CMPS 327: Introduction to Video Game Design and Development

Dr. Arun K. Kulshreshth

Fall 2020

Lecture 5: Unity Basics I

Announcement

- Project 2 is available on Moodle
  - Due date: Monday, September 14, 2020 11:00 PM
  - Based on today’s lecture

About Today’s Lecture

- Game Objects
- Game Camera
- Transforms
- Game physics
- C# Scripting

Game Objects

- Fundamental Unit of any scene
- Multiple types of game objects
  - Models, lights, camera, particle systems etc.

- Built-in game objects
  - Primitive: cube, sphere, etc.
  - Camera, lights, particle systems, etc.

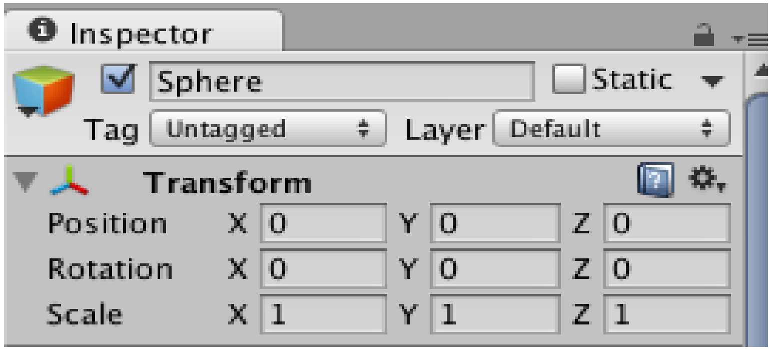
- Custom game objects
  - Can use primitive game objects to create complex objects

Game Objects

- Every game object has several components
- Components add features to the game object
  - Control the behavior of the game object
  - Script is also a component
- Game object has a name
  - You can assign a tag and a layer as well
- Transform is the defaults component

Transform Component

- Three parts
  - Position, Rotation and Scale
- Position along X, Y, Z axis
- Rotation along X, Y, Z axis
  - Euler angles
- Scale along X, Y, Z axis



☑ Sphere □ Static

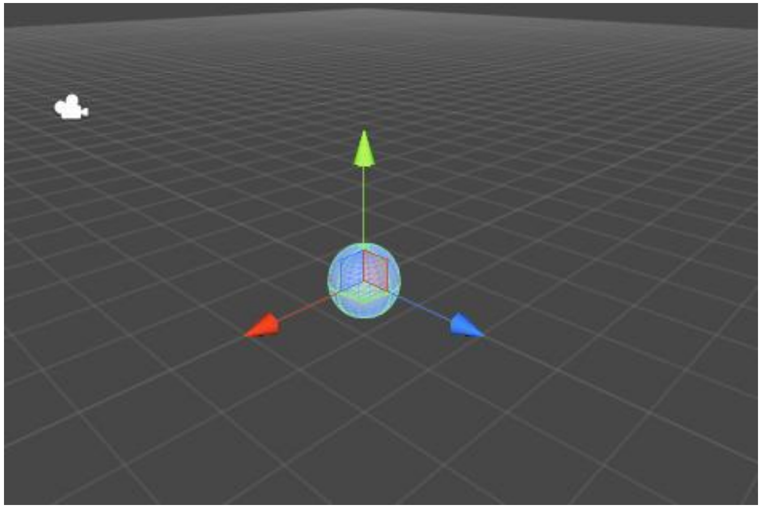
Tag Untagged Layer Default

Transform

Position	X	0	Y	0	Z	0
Rotation	X	0	Y	0	Z	0
Scale	X	1	Y	1	Z	1

Game Object Creation

- GameObject->3D object->sphere
- Edit->Frame Selected (to show the created object)
    - Press F as shortcut when scene tab is active
- Hold onto the arrows to move the sphere or change the position in the Inspector.
- Note: Y is up.



Parenting Game Objects

- Game objects can have parent child relationship
- While changing transform using script you can
    - Change the world position/rotation/scale
    - Change the local position/rotation/scale with respect to the parent (parent is origin)
- When no parent then the changes are in world coordinates

Position change using a script

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

public class testScript : MonoBehaviour {

    // Update is called once per frame
    void Update () {
                transform.position = new Vector3 (10.0f, 10.0f, 10.0f);
    }
}
```

- transform.position.x = newx
- transform.position.y = newy
- transform.position.z = newz
- transform.position += Vector3.up*5.0f;
- Use transform.localPosition for changes relative to its parent

Rotation change using a script

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

public class testScript : MonoBehaviour {

    // Update is called once per frame
    void Update () {
            transform.eulerAngles = new Vector3(0, 90.0f, 0);
    }
}
```

