# Blitzkrieg: Unity Overview

CMSC425.01 Spring 2019

Find your name/group and sit at that table

# Administrivia

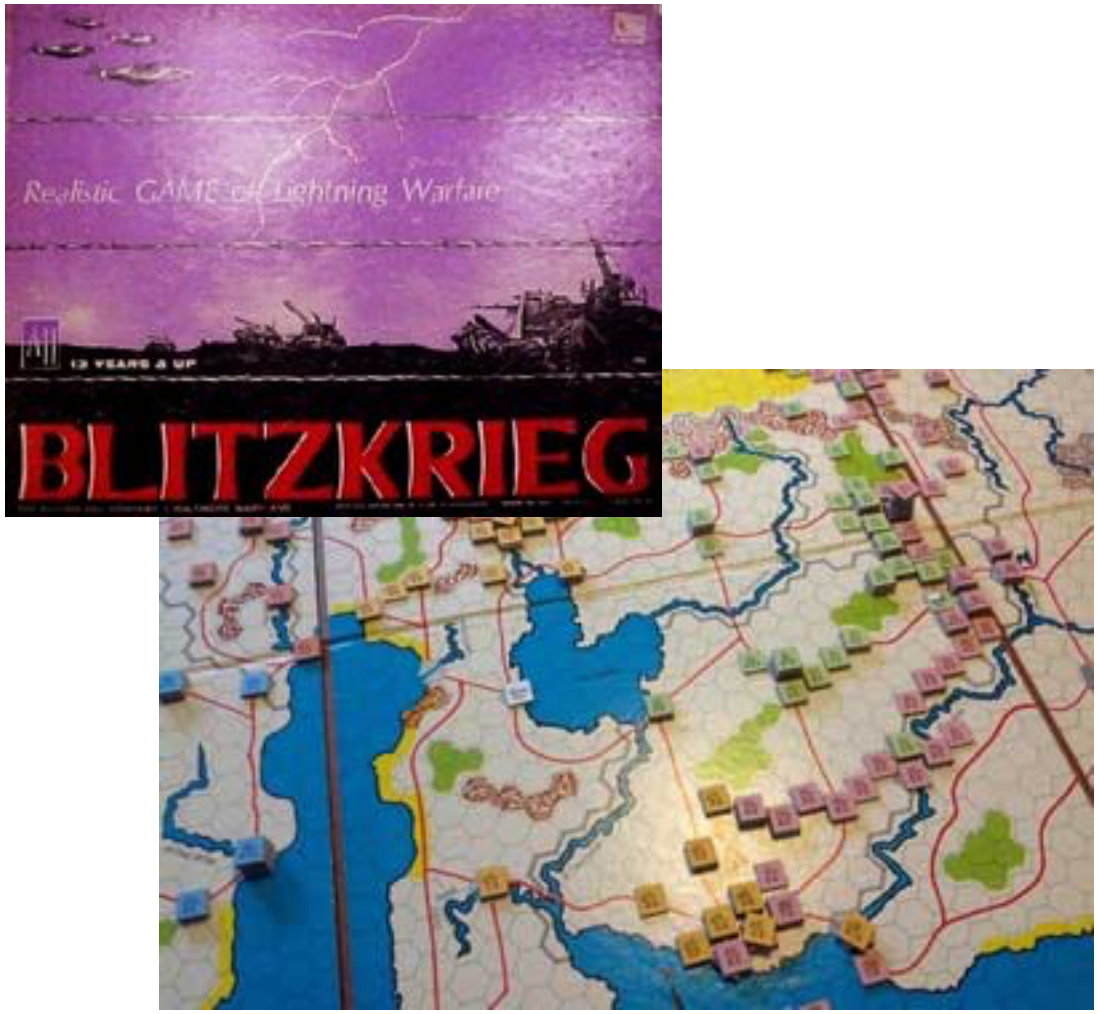
- Really do group rosters
- Get started on Unity
- Review project 1
  - Variation on Roll-A-Ball tutorial
- Today – take moments to work on Unity

# Today’s question

# What do you need to know to use Unity?

# Today: Unity Blitzkrieg

- Lighting war
  - Cover ground quickly
  - Go around enemy strongholds
  - Handle those later
- In Unity
  - Get an overview
  - Leave hard concepts for later
    - Geometry, navmesh, animations
- Work along
  - Experiment with Unity




Avalon Hill 1965

# Two steps

## 1. Build

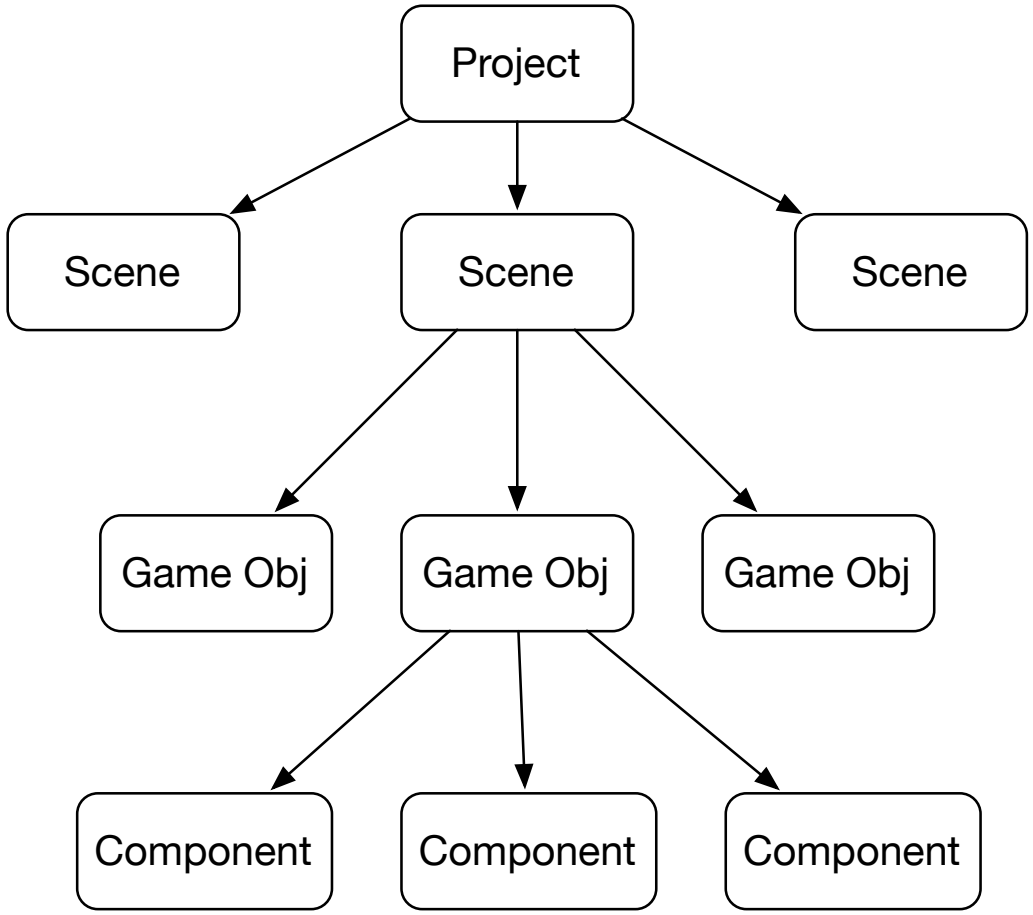
- Assemble resources
- Combine and layout in Unity GUI
- Create your world

## 2. Script

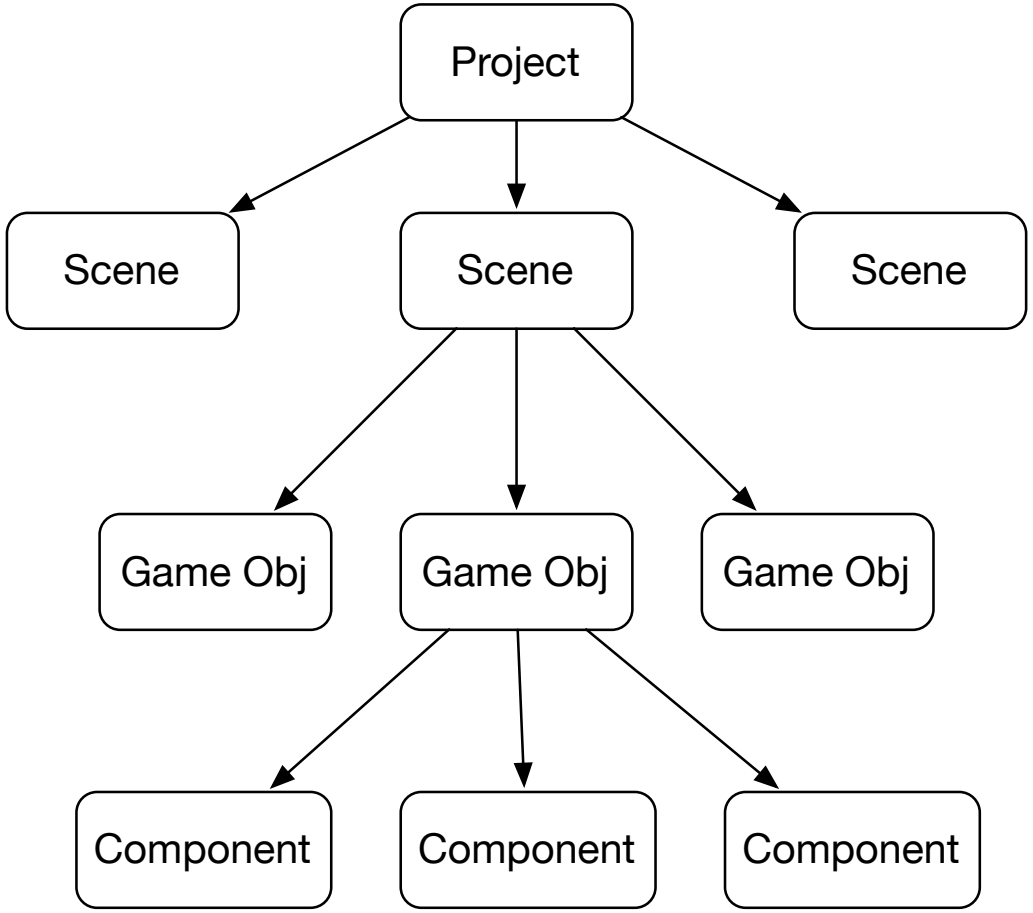
- Add behaviors
- Tie game objects together

