanged, PivotRotationChanged, PivotMode changed events;

- Whether alt and RMB pressed and scene view is in mouse orbiting mode (IsViewing);
- Show or hide Selection Gizmos (ShowSelectionGizmos)
- Show or hide Gizmos (ShowGizmos);
- Is auto-focus mode enabled (AutoFocus);
- Is unit snapping mode enabled (UnitSnapping);

Almost each property has corresponding change event.

```
public static class RuntimeTools
{
        public static event RuntimeToolsEvent ToolChanged;
        public static event RuntimeToolsEvent PivotRotationChanged;
        public static event RuntimeToolsEvent PivotModeChanged;
        public static event SpawnPrefabChanged SpawnPrefabChanged;
        public static event RuntimeToolsEvent IsViewingChanged;
        public static event RuntimeToolsEvent ShowSelectionGizmosChanged;
        public static event RuntimeToolsEvent ShowGizmosChanged;
        public static event RuntimeToolsEvent AutoFocusChanged;
        public static event RuntimeToolsEvent UnitSnappingChanged;
        public static event RuntimeToolsEvent IsSnappingChanged;
        public static event RuntimeToolsEvent SnappingModeChanged;

        public static bool IsViewing {get; set;}
        public static bool IsSceneGizmoSelected {get; set;}
        public static bool ShowSelectionGizmos {get; set;}
        public static bool AutoFocus {get; set;}
        public static bool UnitSnapping {get; set;}
        public static SnappingMode SnappingMode { get; set; }
        public static GameObject SpawnPrefab {get; set;}
        public static Object ActiveTool { get; set; }
        public static RuntimeTool Current {get; set;}
        public static RuntimePivotRotation PivotRotation {get; set;}
        public static RuntimePivotMode PivotMode { get; set; }

        public static void Reset();
}
```

Reset Method should be used to perform cleanup and reset RuntimeTools to initial state.

6.7 RuntimeToolsComponent

Located in Battlehub/RTHandles/Scripts/RuntimeToolsComponent.cs

Add gameObject with this script to scene and switch transform handles using Q,W,E,R keys. Use Z and X keys to switch PivotMode and PivotPointRotation

Q – view tool;

W – position handle;

E – rotation handle;

R – scale handle;

6.8 RuntimeUndo

Located in Battlehub/RTCommon/Scripts/RuntimeUndo.cs

This static class is used to record changes, maintain undo/redo stack and perform undo and redo operations.

```
public static class RuntimeUndo
```

Events:

- Raised before undo operation

  ```
  public static event RuntimeUndoEventHandler BeforeUndo;
  ```

- Raised after undo operation

  ```
  public static event RuntimeUndoEventHandler UndoCompleted;
  ```

- Raised before redo operation

  ```
  public static event RuntimeUndoEventHandler BeforeRedo;
  ```

- Raised after redo operation

  ```
  public static event RuntimeUndoEventHandler RedoCompleted;
  ```

- Raised whenever one of the following operations performed: Store, Restore, Purge

  ```
  public static event RuntimeUndoEventHandler StateChanged;
  ```

Properties & Fields:

- Maximum number of records in stack

  ```
  public const int Limit = 8192;
  ```

- Enables or disables Undo&Redo

  ```
  public static bool Enabled { get; set; }
  ```

- True if can perform undo operation

  ```
  public static bool CanUndo { get;}
  ```

- True if can perform redo operation

  ```
  public static bool CanRedo { get;}
  ```

Methods:

- Begin Record multiple changes

  ```
  public static void BeginRecord();
  ```

- End Record multiple changes

  ```
  public static void EndRecord();
  ```

- Register Create object operation:

  ```
  public static void BeginRegisterCreateObject(GameObject g)
  public static void RegisterCreatedObject(GameObject g)
  ```

- Register Destroy object operation:

```
public static void BeginDestroyObject(GameObject g)
public static void DestroyObject(GameObject g);
```

Note that object isn't actually destroyed. It only marked as destroyed. It will be destroyed during Purge operation.

- Record value of member of target object:

```
public static void RecordValue(object target, MemberInfo memberInfo)
```

- Record transform:

```
public static void RecordTransform(Transform target);
```

- Record selection:

```
public static void RecordSelection()
```

- Record object:

```
public static void RecordObject(object target, object state, ApplyCallback
applyCallback, PurgeCallback purgeCallback)
```

To use this method you have to implement apply and purge callback methods. See RecordTransform implementation for more details.

- Performs redo operation:

```
public static void Redo()
```

- Performs undo operation:

```
public static void Undo()
```

- Purge all records. Stack will be cleared, all "marked as destroyed" objects will be destroyed using Object.DestoryImmdiate

```
public static void Purge();
```

- Create new stack and store current undo&redo stack**:**

```
public static void Store()
```

- Restore previouly stored stack:

```
public static void Restore()
```

6.9 RuntimeUndoComponent

Located in Battleuhb/RTCommon/Scripts/RuntimeUndoComponent.cs

Add gameObject with this script to scene to preform undo & redo operations using CTRL + Z, CTRL + Y keys. (or SHIFT + Z, SHIFT + Y inside of unity editor)

6.10 RuntimeEditorWindow

Located in Battleuhb/RTCommon/Scripts/RuntimeEditorWindow.cs

Currently the main purpose of this class is to catch focus, and notify **RuntimeEditorApplication** which window of editor is active and thus will receive input.

Each window have type:

```
public RuntimeWindowType WindowType;

public enum RuntimeWindowType
{
   None,
   GameView,
   SceneView,
   Hierarchy,
   ProjectTree,
   Resources,
   Inspector,
   Other
}
```

6.11 RuntimeEditorApplication

Located in Battlehub/RTHandles/Scripts/Infrastructure/RuntimeEditorApp.cs

This class represents several important parts of editor state.

Events:

- Raised before play mode state changed (IsPlaying property changed)

  ```
  public static event RuntimeEditorEvent PlaymodeStateChanging;
  ```

- Raised after play mode state changed (IsPlaying property changed)

  ```
  public static event RuntimeEditorEvent PlaymodeStateChanged;
  ```

- Raised when editor opened or closed (IsOpened property changed)

  ```
  public static event RuntimeEditorEvent IsOpenedChanged;
  ```

- Raised when active window changed (ActivateWindow method is called)

  ```
  public static event RuntimeEditorEvent ActiveWindowChanged;
  ```

- Raised when active scene camera changed (ActiveSceneCameraIndex property changed)

  ```
  public static event RuntimeEditorEvent ActiveSceneCameraChanged;
  ```

Properties:

```
public static bool IsOpened {get; set; }
public static bool IsPlaying {get; set; }
public static RuntimeEditorWindow ActiveWindow {get;}
public static RuntimeWindowType ActiveWindowType { get; }
public static Camera ActiveSceneCamera { get; }
public static int ActiveSceneCameraIndex { get; set; }
```

6.12 MouseOrbit

Located in Assets/Battlehub/RTHandles/MouseOrbit.cs.

Responsible for camera rotation around pivot.

Slightly modified version of this script.

6.13 RuntimeSceneView

Located in Assets/Battlehub/RTHandles/Scripts/RuntimeSceneView.cs.

Responsible for scene view navigation

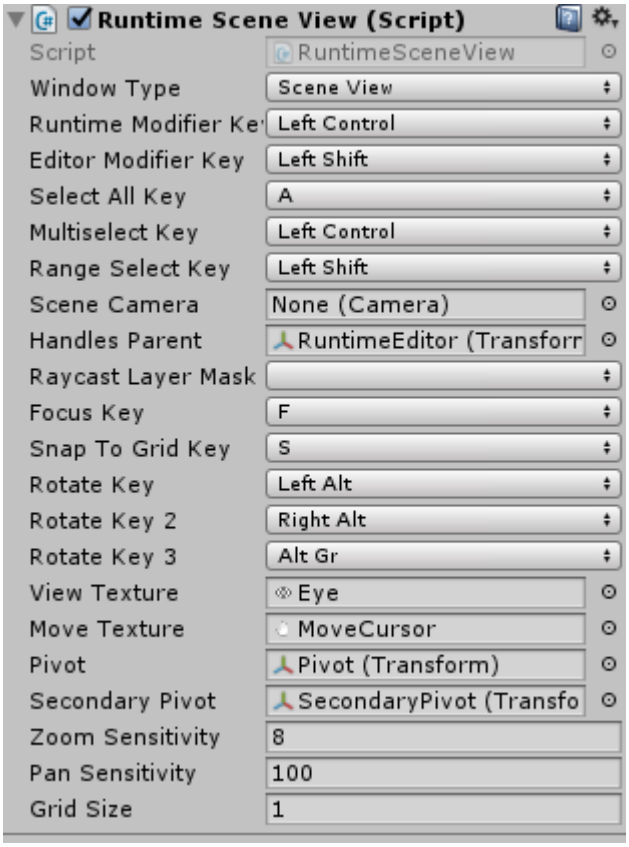

Fig.5.1 RuntimeSceneView script

Properties:

- Scene Camera (if none specified Camera.main will be used)
- Handles Parent defines where to spawn Postion, Rotation and Scale handles
- Pivot – used by mouse oribit
- Secondary Pivot – used to display grid and spawn objects
- Grid Size – measured in world units
- ViewTexture – displayed as cursor if scene view is in mouse orbiting state
- MoveTexture – displayed as cursor if scene view is in panning state

Select All is performed by SHIFT (CTRL) + A

Focus performed by pressing F key

Snap to grid selected object by pressing S key;

Rotation by pressing Rotate Key (Rotate Key 2, Rotate Key 3) + LMB

Pan by pressing RMB**.**

To Create SceneView create Empty GameObject and attach RuntimeSceneView script to it.

6.14 InputController

Located in Assets/Battlehub/RTCommon/Scripts /InputController.cs.

This class is used to block input when InputField selected.

6.15 Game

Located in Assets/Battlehub/RTCommon/Scripts/Game.cs.

The main purpose of this class is to spawn copies of editor objects when entering play mode.

7. Runtime Editor

7.1 Runtime Editor Layout

Located in Assets/Battlehub/RTEditor/Prefabs/RuntimeEditor.prefab is assembled runtime scene editor with all functionality available in package.