- transform.rotation = new Quaternion(0, 90.0f, 0, 0);
- Use transform.localRotation for changes relative to its parent
- transform.localEulerAngles for Euler angle-based change relative to the parent

Scale change using a script

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

public class testScript : MonoBehaviour {

   // Update is called once per frame
   void Update () {
               transform.localScale = new Vector3 (10.0f, 10.0f, 10.0f);
   }
}
```

- transform.localScale.x = newx
- transform.localScale.y = newy
- transform. localScale.z = newz
- transform. localScale += new Vector3(1.0f, 0.0f, 1.5f);

Smooth Rotation

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

public class testScript : MonoBehaviour {

   // Update is called once per frame
   void Update () {
            transform.localEulerAngles += new Vector3(0, 1.0f, 0);
   }
}
```

This example rotates the game object along y-axis at 1 degree per frame

Smooth Motion Between Two Positions

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    public Transform startMarker;
    public Transform endMarker;
    public float speed = 1.0F;
    private float startTime;
    private float journeyLength;
    void Start() {
        startTime = Time.time;
        journeyLength = Vector3.Distance(startMarker.position, endMarker.position);
    }
    void Update() {
        float distCovered = (Time.time - startTime) * speed;
        float fracJourney = distCovered / journeyLength;
        transform.position = Vector3.Lerp(startMarker.position, endMarker.position, fracJourney);
    }
}
```

https://docs.unity3d.com/ScriptReference/Vector3.Lerp.html

Vector3.Lerp

- Linearly interpolates between two vectors.

- Interpolates between the vectors a and b by the interpolant t.
- The parameter t is clamped to the range [0, 1].

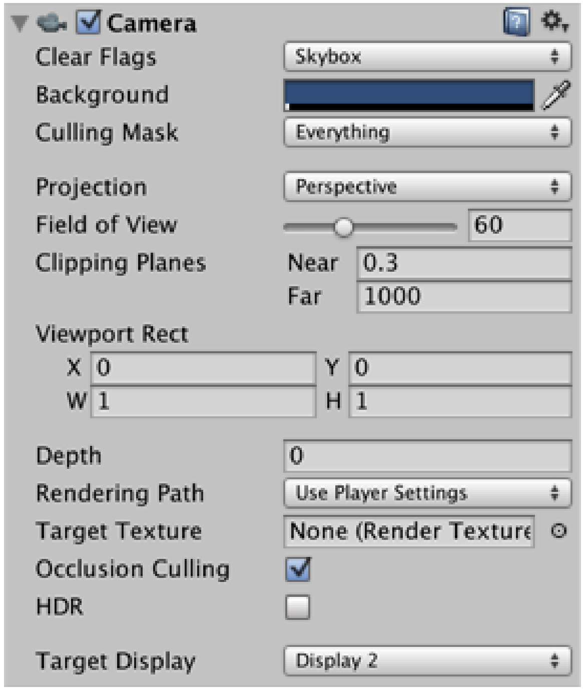
- This is most commonly used to find a point some fraction of the way along a line between two endpoints (e.g. to move an object gradually between those points).

- When t = 0 returns a. When t = 1 returns b. When t = 0.5 returns the point midway between a and b.

Unity 3D Camera

https://docs.unity3d.com/Manual/class-Camera.html

- Cameras are the devices that capture and display the world to the player.
- You can have an unlimited number of cameras in a scene. They can be set to render in any order, at any place on the screen, or only certain parts of the screen.
- Cameras are also responsible for listening for audio.



☑ Camera

Clear Flags	Skybox
Background	🟦
Culling Mask	Everything
Projection	Perspective
Field of View	60
Clipping Planes	Near 0.3 Far 1000
Viewport Rect	X 0 Y 0 W 1 H 1
Depth	0
Rendering Path	Use Player Settings
Target Texture	None (Render Texture
Occlusion Culling	☑
HDR	□
Target Display	Display 2

Unity 3D Camera

- Customizable
  - Puzzle game: keep the Camera static for a full view of the puzzle
  - First-person shooter: parent the Camera to the player character, and place it at the character's eye level
  - Racing game : have the Camera follow your player's vehicle

Camera Properties

- Clear Flags
  - Determines which parts of the screen will be cleared. This is handy when using multiple Cameras to draw different game elements.