- Project 1

# Unity game structure




# Unity game objects: elements of scene




Agents

Player

NPC

Traps

Environment

Terrain

Skybox

Obstacles

Collectables

Weapons

Potions

View

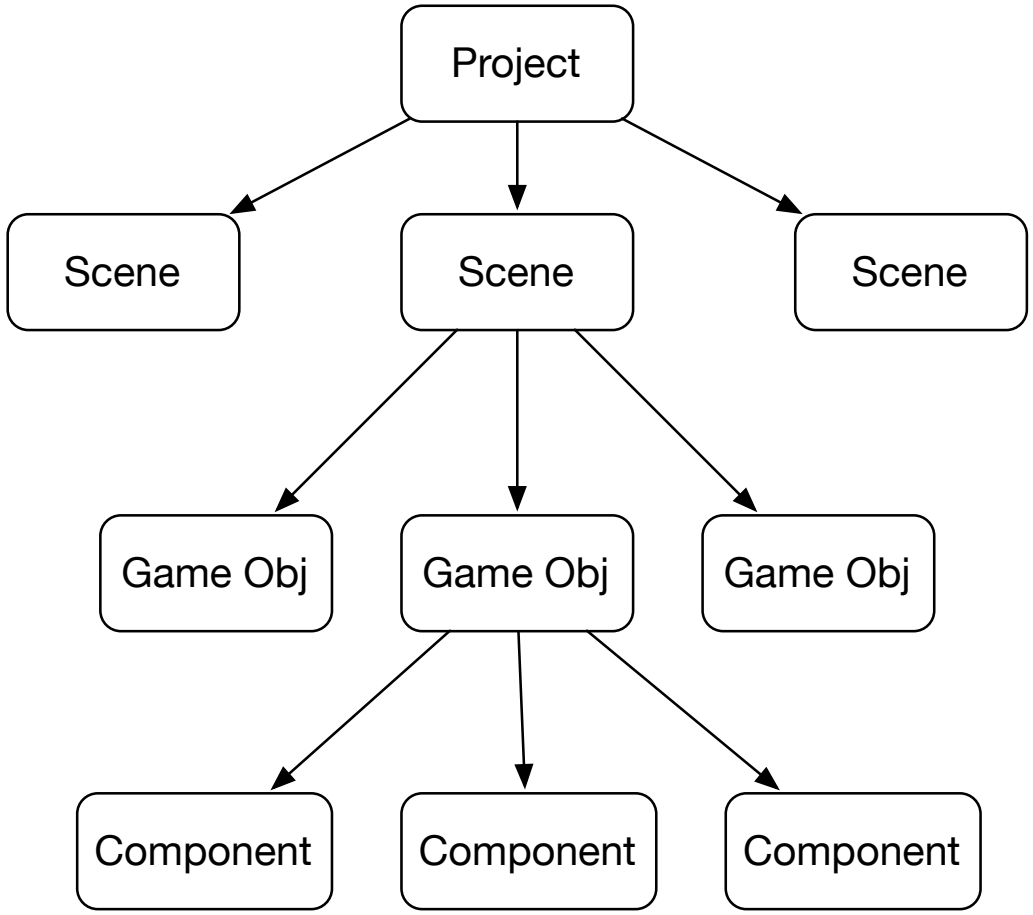
Cameras

Lights

Audio

SndPlayer

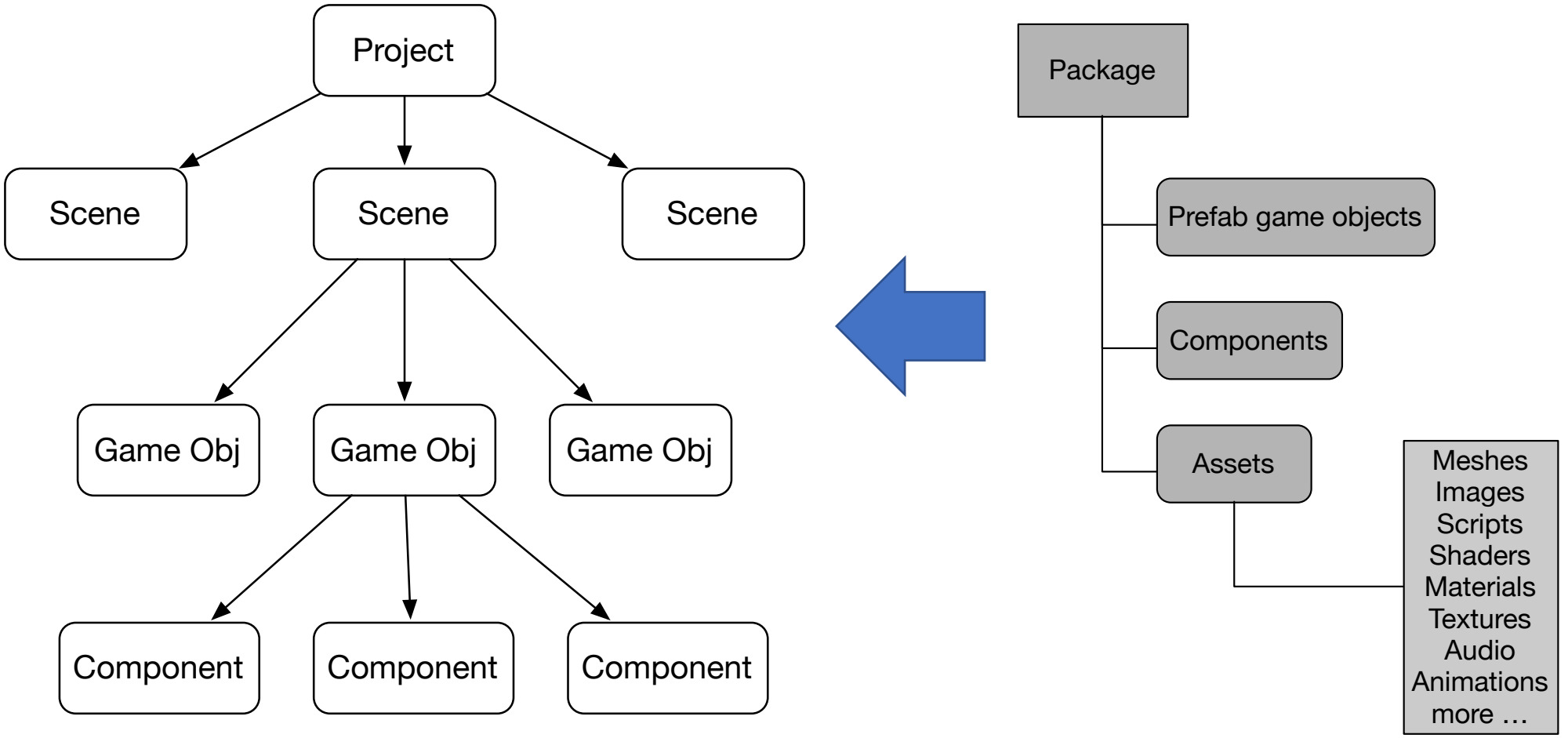
# Unity game components




Player object has

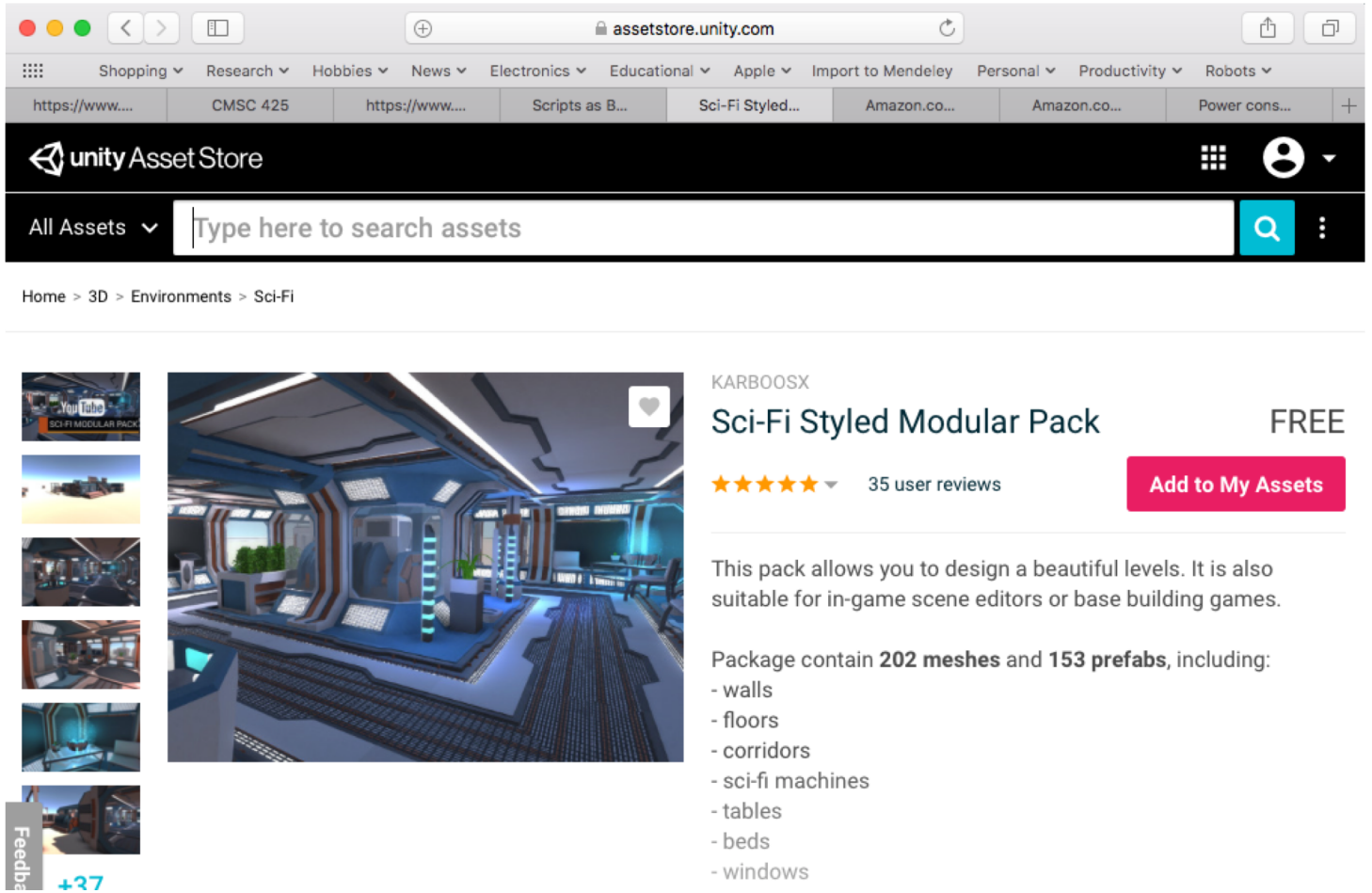
- Shape (mesh)
- Appearance (material, color, texture)
- Physics (rigid body)
- Extent (Collider)
- Behavior (scripts)