Runtime editor UI consists of following components:

- SceneView & ViewportFitter;
- GameView & ViewportFitter;
- Layout Element resizer (resizable panels);
- Header with Tools Panel and Close button;
- Hierarchy Window with TreeView control;
- Project Window with two sections:
    - Project Tree Window;
    - Project Resources Window;
- Inspector Window;
- Object Editor, Material Editor, Component Editor and Property editors;

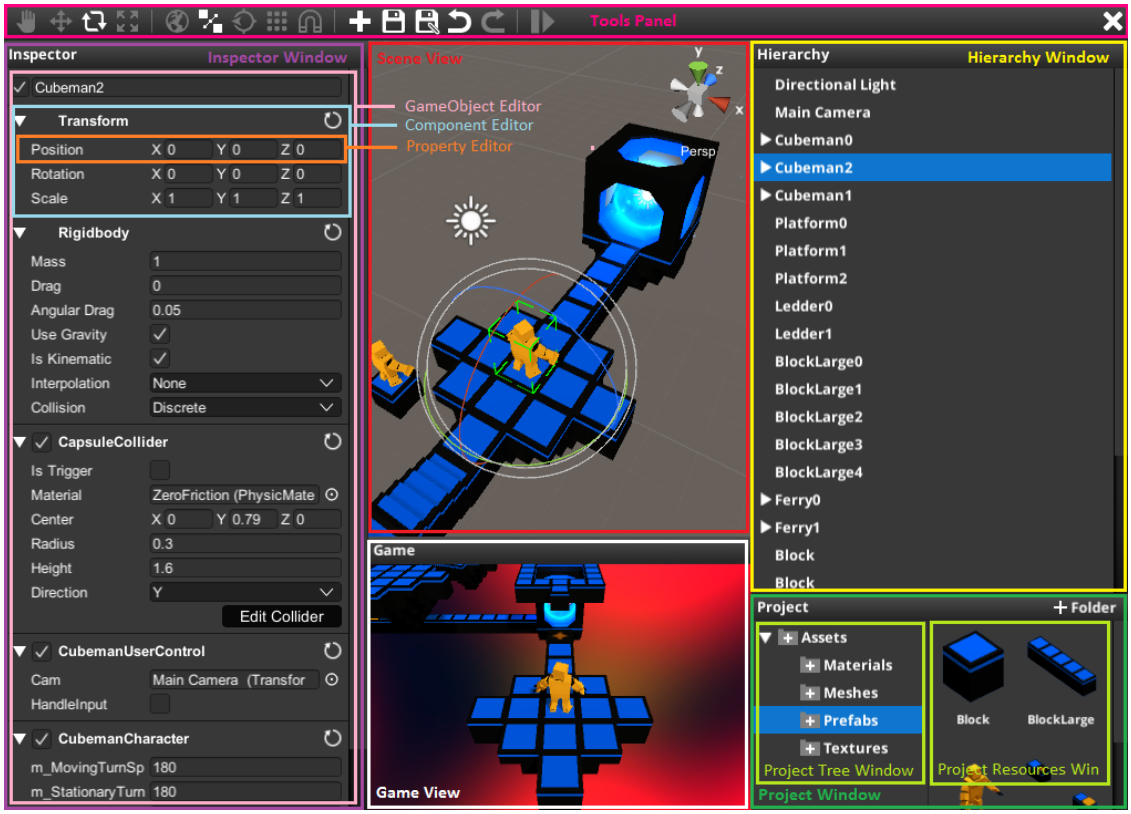

Fig.7.1 Runtime Editor Layout

Scene navigation and shortcuts are almost the same as in unity editor except following: Right mouse button is used to pan;

7.2 Runtime Editor Script

Located in Assets/Battlehub/RTEditor/Scripts/RuntimeEditor.cs

This class is responsible for wiring up hiding and showing editor components.

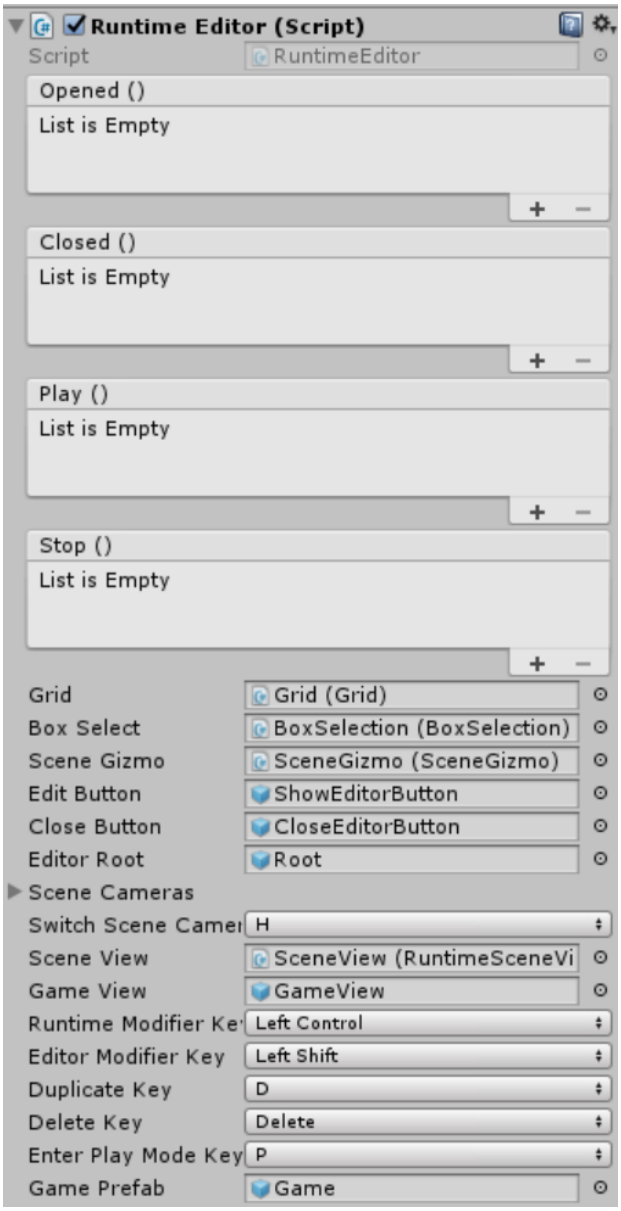

Fig.7.2 Runtime Editor Script

Unity Events:

- Opened – raised when editor opend
- Closed – raised when editor closed
- Play – raised when edtor enters play mode
- Stop – raised when editor enters edit mode

Properties & Fields:

- Grid – reference to object with Grid script attached

- Box Select – reference to object which will perform box selection
- Scene Gizmo – reference to scene gizmo
- Edit button 
- Close Button 
- Scene cameras array. If you want to use your own cameras in scene view, give editor references to them. Otherwise it will create copies of main camera.
- Switch Scene Camera key. If this key is pressed editor will disable scene camera and enable next scene camera from Scene Cameras array.
- SceneView – reference to SceneView
- GameView – reference to game view;
- Runtime Modifier Key, Editor Modifer Key. This is required because unity editor intercepts combinations of keys with CTRL+. Runtime Modifier Key used in build. Editor Modefier Key used inside of unity editor
- Duplicate Key – Key used to duplicate items (D);
- Delete Key - Key used to delete items (Del);
- Enter PlayMode Key (P);
- Game Prefab – editor will instantiate this object each time it enters play mode.

Editor instance can be accessed using Instance field

- ```
  public static RuntimeEditor Instance { get; }
  ```

7.3 Folder Template

Project Structure is defined using scripts located in Assets/Battlehub/RTEditor/Scripts/FolderTemplate.cs

This script can be used to define initial project structure, and specify which resources to display in each folder**.** Following structure is default:

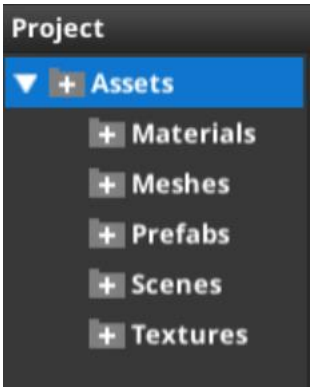

Fig.7.3 Default Project Structure
  ```

Folder structure shown in Fig.7.3 is pretty the same you can see under ProjectTemplate GameObject. Each child of ProjectTemplate defines folder; objects array of FolderTemplate script is used to define contents of that folder; TypeHint is preferred type of objects located in this folder. You could populate Objects array directly, however **recommended** way to do it is **to use Tools->Runtime Editor->Expose To Editor** menu item. This menu item will peform all neccesary checks.

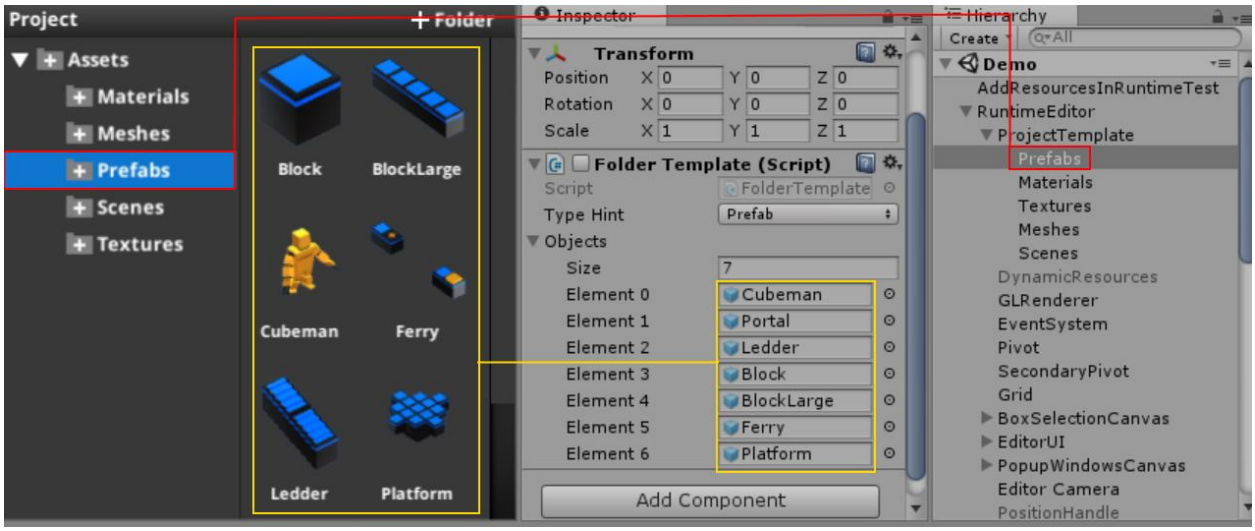

Fig.7.4 Project Template

By default project stored in `Application.persistentDataPath`. See https://docs.unity3d.com/ScriptReference/Application-persistentDataPath.html for details. Each folder has corresponding *.rtmeta file for additional metadata.

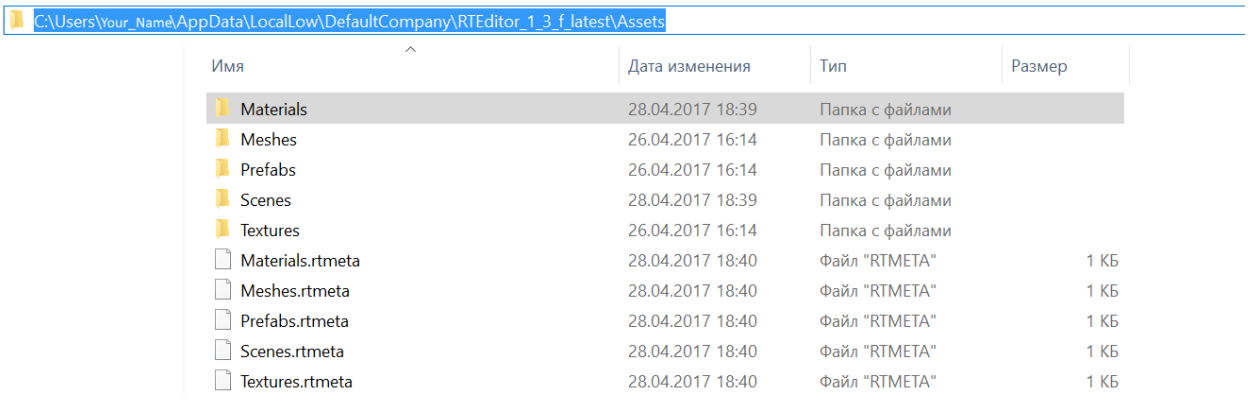

Fig.7.5 Project in file system

7.4 Tools Panel

Located in Assets/Battlehub/RTEditor/Scripts/Tools Panel.cs.

This class is responsible for Tools Panel visual state. It also handle events raised by buttons and toggles and call appopriate editor functions.

Tool Picker Toggles:

```
public Toggle ViewToggle;
public Toggle MoveToggle;
public Toggle RotateToggle;
public Toggle ScaleToggle;
```

Transform Handle Mode Toggles:

```
public Toggle PivotRotationToggle;
public Toggle PivotModeToggle;
public Toggle UnitSnappingToggle;
public Toggle VertexSnappingToggle;
```

Camera Behavior Setting:

```
public Toggle AutoFocusToggle;
```

Scene Management Buttons:

```
public GameObject SaveSceneDialog;
public Button BtnNew;
public Button BtnSave;
public Button BtnSaveAs;
```

Undo Redo Buttons:

```
public Button BtnUndo;
public Button BtnRedo;
```

Play/Stop Toggle

```
public Toggle PlayToggle;
```

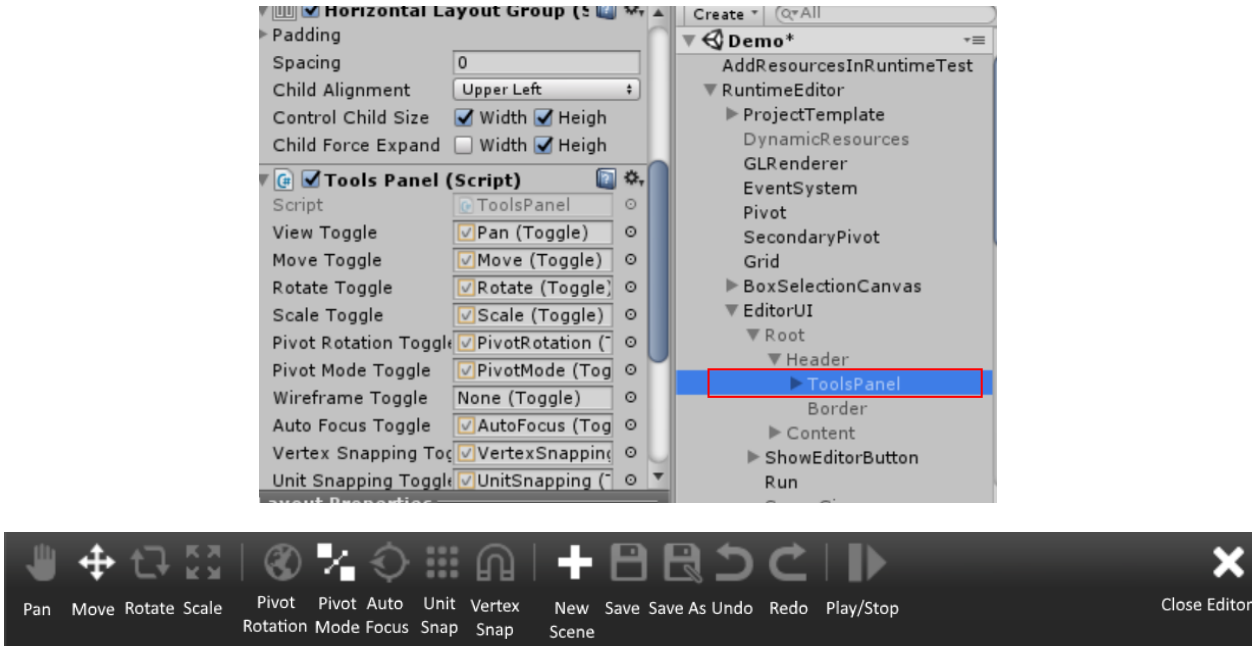

Fig.7.6 Tools Panel

7.5 Inspector Window

Located in Assets/Battlehub/RTEditor/Scripts/InspectorWindow.cs.

This script listen for RuntimeSelection changed events and instantiate GameObjectEditor or MaterialEditor depending on selected object type.

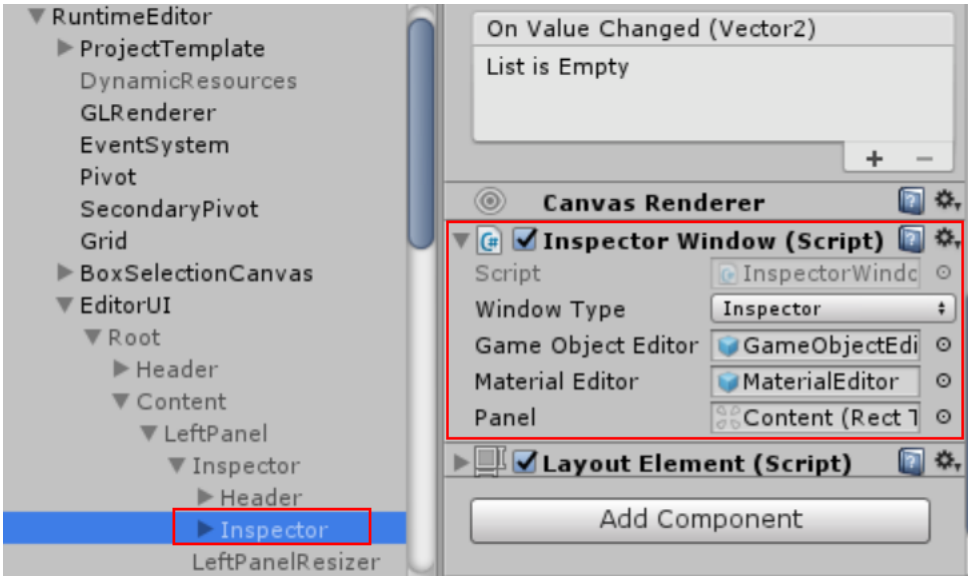

Fig.7.7 Inspector Window

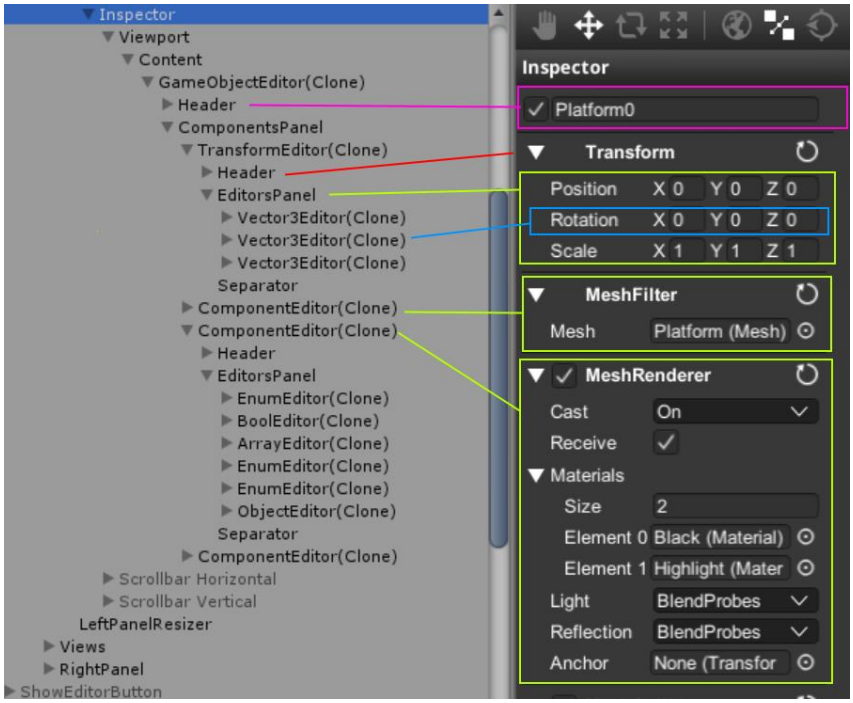

Fig.7.8 Inspector Windows Structure

7.6 GameObject Editor

Located in Assets/Battlehub/RTEditor/Scripts/Editors/GameObjectEditor.cs.

Prefab in Assets/Battlehub/RTEditor/Prefabs/Editors/GameObjectEditor

This script will instantiate component editors for selected gameobject.

7.7 Material Editor

Located in Assets/Battlehub/RTEditor/Scripts/Editors/MaterialEditor.cs.

Prefab in Assets/Battlehub/RTEditor/Prefabs/Editors/MaterialEditor

This script will instantiate property editors for selected material

7.8 Component Editor

Located in Assets/Battlehub/RTEditor/Scripts/Editors/ComponentEditor.cs.

Prefab in Assets/Battlehub/RTEditor/Prefabs/Editors/ComponentEditor

This script will instantiate property editors for component of selected gameObject

7.9 Property Editor

Located in Assets/Battlehub/RTEditor/Scripts/Editors/PropertyEditors.

Prefabs in Assets/Battlehub/RTEditor/Prefabs/Editors/PropertyEditors.

There are 14 property editors availabe out of box

- Array Editor;
- Bool Editor;
- Bounds Editor;
- Color Editor;
- Enum Editor;
- Float Editor;
- Int Editor;
- String Editor;
- Object Editor;
- QuaternionEditor;
- RangeEditor;
- Vector2Editor;
- Vector3Editor;
- Vector4Editor;

7.10 Defining which editors to use

To define which editors to use open Runtime Editor Configuration using Tools->Battlehub->Runtime Editor->Configuration menu item

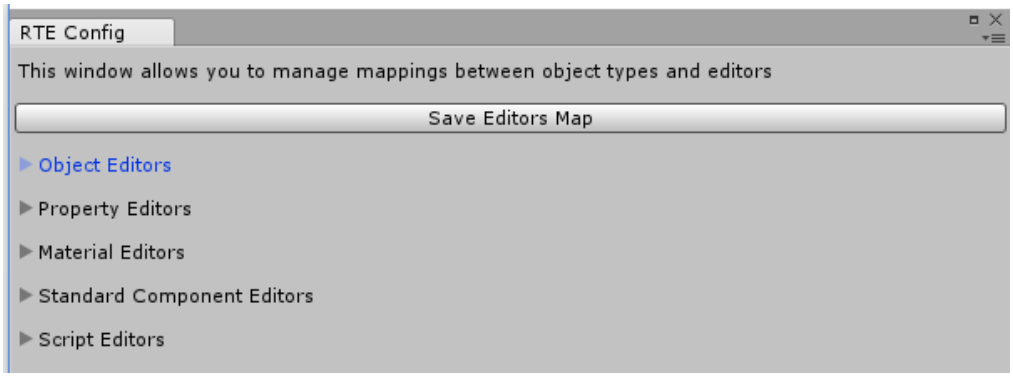

Fig.7.9 Runtime Editor config

There are 5 scections in configuration window:

- Object Editors - mapping between object types and editors
- Property Editors - mapping between property typed and editors
- Material Editors - mapping between shaders and editors
- Standard Component Editors – mapping between standard components and editors
- Script Editors – mappings between scripts and editors

Configuration window allows to specify enable or disable editor for object, property, shader, component or script

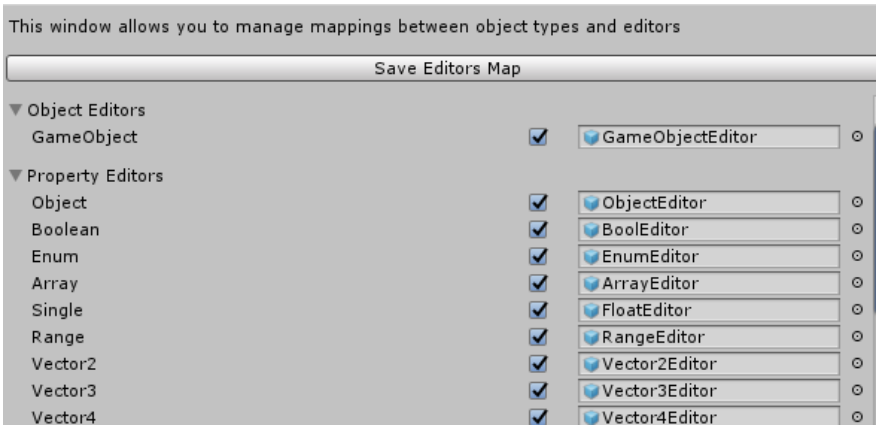

Fig.7.10 Object Editors and Property Editors section expanded

When you done with editing mappings do not forget to **click Save Editors Map** button. Editors Map will be saved to **/Battlehub/RTEditor/Prefabs/Editors/Resources/Battlehub_EditorsMap.prefab**

7.11 Selecting Component Properties

ComponentDescriptors for components are located in following folder Assets/Battlehub/RTEditor/Scripts/Editors/ComponentDescriptors.

To define your own component descriptor you have to create class and implement following interface:

```
public interface IComponentDescriptor
{
    Type ComponentType { get; }

    Type GizmoType { get; }

    object CreateConverter(ComponentEditor editor);

    PropertyDescriptor[] GetProperties(ComponentEditor editor, object converter);
}
```

ComponentType should return type of Component, CreateConverter method could optionally create converter for component properties. GetProperties method should return array of property descriptors for each required property. GizmoType could return type of Gizmo for Component (or null if there is no gizmo for component).

Here is the example implementation of MeshFilter property selector:

```
public class MeshFilterPropertySelector : IComponentPropertySelector
{
        public Type ComponentType
        {
                get { return typeof(MeshFilter); }
        }

        public Type GizmoType
        {
                get { return null; }
        }

        public object CreateConverter(ComponentEditor editor)
        {
                return null;
        }

        public PropertyDescriptor[] GetProperties(
                ComponentEditor editor, object converter)
        {
                MemberInfo sharedMeshInfo =
                        Strong.MemberInfo((MeshFilter x) => x.sharedMesh);
                return new[]
                {
                        new PropertyDescriptor("Mesh", //display name
                            editor.Component,
                            sharedMeshInfo,
                            sharedMeshInfo)
                };
        }
    }
```

7.12 HowTo: Create Component Editor

In this HowTo we will describe simple process of component editor creation by reusing existing ComponentEditor prefab and creating ComponentDescriptor. We will provide example on how to Create TransformComponentDescriptor.

Note: **TransformComponentDescriptor** *actually already exists in a project, so no need to add code from this HowTo into project*.

Step 1. Enable Transform ComponentEditor

a) Click Tools->Runtime Editor->Configuration

b) In RTE Config Window Find **Transform** component and enable it using checkbox.

c) Assign ComponentEditor prefab to EditorPrefab field

d) Click **Save Editors Map** Button

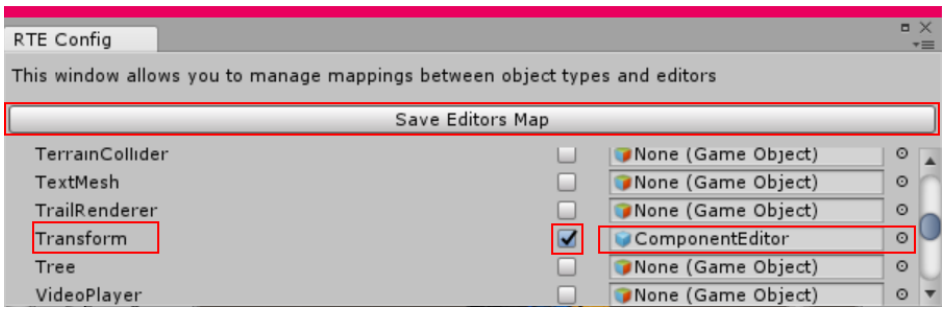

Fig. 7.11 Enabling Transform ComponentEditor

Step 2. Create new script and name it MyTransformComponentDescriptor

Step 3. Remove Start, Update methods and replace MonoBehavior with IComponentDescriptor interface

```
namespace Battlehub.RTEditor
{
    public class MyTransformComponentDescriptor : IComponentDescriptor { }
}
```

Step 4. Implement ComponentType, GizmoType and CreateConverter

```
using System;
using System.Reflection;
using UnityEngine;
using Battlehub.Utils;
namespace Battlehub.RTEditor
{
       public class MyTransformComponentDescriptor : IComponentDescriptor
       {
              public Type ComponentType { get { return typeof(Transform); }}
              public Type GizmoType{ get { return null; } }
              public object CreateConverter(ComponentEditor editor) {return null;}
              public PropertyDescriptor[] GetProperties(
                     ComponentEditor editor, object converter)
              {
              }
```

```
        }
}
```

Step 5. Implement GetProperties Method

a) Get MemberInfo using MemberInfo method of Strong class.

b) Create and return PropertyDescriptor's array

c) Done. You could start RuntimeEditor and you will see Transform Component Editor with position, rotation and scale fields

```
public PropertyDescriptor[] GetProperties(
       ComponentEditor editor, object converterObj)
{

      MemberInfo position = Strong.MemberInfo((Transform x) => x.position);
      MemberInfo rotation = Strong.MemberInfo((Transform x) => x.rotation);
      MemberInfo scale = Strong.MemberInfo((Transform x) => x.localScale);

      return new[]
          {
              new PropertyDescriptor( "Position", editor.Component, position),
              new PropertyDescriptor( "Rotation", editor.Component, rotation),
              new PropertyDescriptor( "Scale", editor.Component, scale)
          };
}
```

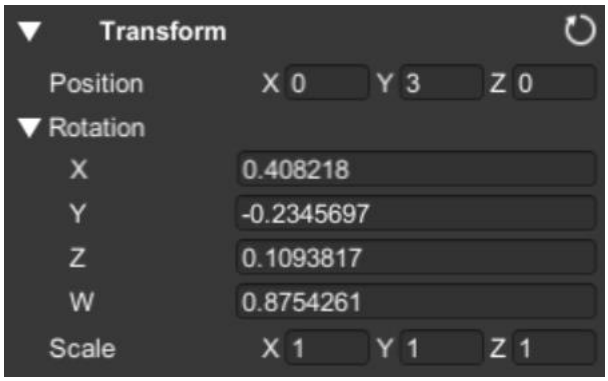

Fig 7.12 Transform Component Editor and Quaternion Rotation

Step 6. Implement Property Converter

The problem with code provided above is that rotation property returns Quaterion. But it is more convenient to edit euler angles instead of quaternion components. It is possible to replace single line of GetProperties method:

```
MemberInfo rotation = Strong.MemberInfo((Transform x) => x.eulerAngles);
```

It will solve problem, but for more complex scenarios you might want to use converter instead:

a) Create TransformPropertyConverter class

b) Implement convertation

c) Update Create Converter method

```
public object CreateConverter(ComponentEditor editor) {
      TransformPropertyConverter converter = new TransformPropertyConverter();
      converter.Component = (Transform)editor.Component;
```

```
            return converter;
        }
    public class TransformPropertyConverter
    {
        public Vector3 Rotation
        {
            get
            {
                if(Component == null)
                {
                    return Vector3.zero;
                }
                return Component.rotation.eulerAngles;
            }
            set
            {
                if (Component == null)
                {
                    return;
                }
                Component.rotation = Quaternion.Euler(value);
            }
        }

        public Transform Component
        {
            get;
            set;
        }
    }
```

Step 7. Using Converter

To use converter update GetProperties method like following:

```
public PropertyDescriptor[] GetProperties(ComponentEditor editor, object converterObj)
{
        var converter = (TransformPropertyConverter)converterObj;

        MemberInfo position = Strong.MemberInfo((Transform x) => x.position);
        MemberInfo rotation = Strong.MemberInfo((Transform x) => x.rotation);
        MemberInfo rotationConverted =
                    Strong.MemberInfo((TransformPropertyConverter x) => x.Rotation);
        MemberInfo scale = Strong.MemberInfo((Transform x) => x.localScale);