- Background
  - The color applied to the remaining screen after all elements in view have been drawn and there is no skybox.
- Culling Mask
  - Includes or omits layers of objects to be rendered by the Camera. Assigns layers to your objects in the Inspector.

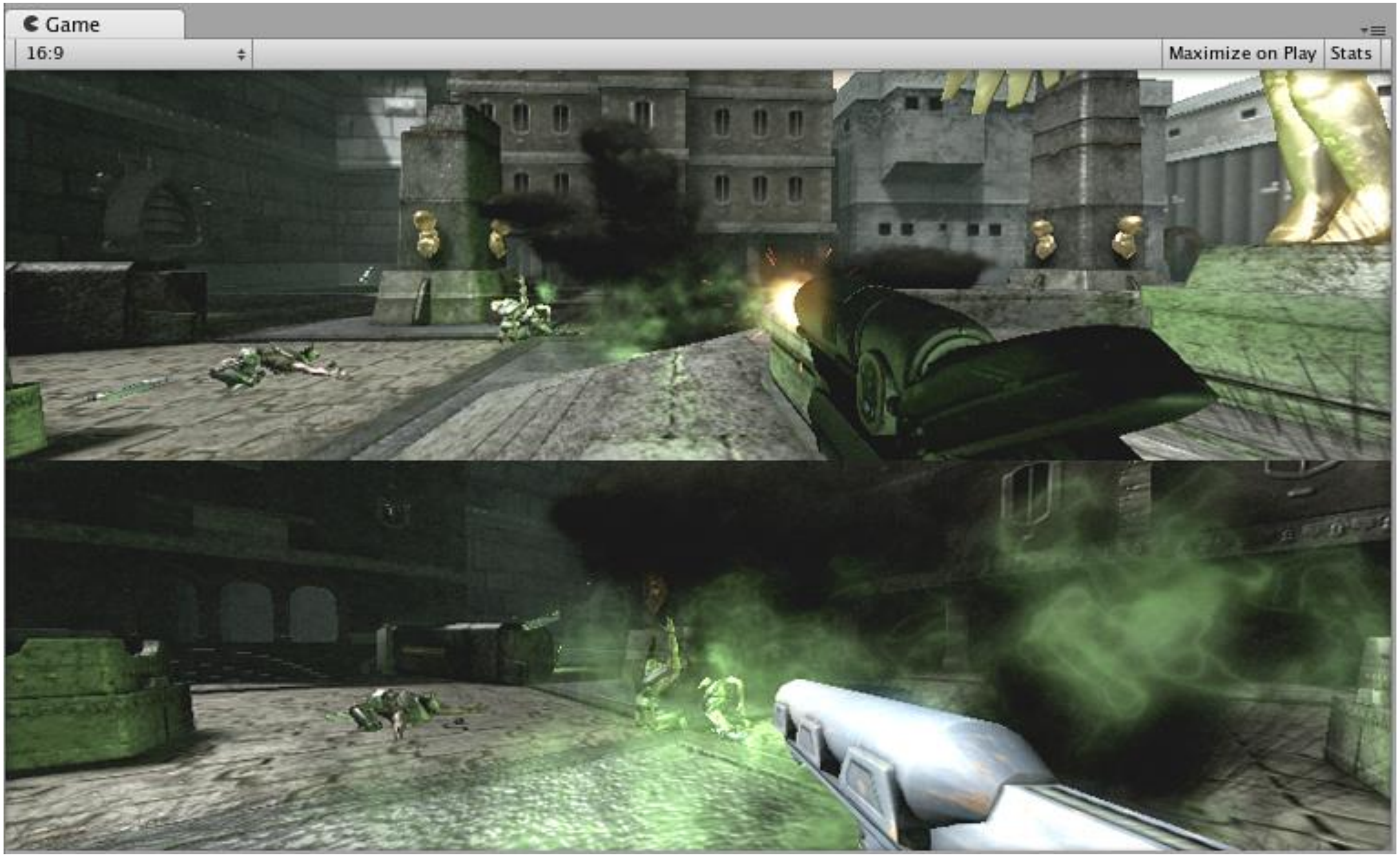
Camera Properties

- Projection
  - Perspective or Orthographic
- Size (when Orthographic is selected)
  - The viewport size of the Camera when set to Orthographic.
- Field of view (when Perspective is selected)
  - The width of the Camera’s view angle, measured in degrees along the local Y axis.
- Clipping Planes
  - Distances from the camera to start and stop rendering.
  - Near: The closest point relative to the camera that drawing will occur.
  - Far: The furthest point relative to the camera that drawing will occur.

Camera Properties

- Viewport Rect
  - Four values that indicate where on the screen this camera view will be drawn.
    - X: The beginning horizontal position that the camera view will be drawn.
    - Y: The beginning vertical position that the camera view will be drawn.
    - W (Width): Width of the camera output on the screen.
    - H (Height): Height of the camera output on the screen.
  - Measured in Viewport Coordinates (values 0 to 1).
- Depth
  - The camera's position in the draw order.
  - Cameras with a larger value will be drawn on top of cameras with a smaller value.

Viewport for two players



Camera Properties

- Target Texture
  - Reference to a Render Texture that will contain the output of the Camera view.
  - Setting this reference will disable this Camera’s capability to render to the screen.
- Target Display
  - Defines which external device to render to.
  - Between 1 and 8.

Game Physics

- Unity 3D has a built-in physics engine that deals with gravity and collisions
- The affects from this engine are very realistic
- In order to access Unity 3D’s powerful physics engine a you must first apply a rigidbody to the game object of choice.
- Rigid bodies basically tell Unity: “Hey, I need to physically react with other game objects”

Types of Objects

- Rigid Objects
  - non-deformable with physical properties (gravity, inertial)
- Non-rigid Objects:
  - Deformable: changeable geometry
  - Breakable: changeable topology
- Intangible Objects
  - No predefined shape.
  - fire, clouds, …

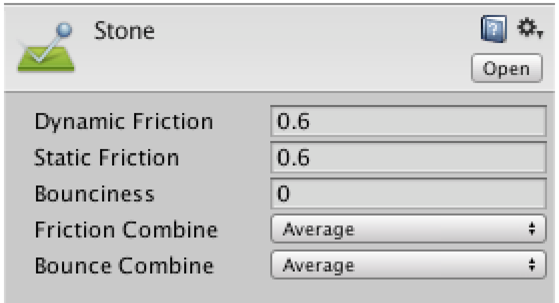
Rigidbody Component

- How to add this component?
  - Select the Game Object
  - GameObject->Component->Physics->Rigidbody

Physic Material

- Used to adjust friction and bouncing effects of colliding objects.

- To create a Physic Material
  - Select Assets > Create > Physic Material
  - Then drag the Physic Material from the Project View onto a Collider in the scene.



Stone

Dynamic Friction	0.6
Static Friction	0.6
Bounciness	0
Friction Combine	Average
Bounce Combine	Average

Physic Material Properties

- Dynamic Friction
  - The friction used when already moving.
  - Usually a value from 0 to 1.
  - A value of zero feels like ice, a value of 1 will make it come to rest very quickly unless a lot of force or gravity pushes the object.
- Static Friction
