Unity and C#

Event Methods

- Start()
  - Runs once when the component is initialized
- Update()
  - Runs once every frame update
- OnGUI()
  - Draws the GUI layer each frame update
- OnCollisionEnter(Collision c)
  - Is called when the object enters a collision

Public Attributes

- Can be accessed by all scripts referencing that behavior
- Can be modified on the Unity screen as part of a GameObject's component properties, making them more useful than usual for coding because:
  - It allows you to modify one object's values without affecting any other instance of that script
  - You do not need to open the editor to quickly test values during development
  - Can also help build for mod development in the future

Behavior Script tricks

- gameobject will always reference what the script is attached to
- Is treated as a part of the object. Things such as ‘transform’ will reference the transform of the game object it is attached to
- Camera.main will always reference the main camera in the scene
- Debug.Log(“string”) will print out messages in the console field. Great for tracking down bugs and confirming that events are being fired off

Some Major Classes

- Time — deals with how time passes
- GUI — handles GUI elements placed on the screen
- Transform — handles how an object occupies simulation space
- Physics — handles the in-engine physics such as collision hits
- Vector3 — 3D vector data and manipulation
- Input — handles input from things such as the keyboard, mouse and game pad
- Camera — handles the cameras in the scene

(GameObject).GetComponent<Script>()

- Will return the reference to the given GameObject's desired behavior script
- If the chosen component is not attached to the GameObject this will return null instead
- You will likely use this method very often

Transformations

- Accessed via the ‘transform’ field of a GameObject
- All GameObjects have a transform component, including empty GameObjects
- Most common usage is to manipulate scale, position and rotation
  - Position can be changed through either the ‘position’ attribute (Vector3) or by using using the Translate method
  - Rotation can be changd through the ’rotation’ attribute (Quaternion) or by using the Rotate method.
  - Scale is usually changed through the ‘localScale’ attribute (Vector3)

Rotating Around a Point

- A useful method from: http://answers.unity3d.com/questions/532297/rotate-a-vector-around-a-certain-point.html

```
Vector3 RotateAroundPivot(Vector3 point, Vector3 pivot, Vector3 angles)
{
    Vector3 dir = point - pivot;                // get point direction relative to pivot
    dir = Quaternion.Euler(angles) * dir;       // rotate it
    point = dir + pivot;                        // calculate rotated point
    return point;
}
```

Creating Primitives

- All Unity primitives can be generated via a method found in the GameObject Class
- Example:
  - GameObject g = GameObject.CreatePrimitive(PrimitiveType.Cylinder);

Loading Textures

- Step 1: Place the image you want to use in the ‘Resources’ folder in assets
- Step 2: load the texture
  - Texture2D textureName = (Texture2D)Resources.Load(“imgName");
- Step 3: Use texture
  - Examples:
    - GUI elements can use them as an argument for some methods
    - Assign as the texture of a Material instance

Casting a Ray

```
GameObject focused = null;
Vector3 mouse = Input.mousePosition;
 mouse.z = 0;
 RaycastHit hit;
 Ray r = Camera.main.ScreenPointToRay(mouse);
 if (Physics.Raycast(r, out hit))
 {
     focused = hit.transform.gameObject;
 }
 else
 {
     focused = null;
 }
```

Determining the Time Between Update calls

- Time.deltaTime
- Useful for timers and keeping movement smooth
- Timer:
  - timerValue += Time.deltaTime;
  - Place that in the update method
- Smooth Movement:
  - Transform.position += new Vector3(0,5,0)*Time.deltaTime;
  - Makes your movement calculations based on time in seconds rather than the frame rate (prevents dips and jerkiness if your frames slow down)

Deforming a 3D Mesh

- To adjust a mesh you first need a reference to the MeshFilter Component of the GameObject and access the ‘mesh’ attribute
- In this case we will focus on the ‘vertices’ and ‘triangles’ attributes of the mesh:
  - Vertices: Array of Vector3 instances
  - Triangles: Array of integer pointers. Each triangle consists of 3 integers arranged in order. Each of these integers represents a corner of a triangle and the value of this integer denotes the index of the corresponding vertex in the vertices array.
- Changing the position of a Vector3 element in the vertices array will result in the vertex moving and the triangles adjusting to accommodate the change.
- However, you need to make a separate reference of the array an then push that reference back into the vertices attribute after you change things to make this work

Overlap Spaces

- Part of the Physics class
- In terms of coding they are similar to that of a raycast.
- These spaces serve as effectively single-use Colliders and as such their dimensions are similar to that of their corresponding collider shape (box, sphere, capsule, etc.)

Oculus VR and Unity

- Standard GUI will not work, must treat it as a 3D object in the scene by moving the GUI canvas into the scene proper or using 3D objects as custom UI elements
- Additionally, the camera will not move via translation/position adjustments. To move an Oculus-controlled camera you must make the camera a child of another GameObject and move this parent object instead