        return new[]
        {
            new PropertyDescriptor( "Position", editor.Component, position, position) ,
            new PropertyDescriptor( "Rotation", converter, rotationConverted, rotation),
            new PropertyDescriptor( "Scale", editor.Component, scale, scale)
        };
}
```

Fig 7.13 Transform Component Editor

7.13 Hierarchy Window

Located in Assets/Battlehub/RTEditor/Scripts/HierarchyWindow.cs

This script listens for RuntimeSelection and ExposeToEditor events and update gameobjects tree hierarchy.

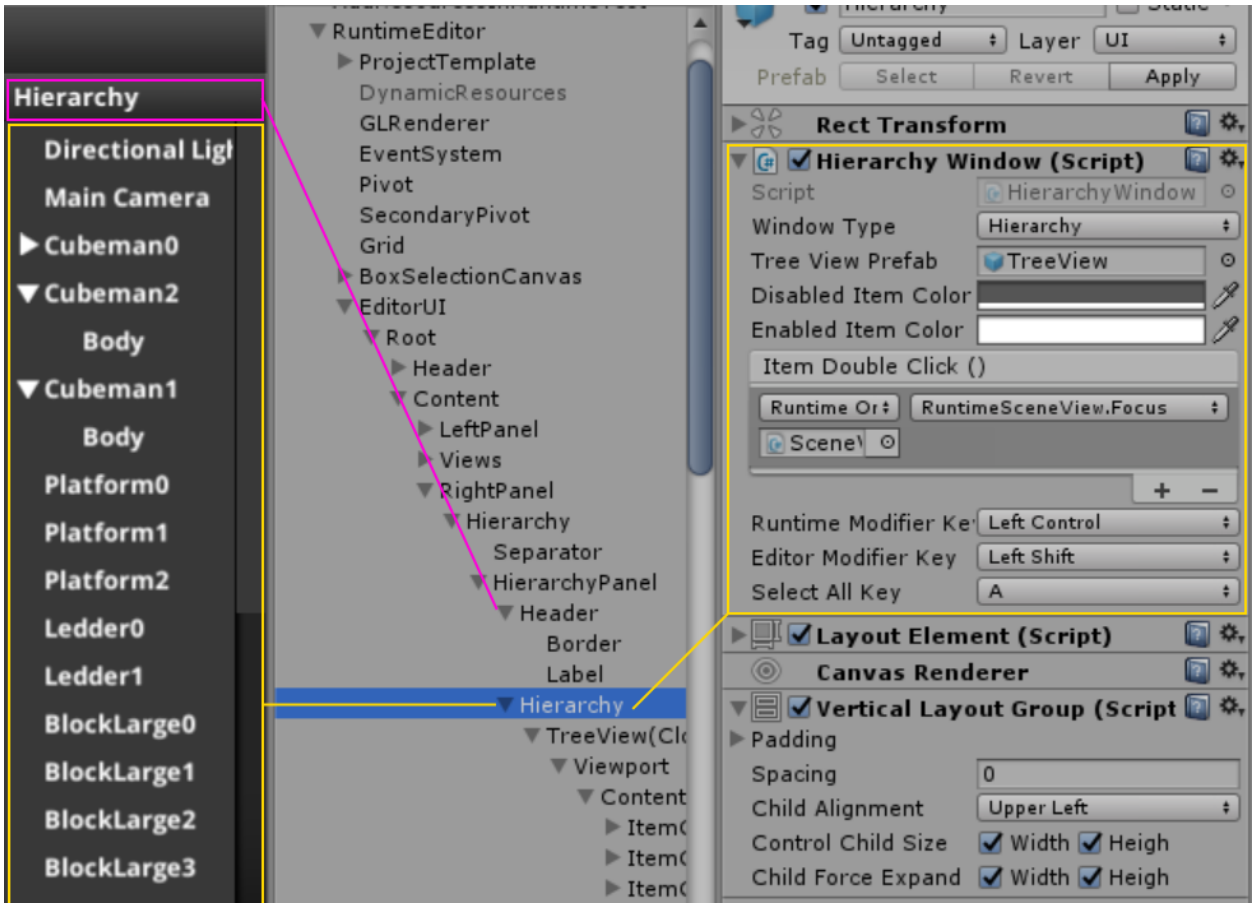

Fig 7.14 Hierarchy Window

Hierarchy Window has following properties:

- Window Type – should be set to **Hierarchy**
- TreeViewPrefab – prefab which will display hierarchy
- Disabled Item Color – Color of item when corresponding game object is inactive;
- Enabled Item Color – Color of item when corresponding game object is active
- Item Double Click event – raised when item double clicked.
- Select All Key – in conjunction with modifier key allows to select all visible elements in hierarchy.

7.14 Project Tree Window

Located in Assets/Battlehub/RTEditor/Scripts/ProjectTreeWindow.cs

Project Tree Window is the left part of Project Window. It is used to represent folders tree of project.

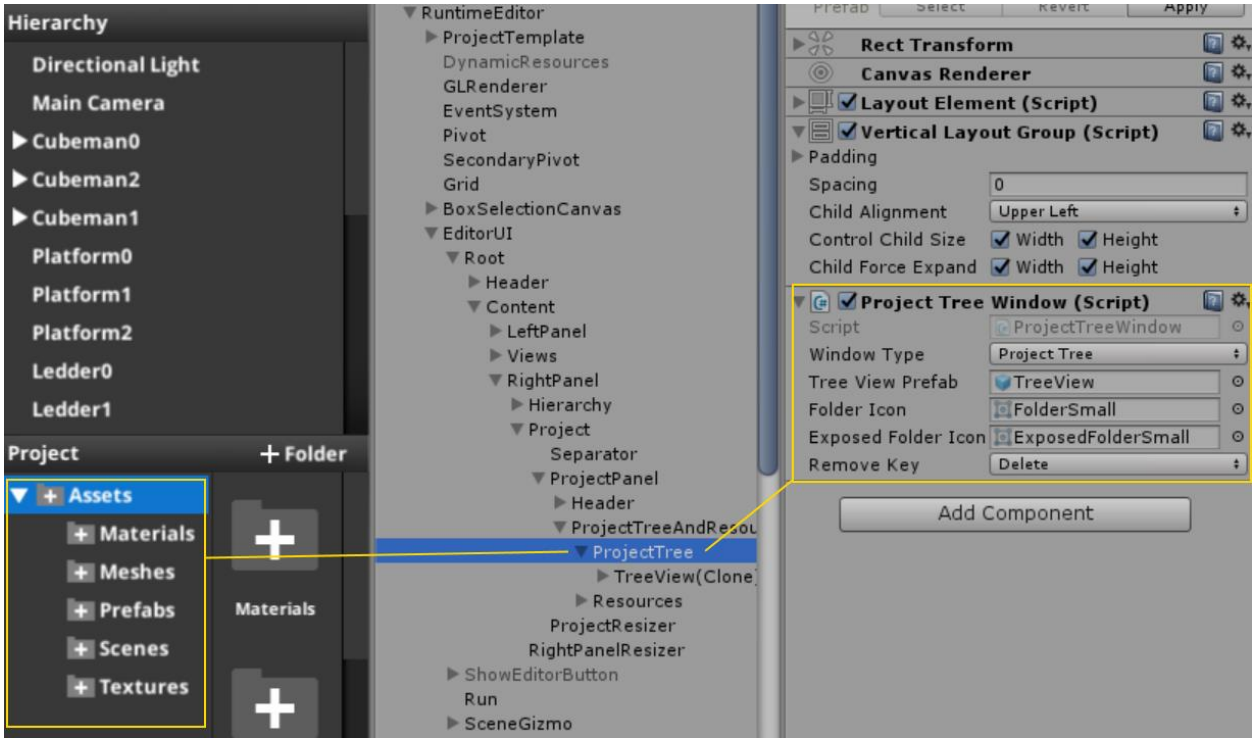

Fig 7.15 Project Tree Window

Project Tree Window has following properties:

- Window Type – should be set to **Project Tree**
- TreeViewPrefab – prefab which will display hierarchy;
- Folder Icon – Icon of folder create in runtime;
- Exposed Folder Icon – Icon of folder exposed to editor using ProjectTemplate

7.15 Project Resources Window

Located in Assets/Battlehub/RTEditor/Scripts/ProjectResourcesWindow.cs

Project Resources Window is the right part of Project Window. It is used to display contents of selected project folders.

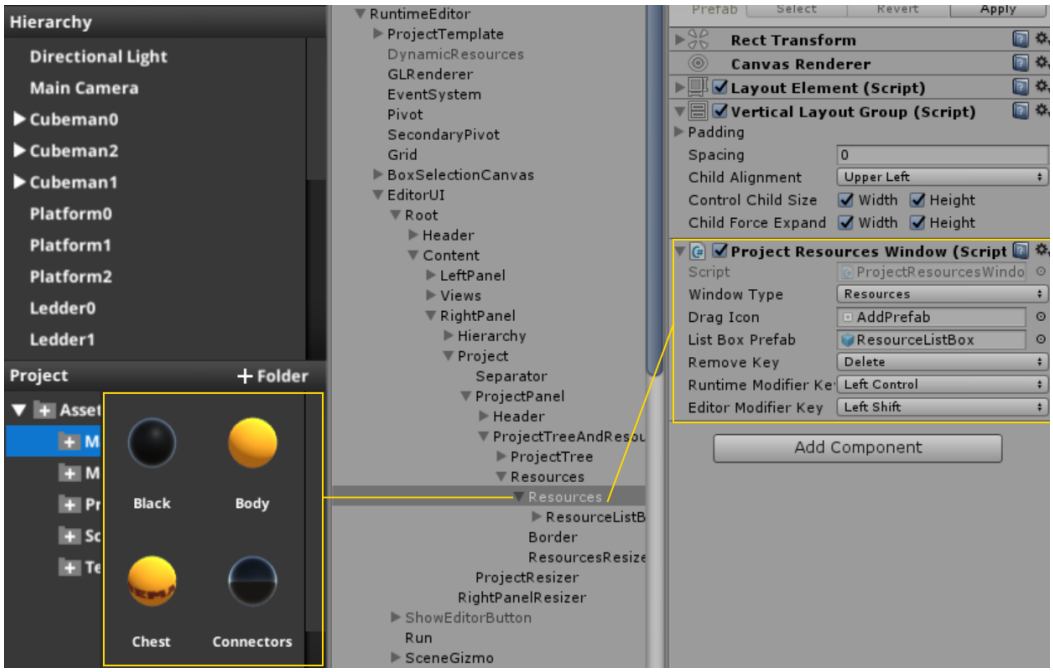

Fig 7.16 Project Resources Window

Project Resources Window has following properties:

- Window Type – should be set to **Resources**
- Drag Icon – icon which will be displayed during Resource Drag&Drop
- ListBoxPrefab – prefab which will display resources list;
- Remove Key – Key which will be used to remove resources;

8. Runtime SaveLoad

8.1 Overview

Runtime SaveLoad subsystem **located in Assets/Battlehub/RTSaveLoad** folder. It is required to save and load scenes, resource and project data. RTSaveLoad folder structure is shown if Fig.8.1

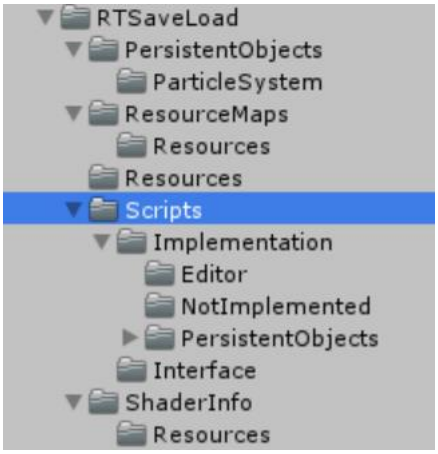

Fig.8.1 RTSaveLoad folders structure

- PersistentObjects - autogenerated datacontracts for storing unity objects from UnityEngine and other Unity assemblies.
- ResourceMaps – mappings between resources and unique identifiers. These mappings generated automatically using following menu item: **Tools->Runtime SaveLoad->Create Resource Map**.
- Resources – required by resource map generator to include build-in resources to ResourceMap
- ShaderInfo – shaderinfo files containing list of shader properties (autogenerated by Tools->Runtime SaveLoad->Create Resource Map menu item)
- Scripts
    - Interface of save load subsystem;
    - Implementation of save load subsystem;

8.2 Dependencies

Located in /Battlehub/RTSaveLoad/Scripts/Interface/Dependencies.cs. Main purpose of this class is to decouple different parts of Runtime SaveLoad subsystem from each other and from external users as well.

Serializer – class which is used for serialization / deserialization ;

Storage – save load funtionality (FileSystem by default);

Project – used to save, load, modify ProjectItems;

BundleLoader – used to load asset bundles;

ProjectManager – high level project management;

SceneManager – create, save, load scenes;

ShaderUtil – provides access to shader info;

```
public static class Dependencies
{
        public static ISerializer Serializer
        {
            get { return new Serializer(); }
        }
        public static IStorage Storage
        {
            get { return new FileSystemStorage(Application.persistentDataPath); }
        }
        public static IProject Project
        {
            get { return new Project(); }
        }
        public static IAssetBundleLoader BundleLoader
        {
            get { return new AssetBundleLoader(); }
        }

        public static IProjectManager ProjectManager
        {
            get { return Object.FindObjectOfType<ProjectManager>(); }
        }

        public static ISceneManager SceneManager
        {
            get { return Object.FindObjectOfType<RuntimeSceneManager>(); }
        }

        public static IRuntimeShaderUtil ShaderUtil
        {
            get { return new RuntimeShaderUtil(); }
        }
}
```

8.3 Serializer

Located in /Battlehub/RTSaveLoad/Scripts/Interface/ISerializer.cs. This interface contains 3 methods: Serialize, Deserialize and DeepClone