# Unity project: game + resources

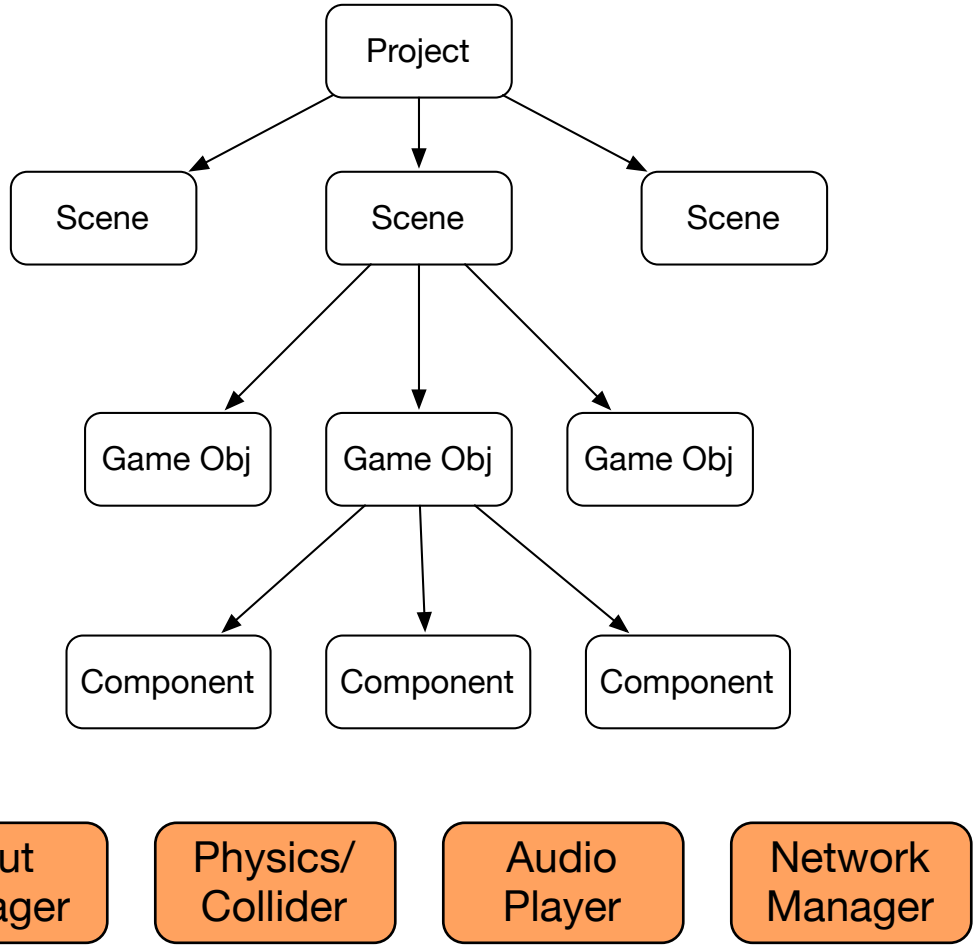


# Resources: Asset Store

- Free and paid
- Can use in projects
- (Animations in particular)
- But, cite your sources



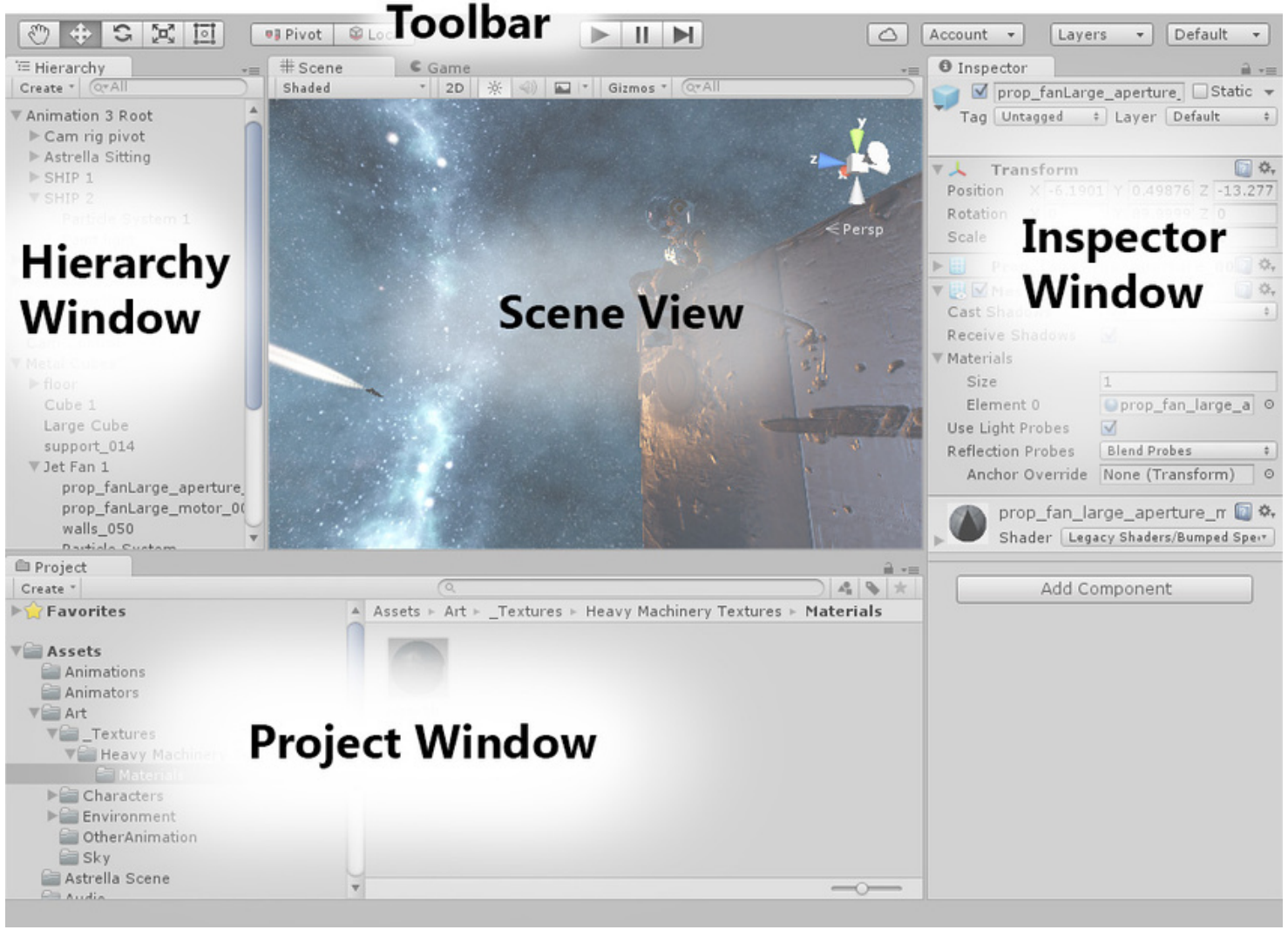
# Unity runtime: game + system elements




- Sources of events
  - Input Manager
  - Network Manager
  - Physics engine/Collider
- Services
  - Audio
  - Visual rendering
  - Access to assets
  - etc.

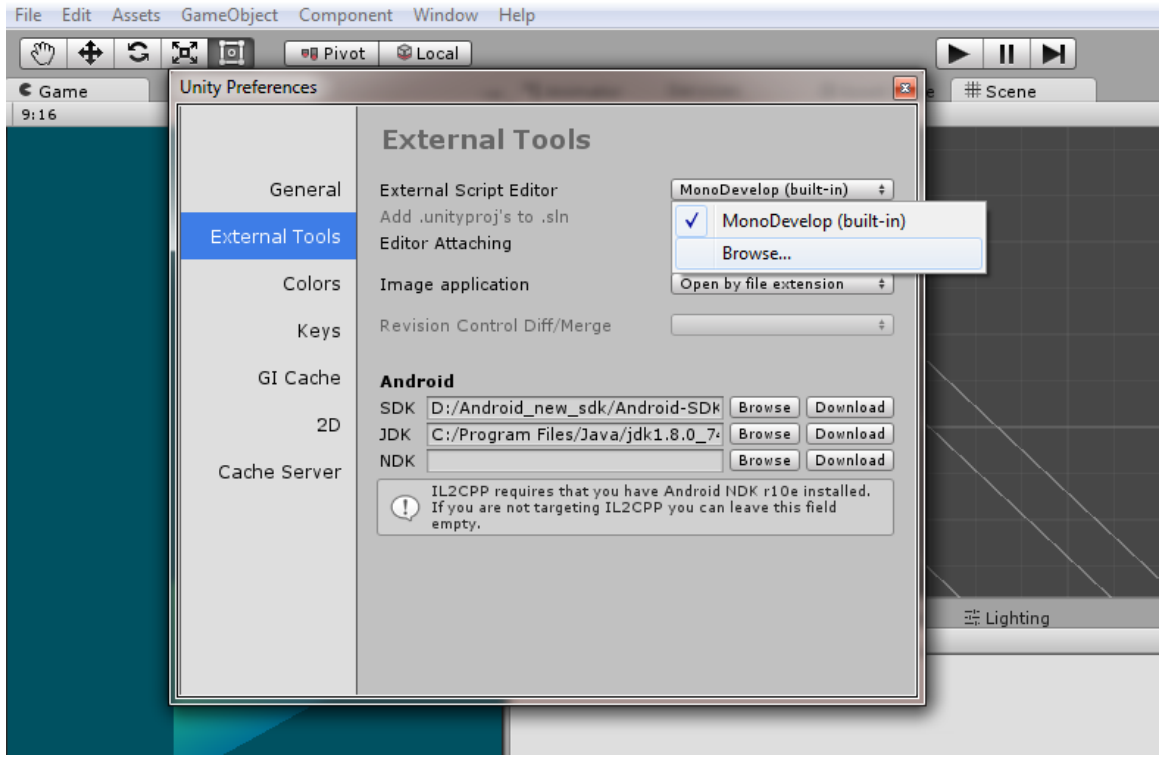
# Interface

- Scene/game view
  - Build scene
  - Play scene
- Hierarchy
  - Manage scene
- Inspector
  - Manage game objects and components
- Project window
  - Manage resources