  - The friction used when an object is laying still on a surface. Usually a value from 0 to 1. A value of zero feels like ice, a value of 1 will make it very hard to get the object moving.

Physic Material Properties

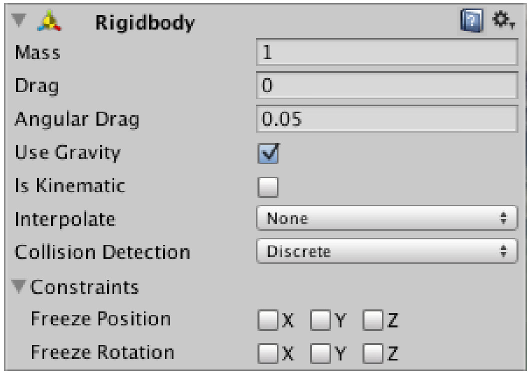
- Bounciness
  - How bouncy is the surface?
  - A value of 0 will not bounce.
  - A value of 1 will bounce without any loss of energy
- Friction Combine
  - How the friction of two colliding objects is combined.
  - Average: The two friction values are averaged.
  - Minimum: The smallest of the two values is used.
  - Maximum: The largest of the two values is used.
  - Multiply: The friction values are multiplied with each other.

Physic Material Properties

- Bounce Combine
  - How the bounciness of two colliding objects is combined. It has the same modes as Friction Combine Mode
- NOTE: Add this material to both colliding objects

Kinematic Rigidbodies

- IsKinematic value set to true or false



Rigidbody

Mass	1
Drag	0
Angular Drag	0.05
Use Gravity	☑
Is Kinematic	□
Interpolate	None
Collision Detection	Discrete
Constraints	
Freeze Position	□ X □ Y □ Z
Freeze Rotation	□ X □ Y □ Z

- Controls whether physics affects the rigidbody.
- If isKinematic is enabled, Forces, collisions or joints will not affect the rigidbody anymore.
- The rigidbody will be under full control of animation or script control by changing transform.position.

Collision Detection

- Rigidbodies don’t know how to detect collision
- Colliders detect collision
- Types of Colliders
  - Box Collider
  - Sphere Collider
  - Mesh Collider
  - Capsule Collider
  - Terrain Collider
  - Wheel Collider
  - Several others…

Colliders

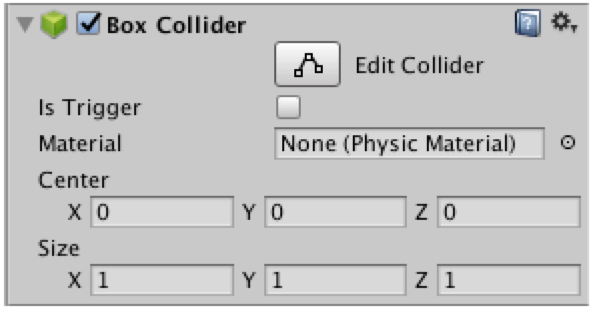
- Colliders are used simply to detect collisions
- When a collision is detected with another collider a message is sent to the physics engine and the two colliders react accordingly.

Collision Callback Functions

- Methods to react to the collision event
- OnCollisionEnter
  - Called when this collider/rigidbody has begun touching another rigidbody/collider.
- OnCollisionStay
  - Called once per frame for every collider/rigidbody that is touching rigidbody/collider.
- OnCollisionExit
  - Called when this collider/rigidbody has stopped touching another rigidbody/collider.

A Trigger Collider

- A collider could be set as a trigger
  - Check “Is Trigger”



Box Collider

Edit Collider

Is Trigger □

Material: None (Physic Material)

Center: X 0, Y 0, Z 0

Size: X 1, Y 1, Z 1

- When a collider is a trigger
  - Objects don’t bump into it
  - Objects pass through it
- Physics engine ignores such colliders
- We can detect the events using trigger callback functions

Trigger Callback functions

- OnTriggerEnter
  - Called when the Collider other enters the trigger.
- OnTriggerExit
  - Called when the Collider other has stopped touching the trigger.
- OnTriggerStay
  - Called almost all the frames for every Collider other that is touching the trigger.

OnMouseDown function

- OnMouseDown is called when the user has pressed the mouse button while over the GUIElement or a Collider.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour
{
    void OnMouseDown()
    {
        Application.LoadLevel("SomeLevel");
    }
}
```