```
public interface ISerializer
{
    byte[] Serialize<TData>(TData data);
    TData Deserialize<TData>(byte[] data);
    TData DeepClone<TData>(TData data);
}
```

Implementation class is wrapper for ProtobufSerializer. Located in **/Battlehub/RTSaveLoad/Scripts/Implementation/Serializer.cs**

8.4 Storage

Located in /Battlehub/RTSaveLoad/Scripts/Interface/Storage.cs. IStorage interface contains several methods for saving, loading, renaming and deleting files. Each method has callback function as last argument. Example of usage:

```
m_storage.LoadFile("Project.meta", loadMetaCallback =>
{
       if(loadMetaCallback.Data != null)
       {
              ProjectMeta result =
                     m_serializer.Deserialize<ProjectMeta>(loadMetaCallback.Data);
       }
}
```

FileSystemStorage is default implementation of IStorage interface.

8.5 ResourceMap

It is crucial to create and update resource map to make Runtime SaveLoad subsystem work correctly

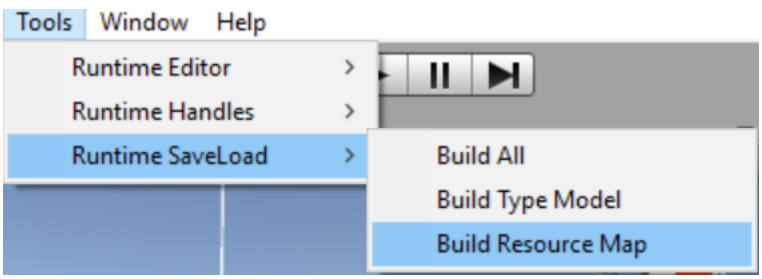

Fig.8.2 Runtime SaveLoad menu

Build Resource Map menu item creates or updates mapping between objects (prefabs, resources, special scene objects) and unique identifiers. These identifiers are required to make Save&Load subsystem work correctly. Project's Resource map will be saved to **Battlehub_ResourceMap.prefab** located in **Assets/Battlehub/RTSaveLoad /ResourceMaps/Resources/** folder**. Create Resource Map** menu item will also create or update resource maps for each asset bundle in project. Resource maps for asset bundles will be saved outside of **Resources** folder. Name of resource map for asset bundle has following format: **ResourceMap_<bundle name>_<guid>** where <bundle name> is name of asset bundle and <guid> is string representation of arbitrary System.Guid

Texture shown in Fig.8.3 will be included to resource map for "bundledemo" asset bundle during next resource map update.

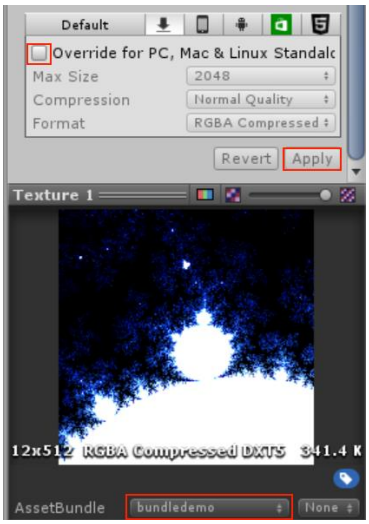

Fig.8.3 Texture added to AssetBundle

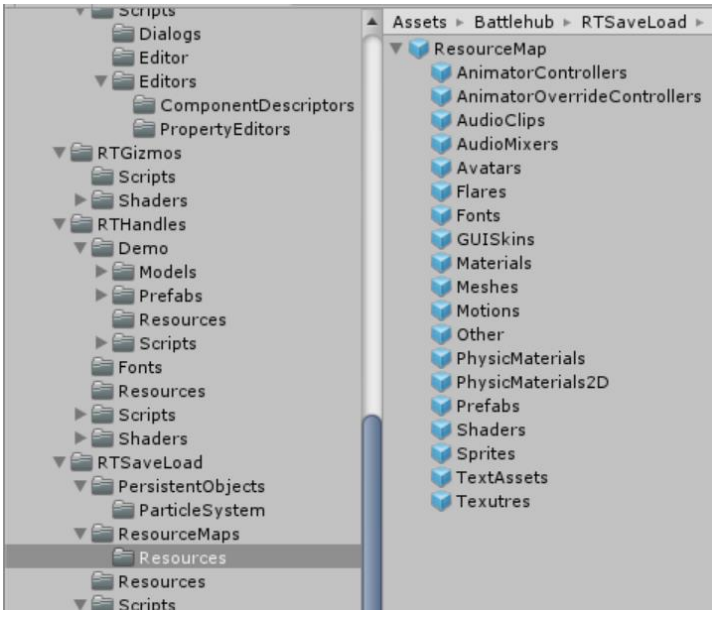

Fig.8.4 ResourceMap prefab

Each child of resource map has ResourceGroup script attached

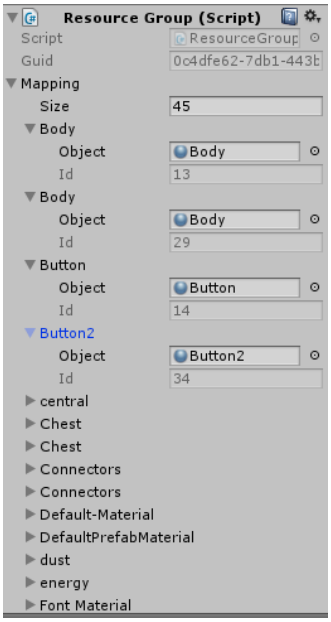

Fig.8.5 Materials Resource Group

Readonly Id field shows unique key for each object.

If for some reason resorce map will be lost, RTSaveLoad will be unable to read files created using this resource map.

8.6 IdentifiersMap

Located in Battlehub/RTSaveLoad/Scripts/Implementation/ResourceMap.cs

Identifiers map use ResourceMap to create mapping between identifiers obtained from objects using Object.GetInstanceID() method and persistent unqiue identifiers stored in ResourceMap. You may ask why these persistent unique identifiers required at all? The answer is simple: because Object.GetInstanceID will not return same identifier for same object each time you run your application. Identifiers will be different each time you run application and it will be impossible to load dependencies for saved objects during next application run.

Instance property will return instance of Identifiers map

```
public static IdentifiersMap Instance { get; set; }
```

GetMappedInstanceID will return unique persistent identifier for object

```
public long GetMappedInstanceID(Object obj)
```

8.7 Runtime TypeModel

RTSaveLoad subsystem use protobuf-net for serialization and deserialization. Due to unity scripting restrictions protobuf runtime type model need to be precomplied before using at runtime. You need to build runtime type model each time you add custom serializable type to project. To do this click **Tools->RTSaveLoad->Build Type Model** menu item.

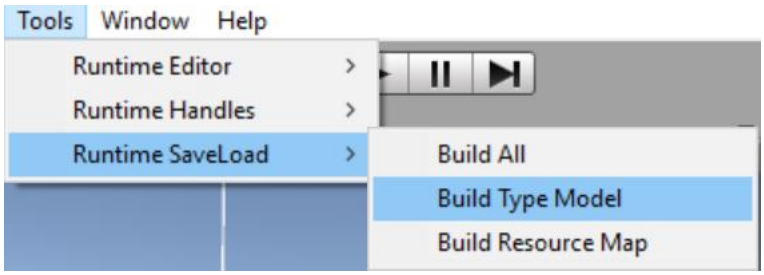

Fig. 8.6 Build Type Model menu item

When precompilation will be completed RTTypeModel.dll will be moved to Deps folder and will be used in runtime for serialization and deserialization.

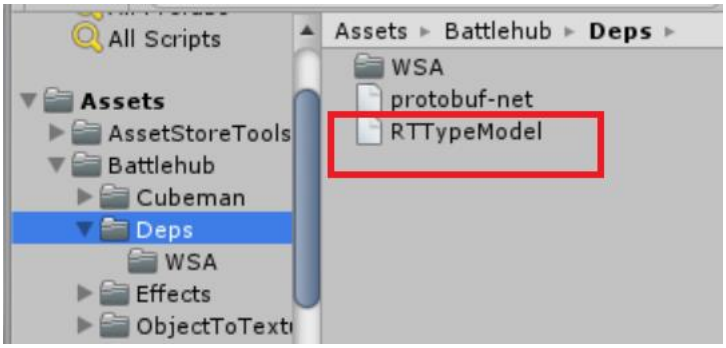

Fig. 8.7 RTTypeModel.dll

8.8 HowTo: Add Custom Type

```
[System.Serializable]
[ProtoBuf.ProtoContract(ImplicitFields = ProtoBuf.ImplicitFields.AllPublic)]
public class MyType
{
    public int Data;
}

public class MyPersistentBehavior : MonoBehaviour
{
    public MyType[] Data;
}
```

8.9 PersistentDescriptor

Located in /RTSaveLoad/Scripts/Implementation/PersistentDescriptor.cs

Following data structure is used to describe objects and components hierarchy:

```
[Serializable]
 public class PersistentDescriptor
 {
        public long InstanceId;
        public string TypeName;
        public PersistentDescriptor Parent;
        public PersistentDescriptor[] Children;
        public PersistentDescriptor[] Components;
```

8.10 PersistentData

Located in /RTSaveLoad/Scripts/Implementation/PersistentData.cs

This is base class for all persistent objects. Most of persistent objects located in PersistentObjects folder:

```
    [Serializable]
    public abstract partial class PersistentData
    {

        public long InstanceId { get; }
        public virtual void ReadFrom(object obj)
        public virtual object WriteTo(object obj, Dictionary<long, UnityObject>
objects /*needed to resolve references*/)

        public virtual void FindDependencies<T>(Dictionary<long, T> dependencies,
        Dictionary<long, T> objects, bool allowNulls);

        protected virtual void GetDependencies(Dictionary<long, UnityObject>
        dependencies, object obj)
    }
```

In derived classes ReadFrom method read values from obj, WriteTo method write values to obj. FindDependencies method find dependencies from objects dictionary and populate dependencies dictionary. GetDependencies method retreives dependencies from obj and populate dependencies dictionary.

8.11 PersistentScene

Located in /RTSaveLoad/Scripts/Implementation/PersistentScene.cs

Following data structure is used to store scene:

```
[Serializable]
  public class PersistentScene
  {
      /// <summary>
      /// hierarchy stored in this array
      /// </summary>
      public PersistentDescriptor[] Descriptors;

      /// <summary>
      /// data for each game object and component stored in this array
      /// </summary>
      public PersistentData[] Data;

      public static void InstantiateGameObjects(PersistentScene scene)

      public static PersistentScene CreatePersistentScene(params Type[] ignoreTypes)
  }
```

CreatePersistentScene creates PersistentScene object for current scene.

InstantiateGameObjects creates GameObjects using data stored in PersistentScene.

8.12 ISceneManager

Located in /RTSaveLoad/Scripts/Interface/IProjectManager.cs

Events:

```
event EventHandler<ProjectManagerEventArgs> SceneCreated;
event EventHandler<ProjectManagerEventArgs> SceneSaving;
event EventHandler<ProjectManagerEventArgs> SceneSaved;
event EventHandler<ProjectManagerEventArgs> SceneLoading;
event EventHandler<ProjectManagerEventArgs> SceneLoaded;
```

Active Scene ProjectItem:

```
ProjectItem ActiveScene
{
    get;
}
```

Check if scene Exists:

```
void Exists(ProjectItem scene, ProjectManagerCallback<bool> callback);
```

Save, Load, Create Scene:

```
void SaveScene(ProjectItem scene, ProjectManagerCallback callback);
void LoadScene(ProjectItem scene, ProjectManagerCallback callback);
void CreateScene();
```

Example Usage:

```
m_sceneManager = Dependencies.SceneManager;
m_sceneManager.LoadScene(m_sceneManager.ActiveScene, () => {});
m_sceneManager.SaveScene(m_sceneManager.ActiveScene, () => {});
```

8.13 IProjectManager

Located in /RTSaveLoad/Scripts/Interface/IProjectManager.cs

Events:

```
event EventHandler ProjectLoading;
event EventHandler<ProjectManagerEventArgs> ProjectLoaded;
event EventHandler<ProjectManagerEventArgs> BundledResourcesAdded;
event EventHandler<ProjectManagerEventArgs> DynamicResourcesAdded;
```

Loaded Project:

```
ProjectItem Project
{
   get;
}
```

Returns false for Scene GameObjects and true for Resources and Prefabs:

```
bool IsResource(UnityObject obj);
```

Get ID for Object. **ID** is just wrapper of persistent unique identifier. If you will need to create your own implementation of IProjectManager interface it allow you to use any type of persistent unique identifier (int, long, Guid):

```
ID GetID(UnityObject obj);
```

Load Project:

```
void LoadProject(ProjectManagerCallback<ProjectItem> callback);
```

Add resources to **folder** from asset bundle with **bundleName** using **filter** function to filter out objects which are not required:

```
void AddBundledResources(ProjectItem folder, string bundleName,
      Func<UnityObject, string, bool> filter,
      ProjectManagerCallback<ProjectItem[]> callback);
```

Add resource with **assetName** from asset bundle with **bundleName** to **folder**:

```
void AddBundledResource(ProjectItem folder, string bundleName, string assetName,
      ProjectManagerCallback<ProjectItem[]> callback);
```

Add resource of type **T** with **assetName** from asset bundle with **bundleName** to **folder**:

```
void AddBundledResource<T>(ProjectItem folder, string bundleName,
      string assetName, ProjectManagerCallback<ProjectItem[]> callback);
```

Add resource with **assetName** of **assetType** from asset bundle with **bundleName** to **folder**:

```
void AddBundledResource(ProjectItem folder, string bundleName,
      string assetName, Type assetType,
      ProjectManagerCallback<ProjectItem[]> callback);
```

Add resources with **assetNames** from asset bundle with **bundleName** to **folder**:

```
void AddBundledResources(ProjectItem folder, string bundleName,
	string[] assetNames, ProjectManagerCallback<ProjectItem[]> callback);
```

Add resources with **assetNames** of **assetTypes** from asset bundle with **bundleName** to **folder** using filter **function** to filter out objects which are not required:

```
void AddBundledResources(ProjectItem folder, string bundleName,
	string[] assetNames, Type[] assetTypes,
	Func<UnityObject, string, bool> filter,
	ProjectManagerCallback<ProjectItem[]> callback);
```

Add dynamic resource (**obj** created in runtime) to **folder**:

```
void AddDynamicResource(ProjectItem folder, UnityObject obj,
	ProjectManagerCallback<ProjectItem[]> callback);
```

Add dynamic resources (**objects** created in runtime) to **folder**:

```
void AddDynamicResources(ProjectItem folder, UnityObject[] objects,
	ProjectManagerCallback<ProjectItem[]> callback);
```

Add dynamic resource (**obj** created in runtime) to **folder** using filter function to filter out objects which are not required and **includeDepenencies** if required:

```
void AddDynamicResource(ProjectItem folder, UnityObject obj,
	bool includingDependencies, Func<UnityObject, bool> filter,
	ProjectManagerCallback<ProjectItem[]> callback);
```

Add dynamic resources (**objects** created in runtime) to **folder** using filter function to filter out objects which are not required and **includeDepenencies** if required:

```
void AddDynamicResources(ProjectItem folder, UnityObject[] objects,
	bool includingDependencies, Func<UnityObject, bool> filter,
	ProjectManagerCallback<ProjectItem[]> callback);
```

Create Folder with **name** inside of **parent** folder:

```
void CreateFolder(string name, ProjectItem parent,
ProjectManagerCallback<ProjectItem> callback);
```

Save objects:

```
void SaveObjects(ProjectItemObjectPair[] itemObjectPairs,
	ProjectManagerCallback callback);
```

Get or create objects from **folder**:

```
void GetOrCreateObjects(ProjectItem folder,
	ProjectManagerCallback<ProjectItemObjectPair[]> callback);
```

Get or create objects from **folders**:

```
void GetOrCreateObjects(ProjectItem[] projectItems,
	ProjectManagerCallback<ProjectItemObjectPair[]> callback);
```

Duplicate **projectItems**:

```
void Duplicate(ProjectItem[] projectItems,
	ProjectManagerCallback<ProjectItem[]> callback);
```

Rename **projectItems**:

```
void Rename(ProjectItem projectItem, string newName,
       ProjectManagerCallback callback);
```

Move **projectItems** to folder:

```
void Move(ProjectItem[] projectItems, ProjectItem folder,
      ProjectManagerCallback callback);
```

Delete **projectItems**:

```
void Delete(ProjectItem[] projectItems, ProjectManagerCallback callback);
```

Do not save & load types:

```
void IgnoreTypes(params Type[] types)
```

8.14 IAssetBundleLoader

Located in /RTSaveLoad/Scripts/Interface/IAssetBundleLoader.cs

You may want to provide your own implementation of this interface. Currently there is following implementation:

```
public class AssetBundleLoader : IAssetBundleLoader
{
        public void Load(string name, AssetBundleEventHandler callback)
        {
            if(!System.IO.File.Exists(
                    Application.streamingAssetsPath + "/" + name))
            {
                callback(name, null);
                return;
            }
            AssetBundle bundle = AssetBundle.LoadFromFile(
                    Application.streamingAssetsPath + "/" + name);
            if(callback != null)
            {
                callback(name, bundle);
            }
        }
}
```

8.15 HowTo: Add Dynamic Resource

Example Located in /RTSaveLoad/Scripts/Demo/AddResourcesTest.cs

1) Add usings

```
using UnityEngine;
using Battlehub.RTCommon;
using System.Collections.Generic;

using UnityObject = UnityEngine.Object;
```

1) Get reference to IProjectManager

```
IProjectManager m_projectManager = Dependencies.ProjectManager;
```

2) Get root folder

```
ProjectItem rootFolder = m_projectManager.Project;
```

3) Create objects and add them to objects list

```
List<UnityObject> objects = new List<UnityObject>();

Material material = new Material(Shader.Find("Standard"));
material.color = Color.yellow;

Mesh mesh = RuntimeGraphics.CreateCubeMesh(Color.white, Vector3.zero, 1);
mesh.name = "TestMesh";

GameObject go = new GameObject();
MeshRenderer renderer = go.AddComponent<MeshRenderer>();
MeshFilter filter = go.AddComponent<MeshFilter>();

go.name = "TestGO";
renderer.sharedMaterial = material;
filter.sharedMesh = mesh;

objects.Add(material);
objects.Add(mesh);
objects.Add(go);
```

4) Call AddDynamicResources method

```
m_projectManager.AddDynamicResources(rootFolder, objects.ToArray(),
    addedItems =>
{
    for (int i = 0; i < objects.Count; ++i)
    {
        Destroy(objects[i]);
    }
});
```

5) Result

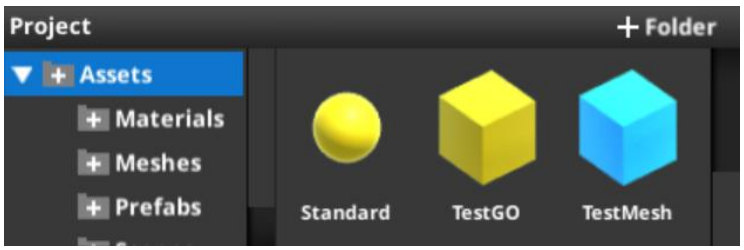

Fig.8.8 Programmatically created objects added to Project

8.16 HowTo: Add Bundled Resource

Example Located in /RTSaveLoad/Scripts/Demo/AddResourcesTest.cs

1) Add usings

```
using UnityEngine;
using Battlehub.RTCommon;
using System.Collections.Generic;

using UnityObject = UnityEngine.Object;
```

2) Get reference to IProjectManager

```
IProjectManager m_projectManager = Dependencies.ProjectManager;
```

3) Get root folder

```
ProjectItem rootFolder = m_projectManager.Project;
```

4) Asset bundles create automatically using during create of resource maps and saved to StreamingAssets folder

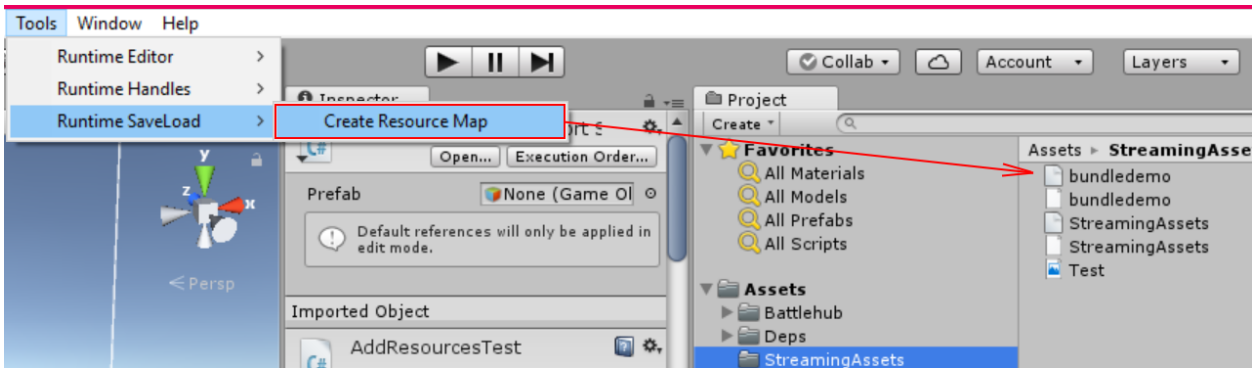

Fig.8.9 Asset bundles created automatically

5) Call AddBundledResources method