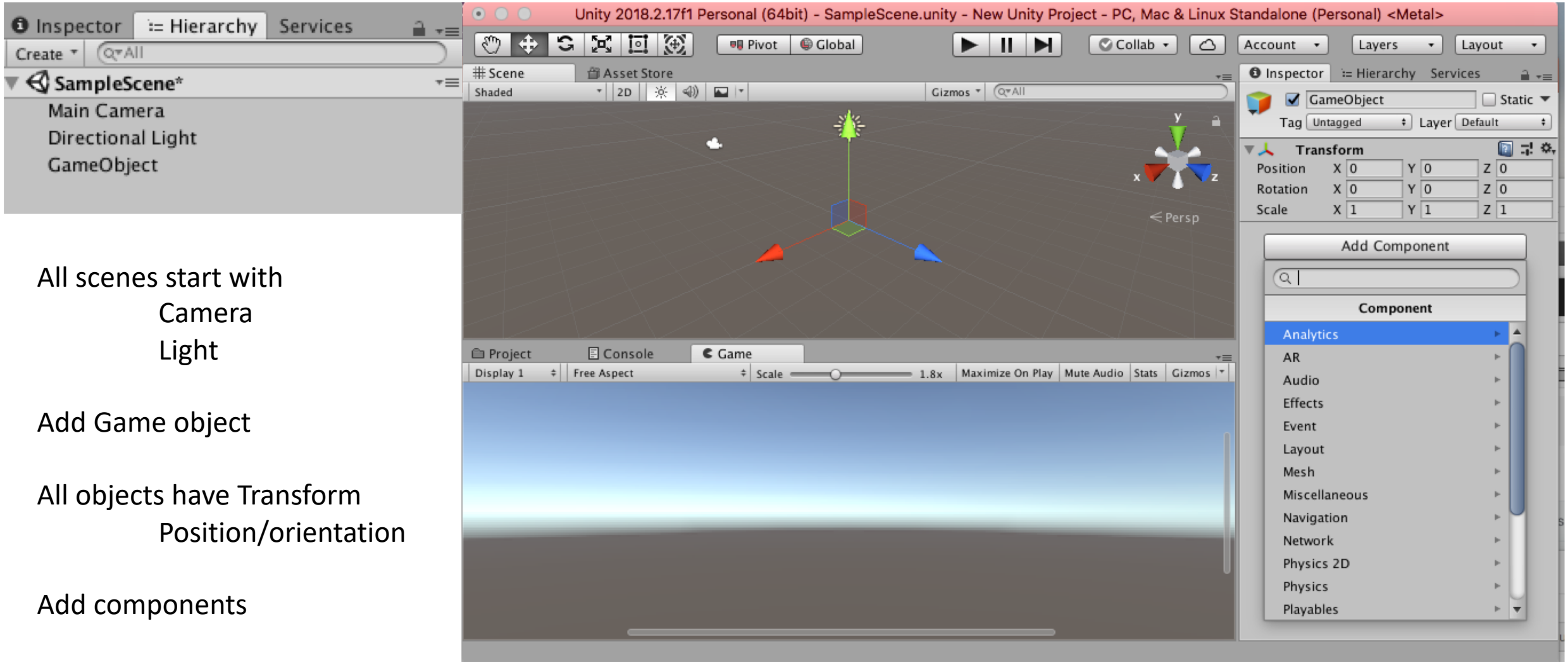


# Editing assets externally

- Use external editors to
  - Create/edit scripts
  - Create/edit images, meshes, shaders
  - Create character animations
- Unity does have internal editors
  - Terrain
  - Trees
- For C# scripts: Monodevelop or Visual Studio



# Blank game




All scenes start with
- Camera
- Light

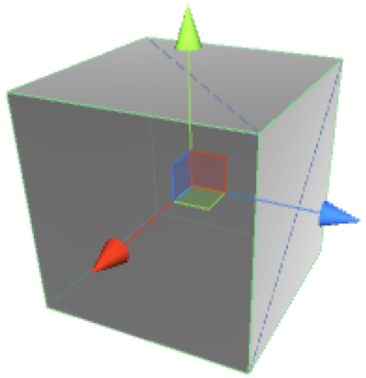
Add Game object

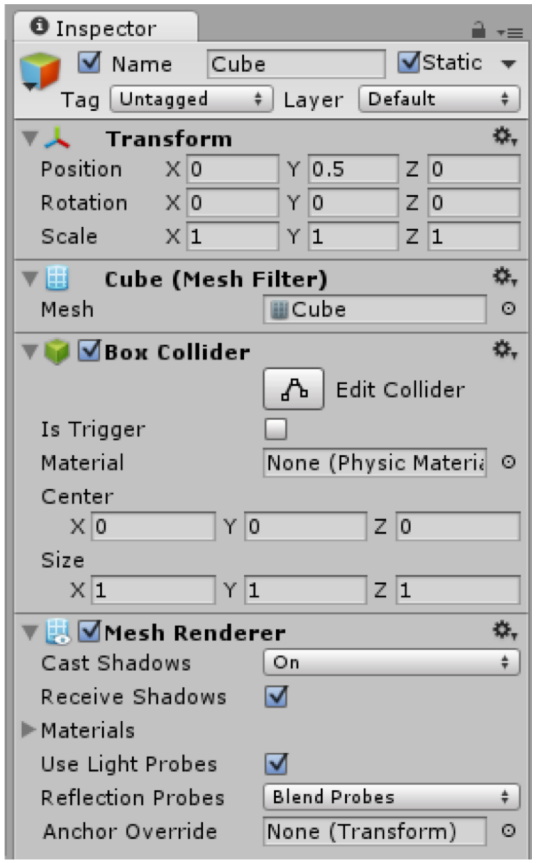
All objects have Transform
- Position/orientation

Add components

# Editing objects

- Add
  - Shape – Mesh filter
  - Collider
  - Renderer – color, reflection, etc
- Edit
  - Set position, orientation, scale
  - Set collider offset (if needed)
  - Set color, other properties