Monobehaviour Methods

- Awake is called when the script instance is being loaded.
- Start is called on the frame when a script is enabled just before any of the Update methods is called the first time.
- Update is called every frame, if the MonoBehaviour is enabled.

More Methods: https://docs.unity3d.com/ScriptReference/MonoBehaviour.html

Monobehaviour Methods

- FixedUpdate
  - Called every fixed framerate frame, if the MonoBehaviour is enabled.
  - FixedUpdate should be used instead of Update when dealing with physics. For example when adding a force to a rigidbody, you have to apply the force every fixed frame inside FixedUpdate instead of every frame inside Update.

Monobehaviour Methods

- LateUpdate
  - Called every frame, if the Behaviour is enabled.
  - LateUpdate is called after all Update functions have been called.
  - This is useful to order script execution.
  - For example a follow camera should always be implemented in LateUpdate because it tracks objects that might have moved inside Update.

Adding Force

- Rigidbody.AddForce function
  - public void AddForce(Vector3 force, ForceMode mode = ForceMode.Force);
  - public void AddForce(float x, float y, float z, ForceMode mode = ForceMode.Force);

- Specifying the ForceMode mode allows the type of force to be changed to an Acceleration, Impulse or Velocity Change.

- Force can be applied only to an active Rigidbody. If a GameObject is inactive, AddForce has no effect.

https://docs.unity3d.com/ScriptReference/Rigidbody.AddForce.html

AddForce Example

```
using UnityEngine;
public class ExampleClass : MonoBehaviour
{
    public float thrust;
    public Rigidbody rb;

    void Start()
    {
        rb = GetComponent<Rigidbody>();
    }

    void FixedUpdate()
    {
        rb.AddForce(0, 0, thrust, ForceMode.Impulse);
    }
}
```

- This example applies an Impulse force along the Z axis to the GameObject's Rigidbody.

Summary

- Unity basics
  - Game objects, transform, camera, physics
  - Scripting
- Watch Videos
  - https://unity3d.com/learn/tutorials/s/scripting
  - https://unity3d.com/learn/tutorials/topics/physics
- Next class: Continue with Unity 3D basics