```
m_projectManager.AddBundledResources(rootFolder, "bundledemo",
        (obj, assetName) =>
        {
            return true;
        },
        addedItems =>
        {
            for (int i = 0; i < addedItems.Length; ++i)
            {
                Debug.Log(addedItems[i].ToString() + " added");
            }
        });
}
```

6) Result

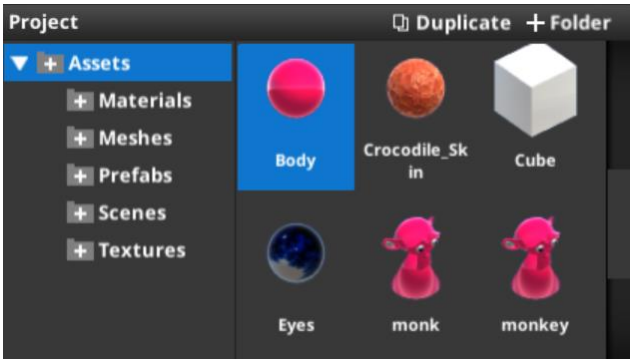

Fig.8.10 Resources from asset bundle “bundledemo” added to project

8.17 Persistent Ignore

Located in /RTSaveLoad/Scripts/Implementation/Persistent Ignore.cs

Attach this script to object you want to be ignored by Save & Load system

9. UIControls

There are two major UI controls included in package: TreeView and ListBox;

These controls implements drag & drop, databinding, selection operations and events and are highly customizable. There are also two base clases ItemsControl and ItemContainer which can be used to implement your own items control.

9.1 ItemsControl

Located in Assets/Battlehub/UIControls/Scripts/ItemsControl.cs.

Base class for TreeView and ListBox

```
public class ItemsControl<TDataBindingArgs> : MonoBehaviour, IPointerDownHandler,
       IDropHandler where TDataBindingArgs : ItemDataBindingArgs, new()
    {
        //Drag & Drop Events
        public event EventHandler<ItemDragArgs> ItemBeginDrag;
        public event EventHandler<ItemDropArgs> ItemDrop;
        public event EventHandler<ItemDragArgs> ItemEndDrag;

        //Raise when data for ItemContainer required
        public event EventHandler<TDataBindingArgs> ItemDataBinding;

        //Selection Changed
        public event EventHandler<SelectionChangedEventArgs> SelectionChanged;

        //Item Removed
        public event EventHandler<ItemsRemovedArgs> ItemsRemoved;

        //Key bindings
        public KeyCode MultiselectKey = KeyCode.LeftControl;
        public KeyCode RangeselectKey = KeyCode.LeftShift;
        public KeyCode RemoveKey = KeyCode.Delete;

        //Is Drag & Drop allowed
        public bool CanDrag = true;


        //GameObject with ItemsContainer script (or with ItemsContainer derived class)
        [SerializeField]
        private GameObject ItemContainerPrefab;

        //Layot Panel
        public Transform Panel;

        //Raycasting Camera (used if Canvas.RenderMode == RenderMode.WorldSpace)
        public Camera Camera;

        //Scroll Speed (when item dragged out of ScrollViewer content area)
        public float ScrollSpeed = 100;

        //Set of data items
        public IEnumerable Items { get; set; }

        //items count
        public int ItemsCount { get; }
```

```
    //Selected Items count
    public int SelectedItemsCount { get; }

    //Set of selected items
    public IEnumerable SelectedItems { get; set;}

    //First Selected item
    public object SelectedItem { get; set;}

    //Index of first Selected item (-1 if no items selected)
    public int SelectedIndex { get; set;}

    //Get index of data item
    public int IndexOf(object obj)

    //Get Item Container for dataitem
    public ItemContainer GetItemContainer(object obj)

    //Get Item Container for last dataitem
    public ItemContainer LastItemContainer()

    //Get Item Container by index
    public ItemContainer GetItemContainer(int siblingIndex)

    //Add data item (if you have a collection of items use Items property instead)
    public ItemContainer Add(object item)

    //Insert data item (if you have a collection of items use Items property instead)
    public ItemContainer Insert(int index, object item)

    //Remove data item
    public void Remove(object item)

    //Remove data item by index
    public void RemoveAt(int index)
```

9.2 ItemContainer

Located in Assets/Battlehub/UIControls/Scripts/ItemsContainer.cs

Base class for data item representation component (for TreeViewItem and for ListBoxItem).

```
[RequireComponent(typeof(RectTransform), typeof(LayoutElement))]
public class ItemContainer : MonoBehaviour, IPointerDownHandler, IPointerUpHandler,
        IPointerEnterHandler, IPointerExitHandler, IBeginDragHandler,
        IDragHandler, IDropHandler, IEndDragHandler
{

    //Is Drag & Drop allowed?
    public bool CanDrag = true;

    //Events
    public static event EventHandler Selected;
    public static event EventHandler Unselected;
    public static event ItemEventHandler PointerDown;
    public static event ItemEventHandler PointerUp;
    public static event ItemEventHandler PointerEnter;
    public static event ItemEventHandler PointerExit;
    public static event ItemEventHandler BeginDrag;
```

```
    public static event ItemEventHandler Drag;
    public static event ItemEventHandler Drop;
    public static event ItemEventHandler EndDrag;

    //ItemContainer's LayoutElement
    public LayoutElement LayoutElement { get; }

    //ItemContainer's RectTransform
    public RectTransform RectTransform { get; }

    //Is Item Container selected
    public virtual bool IsSelected { get; set; }

    //Data Item bound to Item Container
    public object Item { get; set; }
```

9.3 ItemDropMarker

Located in Assets/Battlehub/UIControls/Scripts/ItemDropMarker.cs

Item Drop Marker is used to highlight item drop location.

ItemDropMarker could be in one of the states specified by ItemDropAction enum.

```
public enum ItemDropAction
{
    None,
    SetLastChild,
    SetPrevSibling,
    SetNextSibling
}
```

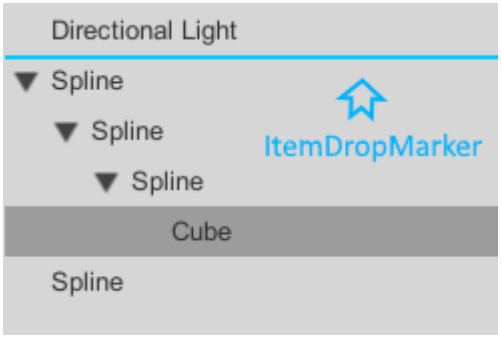

Fig.11 ItemDropMarker

9.4 ListBox

Located in Assets/Battlehub/UIControls/Scripts/ListBox.cs

Prefab Assets/Battlehub/UIControls/Prefabs/ListBox.prefab

See **Assets/Battlehub/RTEditor/Scripts/RuntimePrefabs.cs** for usage example.

ListBox has same funtionality as ItemsControl and defined as following:

```
public class ListBox : ItemsControl { }
```

ListBox supports multiselection, drag & drop and delete item's operation. But in prefabs panel all these functions disabled.

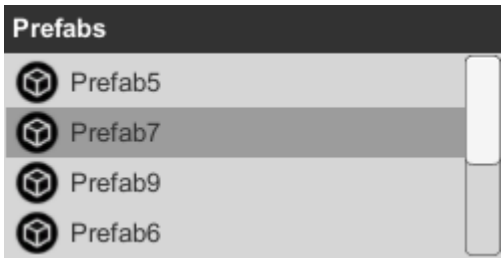

Fig.12 ListBox

9.5 ListBoxItem

Located in Assets/Battlehub/UIControls/Scripts/ListBoxItem.cs

Prefab Assets/Battlehub/UIControls/Prefabs/ ListBoxItem.prefab

ListBoxItem use Toggle to implement selected and unselected visual state

```
public override bool IsSelected
{
        get { return base.IsSelected; }
        set
        {
            if (base.IsSelected != value)
            {
                m_toggle.isOn = value;
                base.IsSelected = value;
            }
        }
 }

 protected override void AwakeOverride()
 {
        m_toggle = GetComponent<Toggle>();
        m_toggle.interactable = false;
        m_toggle.isOn = IsSelected;
  }
```

9.6 TreeView

Located in Assets/Battlehub/UIControls/Scripts/TreeView.cs

Prefab Assets/Battlehub/UIControls/Prefabs/TreeView.prefab

TreeView supports multiselection, drag & drop and delete item's operation

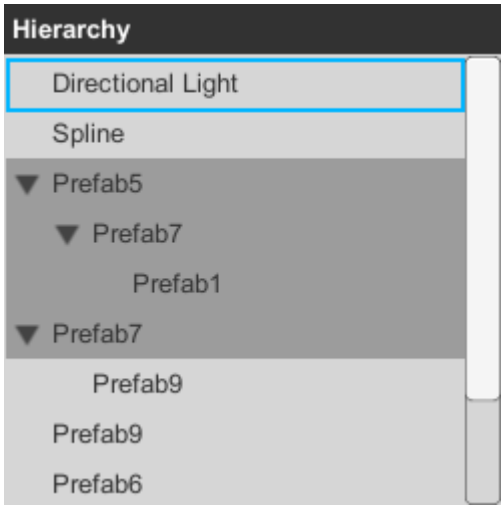

Fig.12 TreeView

```
public class TreeView : ItemsControl<TreeViewItemDataBindingArgs>
{
        //Raised when item is expanded
        public event EventHandler<ItemExpandingArgs> ItemExpanding;

        //Indent between treeview levels
        public int Indent = 20;

        //Add child data item to tree view
        public void AddChild(object parent, object item)

        //Change parent of item
        public void ChangeParent(object parent, object item)

        //Expand TreeViewItem
        public void Expand(TreeViewItem item)

        //Collapse TreeViewItem
        public void Collapse(TreeViewItem item)
```

9.7 TreeViewItem

Located in Assets/Battlehub/UIControls/Scripts/TreeViewItem.cs

Prefab Assets/Battlehub/UIControls/Prefabs/TreeViewItem.prefab

```
public class TreeViewItem : ItemContainer
{
    //Raised when item's parent changed
    public static event EventHandler<ParentChangedEventArgs> ParentChanged;

    //Accumulated indent
    public int Indent { get; }

    //Parent TreeViewItem
    public TreeViewItem Parent { get; set; }

    public override bool IsSelected { get; set; }

    //Whether tree view item can be expanded (if true expander arrow is visible)
    public bool CanExpand { get;set;}

    //Is tree view item expanded
    public bool IsExpanded { get; set; }

    //Whether tree view item has children
    public bool HasChildren { get; }

    //Is tree view item is descendant of other tree view item;
    public bool IsDescendantOf(TreeViewItem parent)

    //Returns first child if exists
    public TreeViewItem FirstChild()

    //Returns next child if exists
    public TreeViewItem NextChild(TreeViewItem currentChild)

    //Returns last child
    public TreeViewItem LastChild()
```

Limitations and Issues

This Runtime Editor is very basic, no windows available in unity editor, etc. There is lack of following important functions:

- **Touch support;**
- **Main menu;**
- **Add Component button.**
- **Dockable panels;**

Furthermore API (if can be called so) is completely different from API located in UnityEditor namespace.

To conclude this is not at all replacement of original unity editor. Please do not expect too much.

Support

If you have any questions, suggestions, you want to talk or you have some issues please send mail to Vadim.Andriyanov@outlook.com or Battlehub@outlook.com.