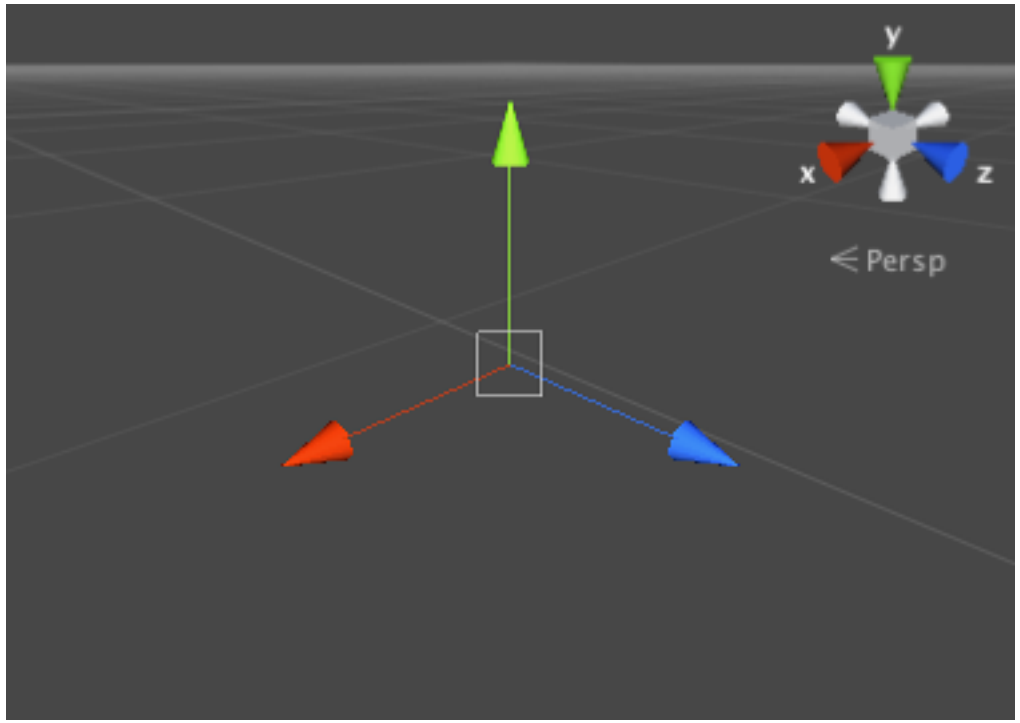




# Unity coordinate system: left handed

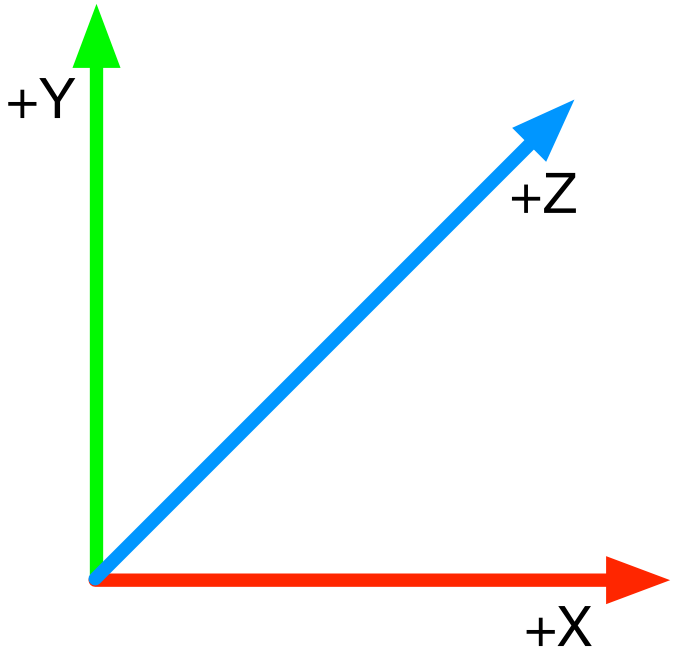
**World space:**

**left handed**




**Screen space:**

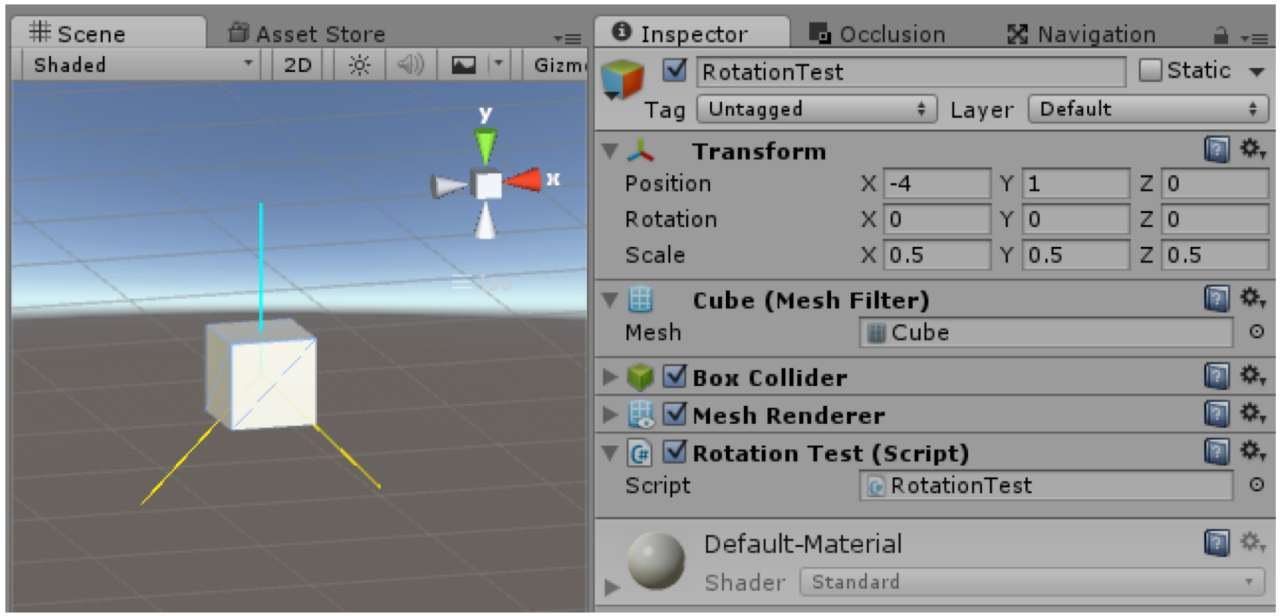
**Origin bottom left, positive z away**

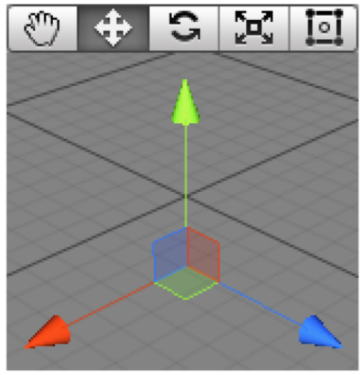



Many graphics systems right handed -> depth negative

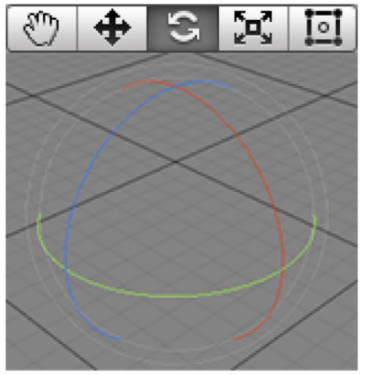
# Transforms

- Translate along axes
- Rotate around axes
- Scale along axes

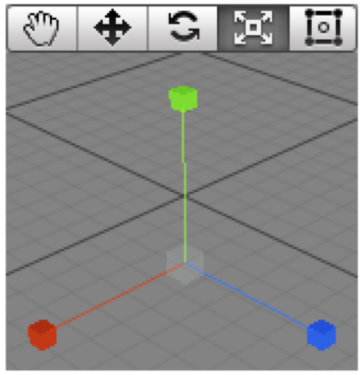


Translate (W)



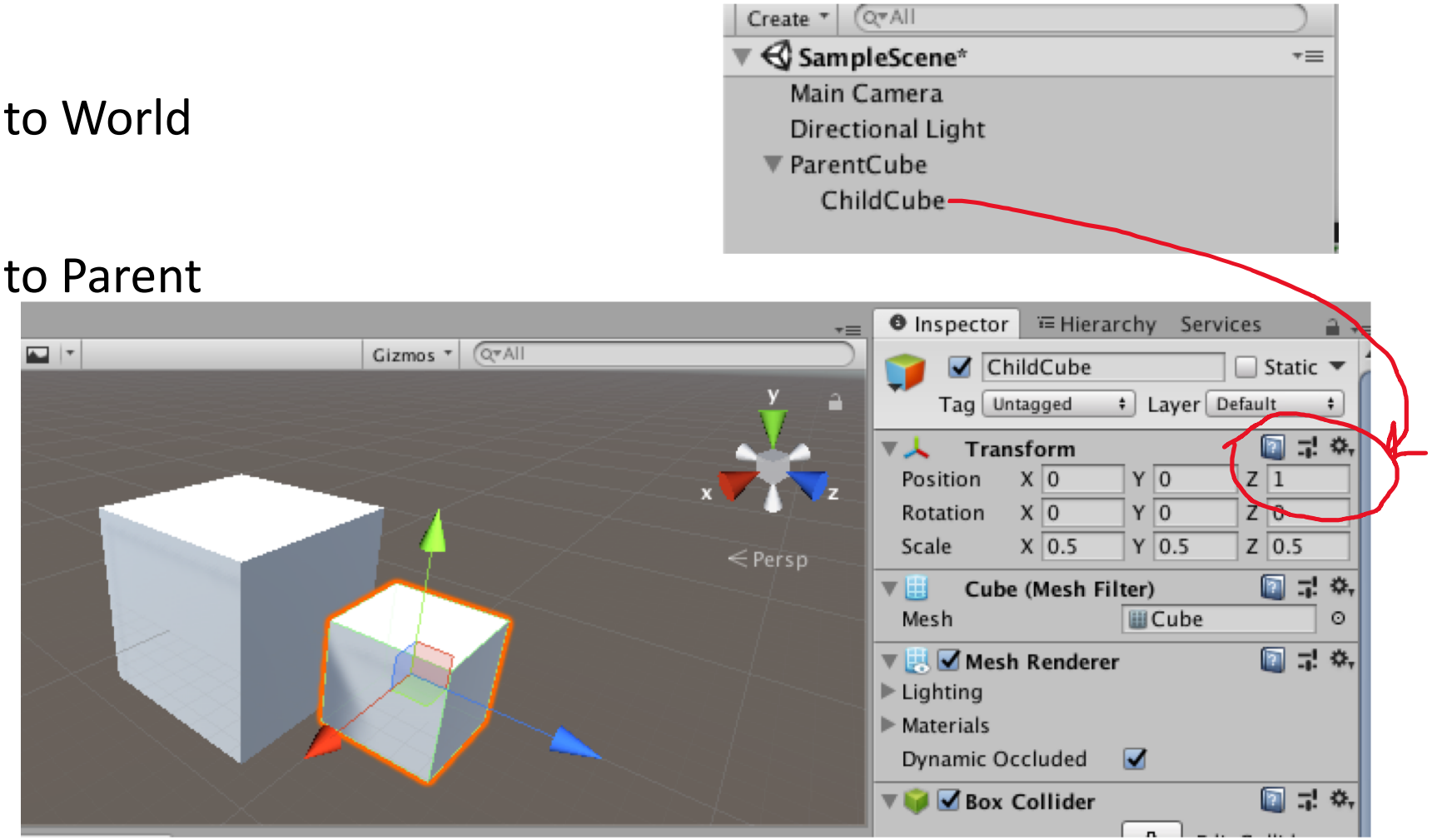
Rotate (E)



Scale (R)

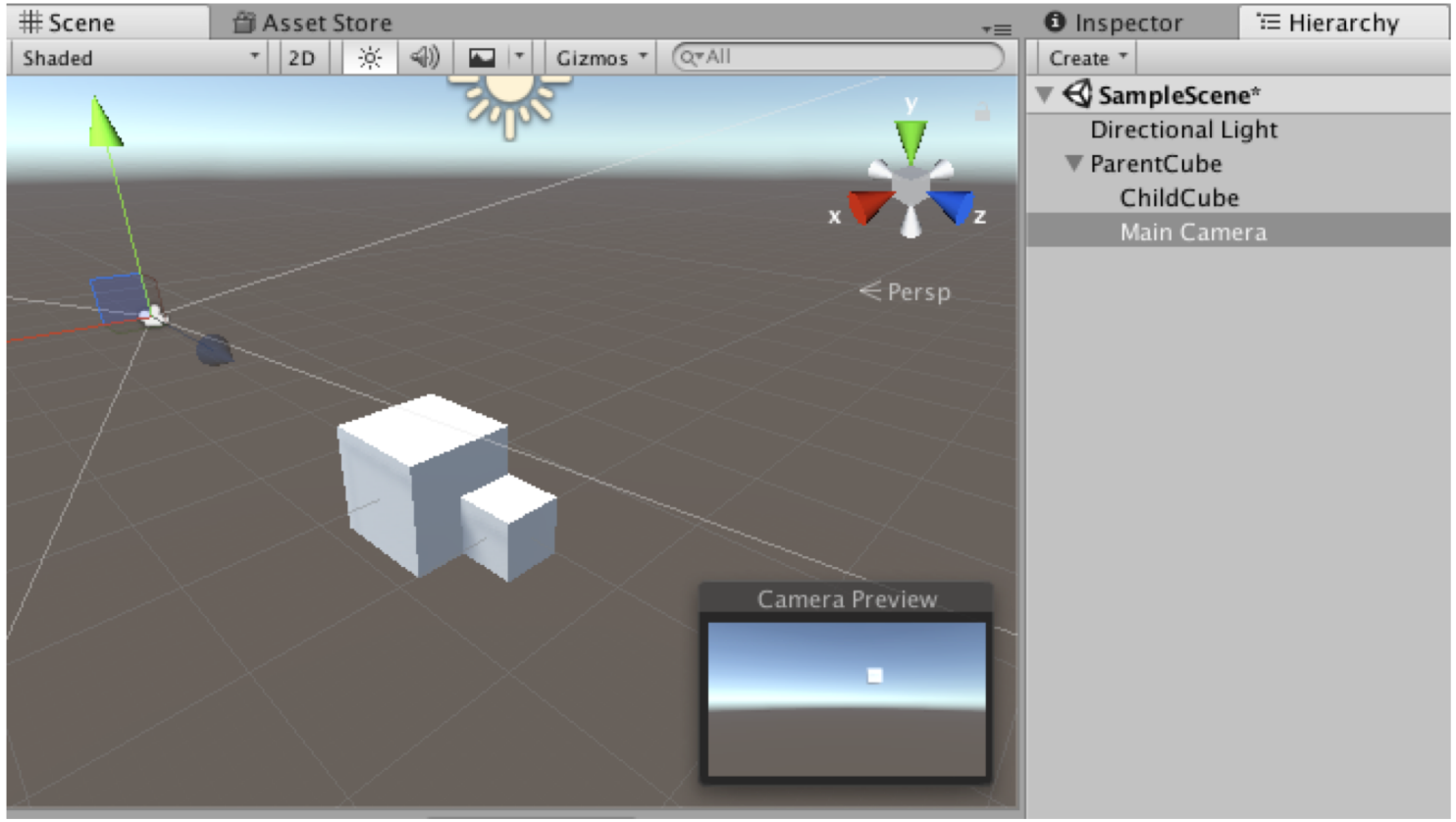
# Object hierarchy and transforms

- Root objects
  - Transform relative to World
- Child objects
  - Transform relative to Parent
- Move Parent
  - Move Child
- Scale Parent
  - Scale Child



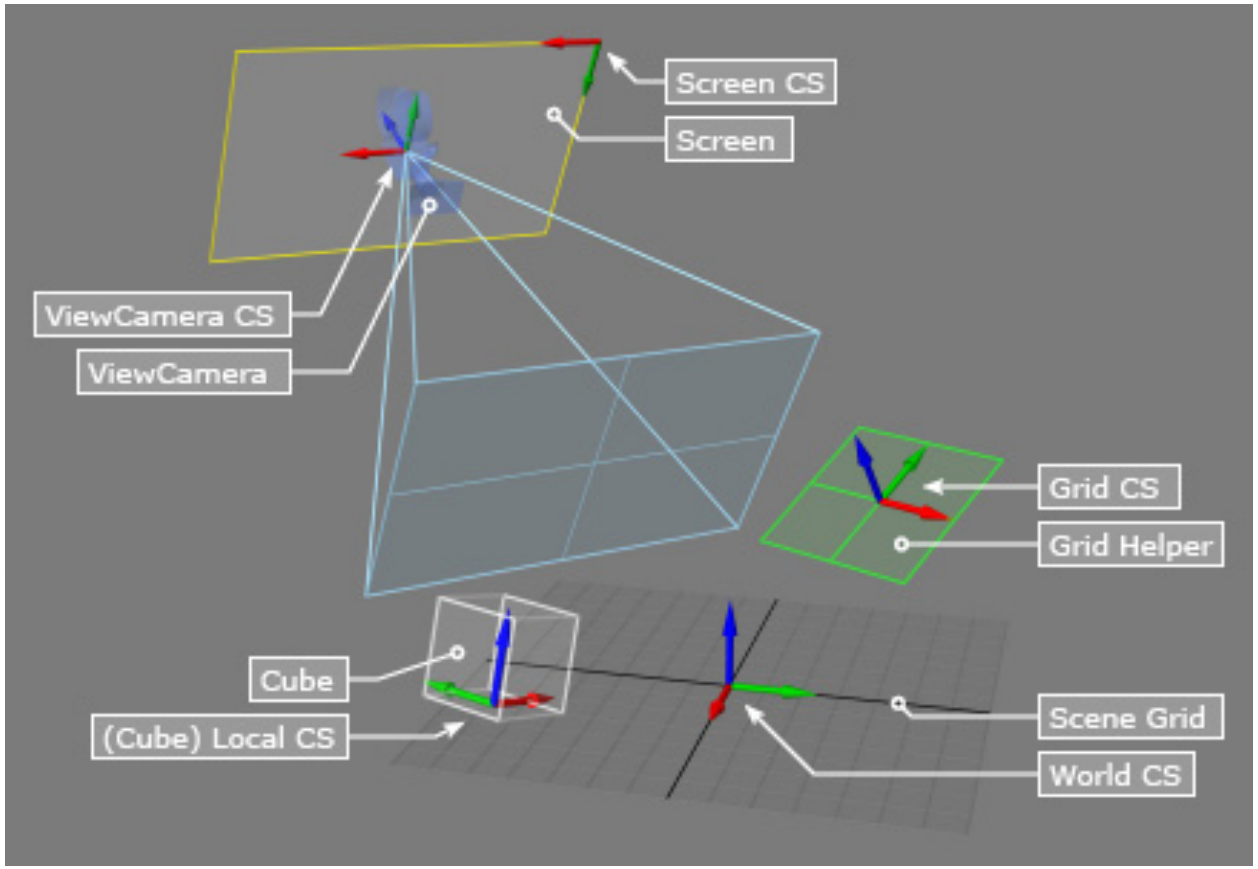
# Camera following object

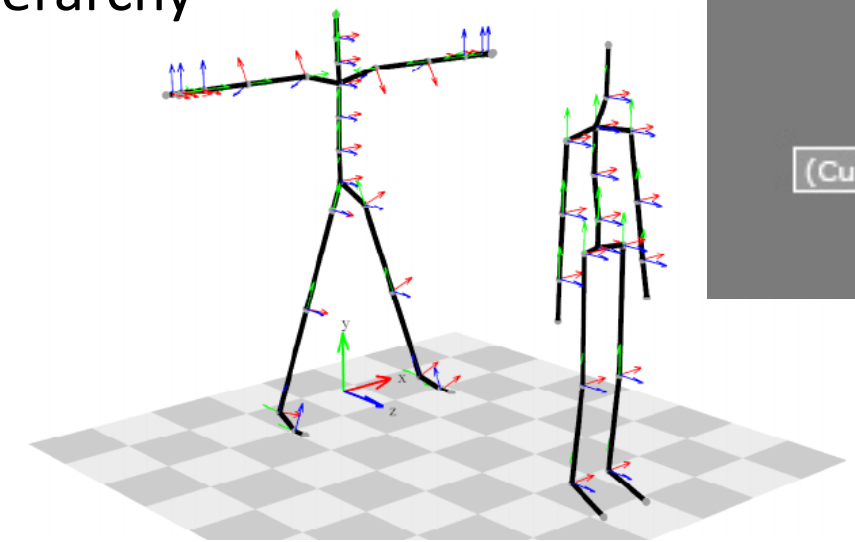
- Camera as child object
- Can also attach camera to follow in script (lookAt)



# Multiple coordinate systems

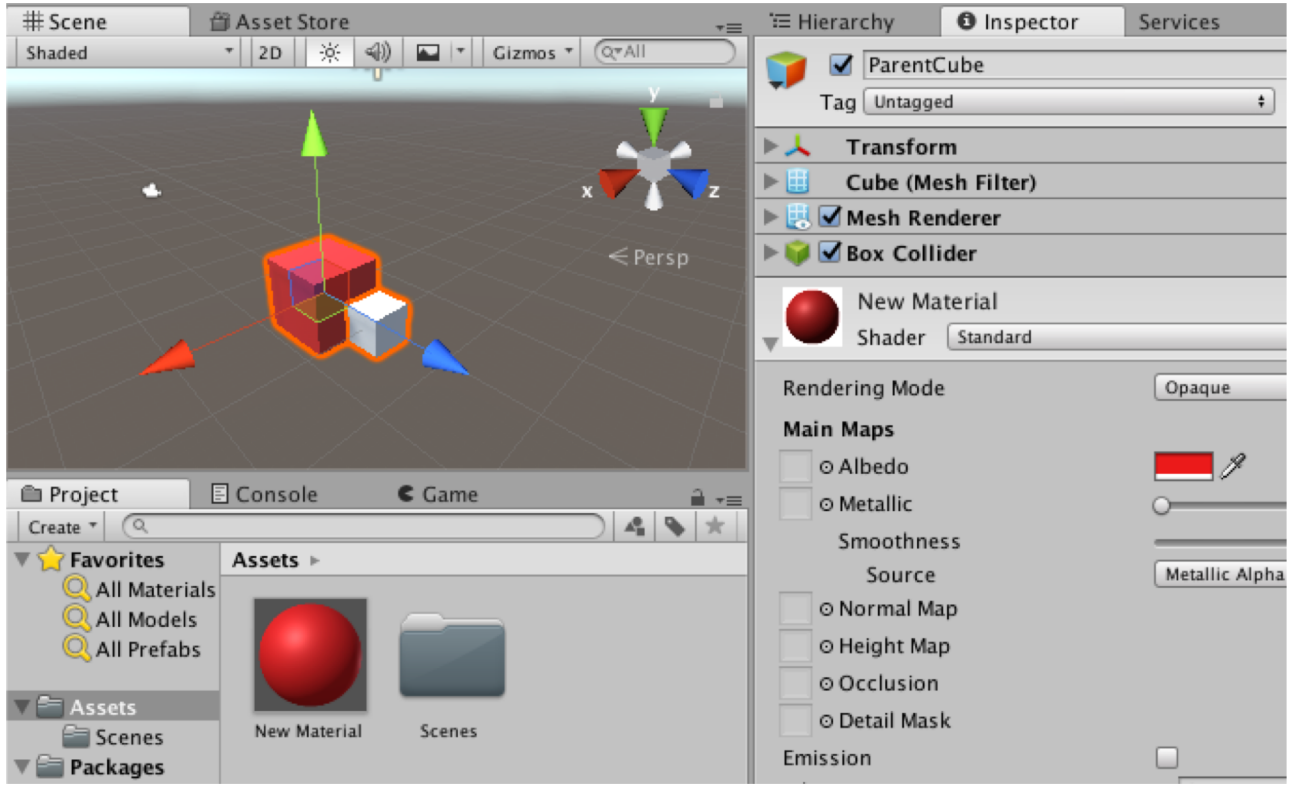
- World
- Scene
- Camera (3d), Screen (2d)
- Object
  - Object hierarchy
- Move left in which?



# Materials

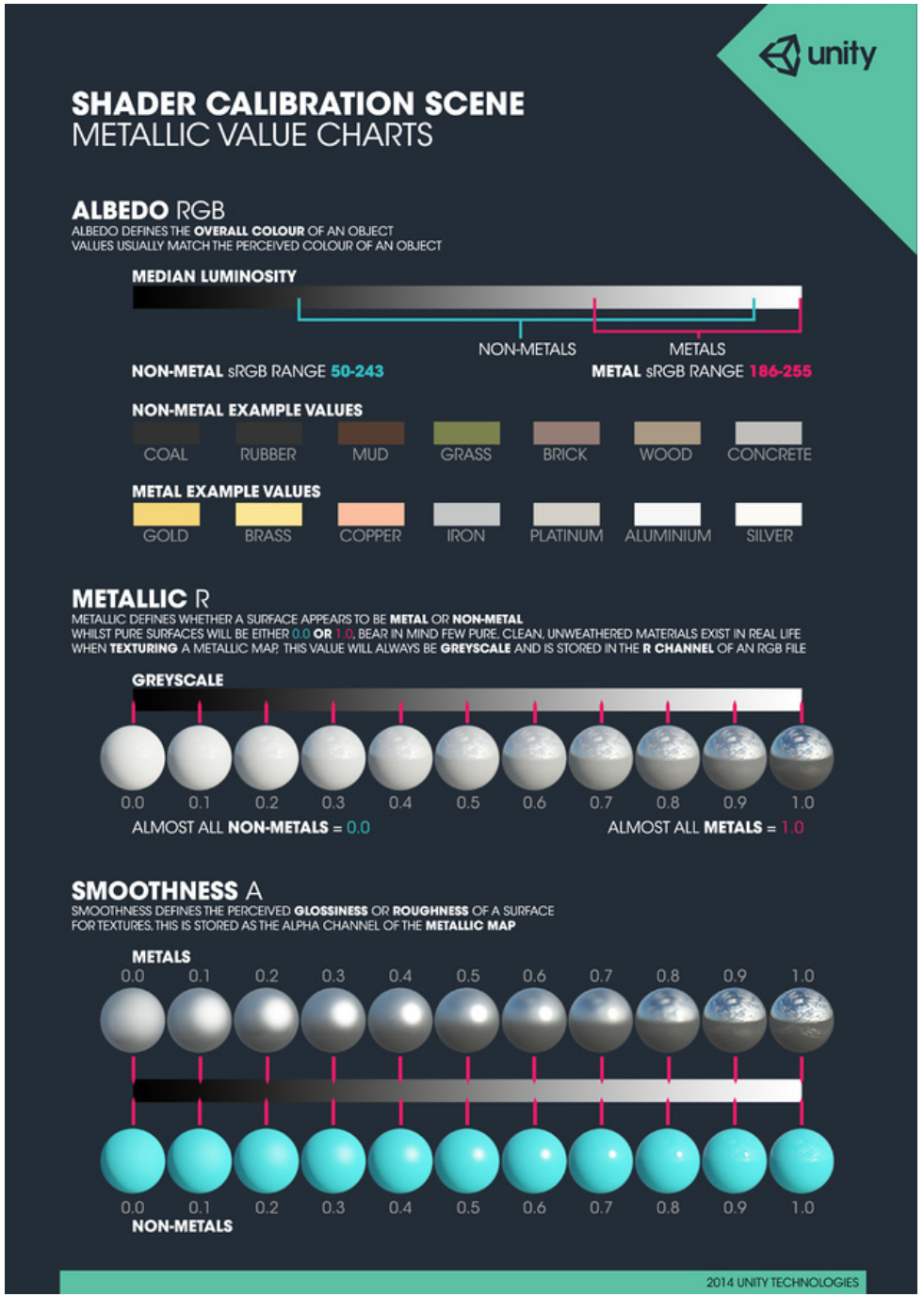
- Standard, default material
  - Tricky - starts greyed out
  - Can't edit directly
- Instead
  - Create new Material in Project
  - Drag onto Object to replace Standard material
  - Edit
- Can code with Shaders



# Material properties

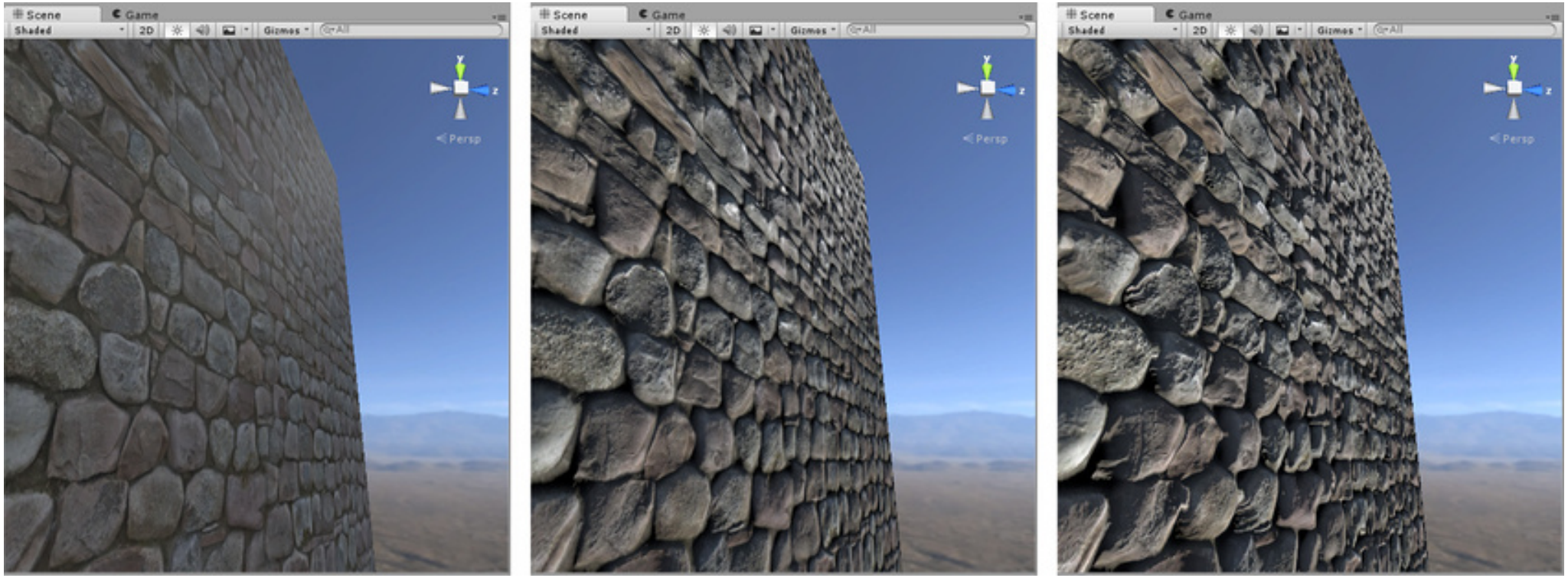
- Albedo (RGB color)
- Metallic (mirror-ness)
- Smoothness (shininess)

- Try yourself with sphere



# Texture mapping

- ***Albedo map*** – color
- ***Normal map*** –local orientation
- ***Height map*** – local displacement



# Unity stage one (build) summary

- Structure of game
  - Project-scene-object-component
  - Resources-packages-prefabs-assets
- Interface
  - Project(assets)-Hierarchy(objects)-Inspector(components)
  - Scene view(build)
  - Game view(play)
  - External editors
- Key components
  - Shape, transform, material

# Stage 2: With scripts you can

- Create and destroy objects
- Initialize objects
- Activate and inactivate objects
- Move objects
- Activate animations
- Change object appearance
- Keep score
- And more

Topics

1. Events
2. Life of an object
3. Event loop
4. Accessing data
5. Key Unity data types

# Scripting: UnityEvents

```
using UnityEngine; // basic objects
using System.Collections; // basic structures
public class MyGameObject : MonoBehaviour {
 void Start () {
   // ... initializations
   }

 void Update () {
   // ... code repeated each frame tick
   }
}
```

- C#
- Event driven
- No main
- Multiple scripts possible per object
- Base class for UnityEvents: MonoBehaviour

# Example: rotating cube

```
public class MyGameObject : MonoBehaviour {

 void Start () {
      transform.rotation = Quaternion.Euler(0,0,0);
      }

 void Update () {
      transform.Rotate (new Vector3 (0, 45, 0) * Time.deltaTime);
      }
}
// This shows: accessing component, use of delta time, Quaternions
```

# Comparing: event program in Processing

```
void setup() {
  size(400,400);
}
void draw() {
}
void mousePressed() {
  ellipse(mouseX,mouseY,20,20);
}
void keyPressed() {
  save("pic.jpg");
}
```

- setup – called once on program start
- draw – called every frame (rate adjustable)
- mousePressed – called once when mouse is pressed
- keyPressed – called once when key is pressed

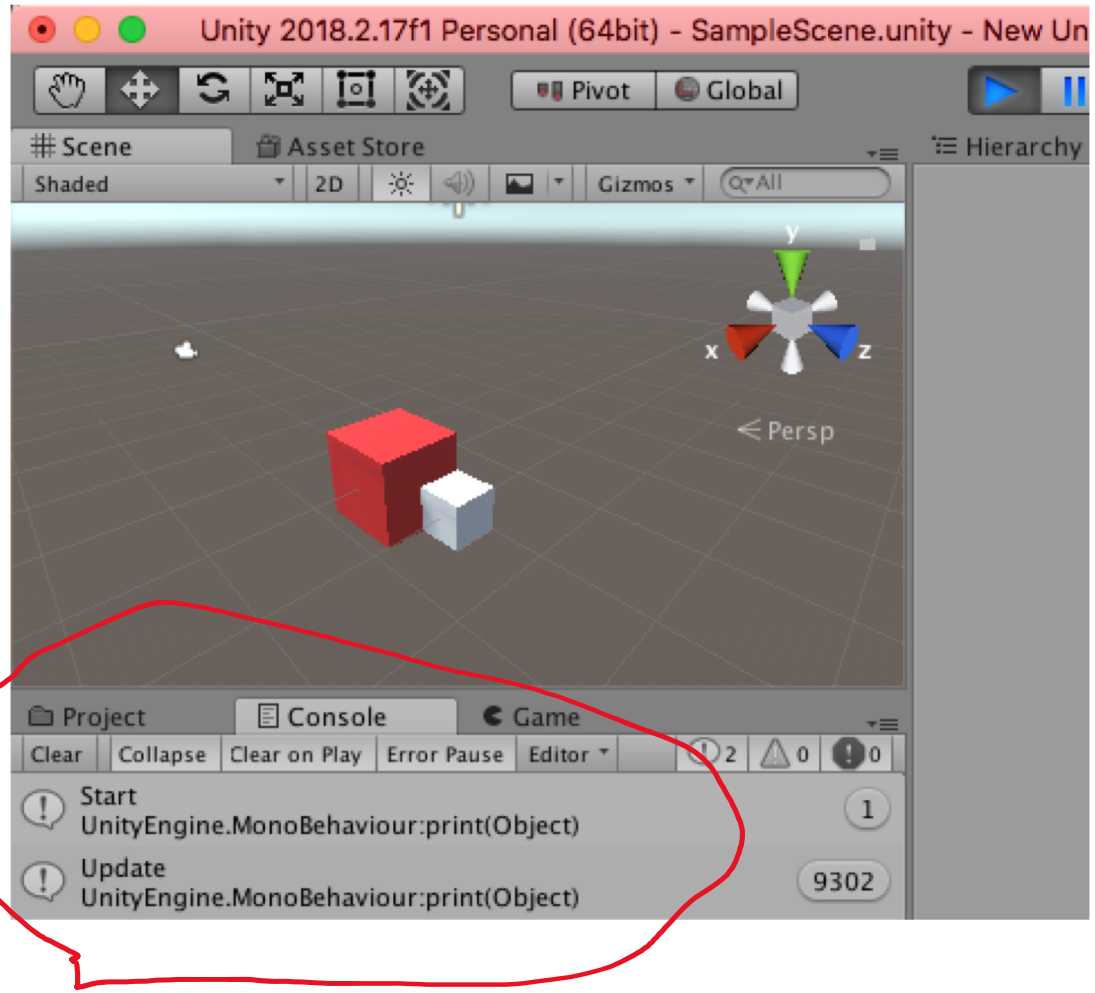
# Scripting: UnityEvents

```
using UnityEngine; // basic objects
using System.Collections; // basic structures

public class MyGameObject : MonoBehaviour {
 void Start () {     // ... LIKE SETUP
   }
 void Update () {    // ... LIKE DRAW(but, no draw cmds)
   }
 void OnMouseDown() {// ... LIKE MOUSEPRESSED
   }
}
```

# Tracking events through console log

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class NewBehaviourScript : MonoBehaviour {
6
7       // Use this for initialization
8       void Start () {
9           print("Start");
10
11      }
12
13      // Update is called once per frame
14      void Update () {
15          print("Update");
16
17      }
18  }
```



# Types of events

A. Object initialization and state
B. Object updates
C. Physics events including collisions and trigger
D. User input events

# 2. Lifetime of objects

- Some objects persist throughout the game – player, etc
- Some objects only need be enabled when their room is entered
  - Avoid spending time calling their update, etc, when not used or viewed
- Some objects only needed to be rendered when viewable
  - Don't try to render things behind you, or too far away
- Some objects have short lifetime – create and destroy quickly
  - Projectiles, spell animations, and so on

# A. Object initialization and state events

- void Awake - when object is initialized (set up all objects)
- **void Start** - when object is enabled (eg, when room is entered) *(enough for now)*
- object.enable - turns off update, rendering, but not all physics
- object.active - turns off all components/events
- Tricky! Can't re-enable in Update if that's turned off

# Create new object from prefab

- Load Missile prefab

```
public class RocketShipController : MonoBehaviour {
    public GameObject mPrefab;
    void Start () {
       GameObject mPrefab = Resources.Load("Missile") as GameObject;
       }
```

- Instantiate

```
void ShootMissile () {
      GameObject m = Instantiate(mPrefab , transform.position ,
                                          transform.rotation);
     m.velocity = transform.TransformDirection(Vector3.forward*10);
      }
```

# B. Object updates

- void Update
  - called at frame rate
  - intervals not constant
- void FixedUpdate
  - called at fixed interval
  - Time.fixedDeltaTime
  - for accurate physics
- void LateUpdate
  - called after Update calls are done
  - for objects that react to all others

# C. Physics: collisions and triggers

- Events when objects overlap

- For colliders: — When two objects collide!
  - void OnCollisionEnter()
  - void OnCollisionStay()
  - void OnCollisionExit()
- For triggers: — Put invisible objects in doors, pads
  - void OnTriggerEnter()
  - void OnTriggerStay()
  - void OnTriggerExit()

# D. User input events

- Event handlers

```
void OnMouseDown()
void OnMouseUp()
void OnMouseOver()


void OnMouseDrag(){
   print("dragging");
}
```

- Polling

```
public void Update() {
 if(Input.GetButtonDown("Fire1")) {
   Debug.Log(Input.mousePosition);
  }
 }
```

- Choice: efficiency, code complexity

# 3. Unity game loop

Initialize game

do

  Physics (+collision)

  Input

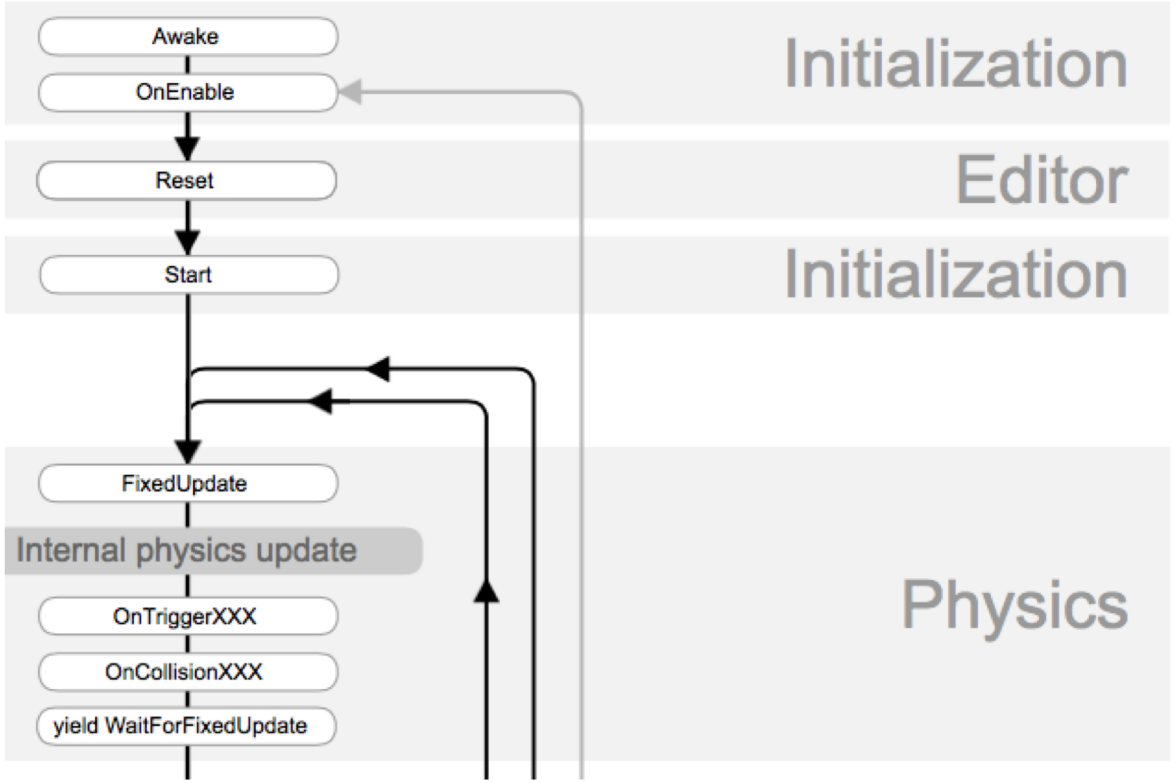
  *Game logic(new)*

  Rendering

  GUI rendering

loop

Clean up




- Events handled in order during loop
- https://docs.unity3d.com/Manual/ExecutionOrder.html

# Review: Time!

- Frame time (not constant)
  - Things executed every frame
  - Most important is rendering of scene
- Physics time
  - Steps in physics simulation
  - May run faster than frame time to get physics right (avoid big steps)
- Real time
  - System clock
  - For syncing music, video, other things that need real time

# Event loops and time: 45 degrees/second

- Frame time (not constant)

```
void Update () {
      transform.Rotate (new Vector3 (0, 45, 0) * Time.deltaTime);
}
```

- Physics time

```
void FixedUpdate () {
      transform.Rotate (new Vector3 (0, 45, 0) * Time.fixedDeltaTime);
}
// 0.02 typically
```

# Going slower than frame rate?

- Coroutines
- Yield control each loop with "yield" command
- Call in Update, resumed with each new Update

```
IEnumerator Fade() { // gradually fade from opaque to transparent
   for (float f = 1f; f >= 0; f -= 0.1f) {
       Color c = renderer.material.color;
       c.a = f;
       renderer.material.color = c;
       yield return null; // return to Unity to redraw the scene
       }
   }
```

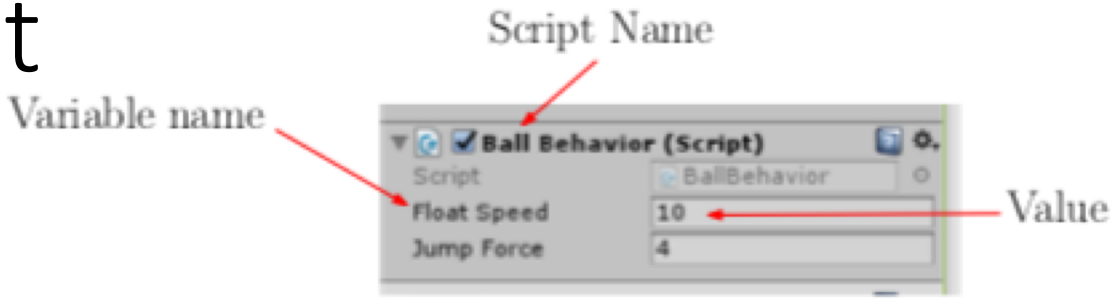
# 4. Key Unity components and data types

- Transform - Position and orientation
  - **Vector3:** `Vector3 u = new Vector3(1, 2, -3);`
  - **Ray:** `Ray ray = new Ray(FromVector, ToVector);`
  - **Quaternion:** `Quaternion q1 = Quaternion.Euler(0,30,0);`
- Rigid body - Physical properties
  - `rb.mass = 10f; // change this body's mass`
  - `rb.AddForce(Vector3.up * 10f); //up force`
- Collider - Extent of game object
- Material - Color and surface properties

# Motion options

- Rigid body — Has mass, extent, other properties<br>Nudge by forces
- Kinematic object — Set position and velocity directly
- Static object — Doesn’t move<br>So don't do static/static collision detection

# 5. Accessing data in a script




- Access public script variables

```
public float floatSpeed = 10.0f; // how fast ball floats up
public float jumpForce = 4.0f; // force applied when ball jumps
```

- Access object components of your object

```
Rigidbody rb = GetComponent <Rigidbody >();
```

- Access other game objects by game or tag

```
GameObject camera = GameObject.Find ("Main Camera");
GameObject player = GameObject.FindWithTag("Player");
GameObject[] enemies =
     GameObject.FindGameObjectsWithTag("Enemy");
```

# C# vs Java

- Similar
  - OOP, garbage collection, bytecode, data types, control structures
- Differences that matter in Unity
  - yield statement allows coroutines in Unity
  - inheritance system different- can packages up objects more completely
- Will leave it to you to learn the details of C#

# Summary

- After today you should be able:

Have a better handle on Unity tutorials

1) Explain the hierarchical structure of a Unity game
2) List the usual components of a game object
3) Use the Unity interface to create and edit Unity projects and elements
4) Explain and use the Unity left handed coordinate system
5) Use transform component to move and orient an object
6) Explain how the parent child relationship effects object position
7) Explain some basic properties of the material component
8) Start on writing Unity C# scripts with an understanding of events, object life, event loop, accessing data inside and outside objects, key data types

# Readings

- David Mount's lecture
- "**Intro to Unity**"
- Roll-A-Ball tutorial
- Project 1 assignment
- Unity Manual (browse as you need)
- Find other tutorials, use the manual as you wish
- https://www.raywenderlich.com/980-introduction-to-unity-scripting

# Activity 4a: Design a computer game

- At each table plan out a game for your team. Answer these questions (quickly!)

- What type of game? (platformer, FPS, RPG, etc. Multi-player?)
- What design choices?
  - Story
  - Environment
  - Characters
  - Gameplay
  - Visual look and feel

# Activity 4b: Build a computer game

- At each table plan out a game for your team. Answer these questions (quickly!)

- What platform(s)?
- Any special hardware or peripherals needed?
- What software elements needed?
- Build from scratch or use engine? Which language or engine?
- What assets will you need? How will you make or get them?