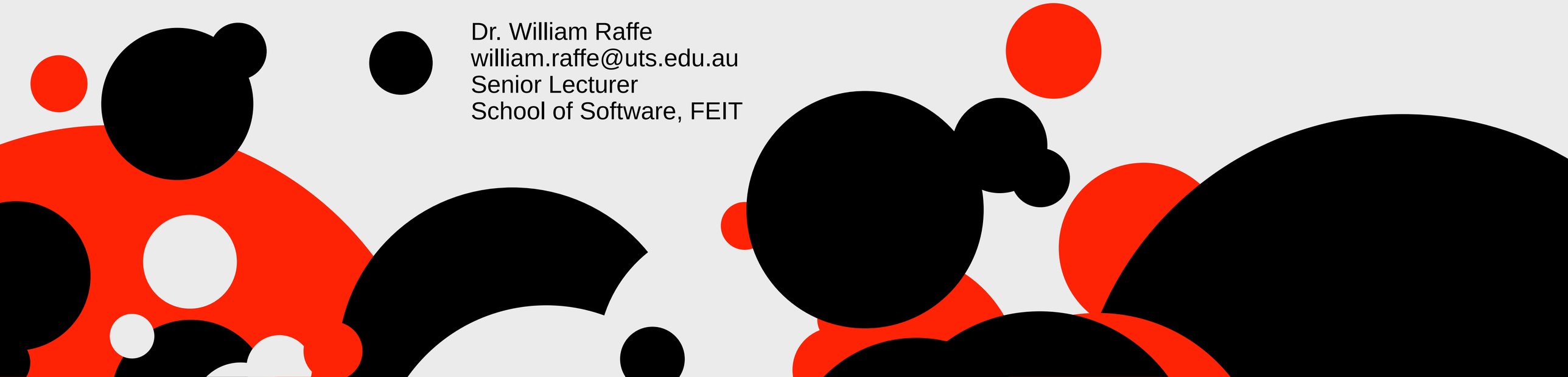
31263 / 32004
Intro to Games Development
Week 5

Dr. William Raffe
william.raffe@uts.edu.au
Senior Lecturer
School of Software, FEIT



Starting to Code in Unity (Overview)

- **Get Component**
- **Add Component**
- **Delete a Component**
- **Global (World) vs Local (Self) Space**
- **Child and Parent Relationships**
- **Position, Rotation, and Scale**
- **Color**
- **Input in Unity**

Get Component

A way to access components from this game object or other game objects.

- **Is known to be expensive, so better to store the reference in a member variable.**

GameObject.GetComponent<ComponentName>()

For example:

Renderer rend otherGO.GetComponent<Renderer>;

MyComponent temp = gameObject.GetComponent<MyComponent>();

Add Component

Add components to this game object or other game objects.

GameObject.AddComponent<ComponentName>();

For example:

gameObject.AddComponent<Rigidbody>();

otherGO.AddComponent<MyCompnent>();

Deleting Components

Deleting a Component is like deleting any other Object

- **All components inherit from “Object”. All GameObjects inherit from “Object”**
- **Calling Destroy() marks an Object for Garbage Collection**

Destroy(Object)

For example:

Destroy(gameObject);

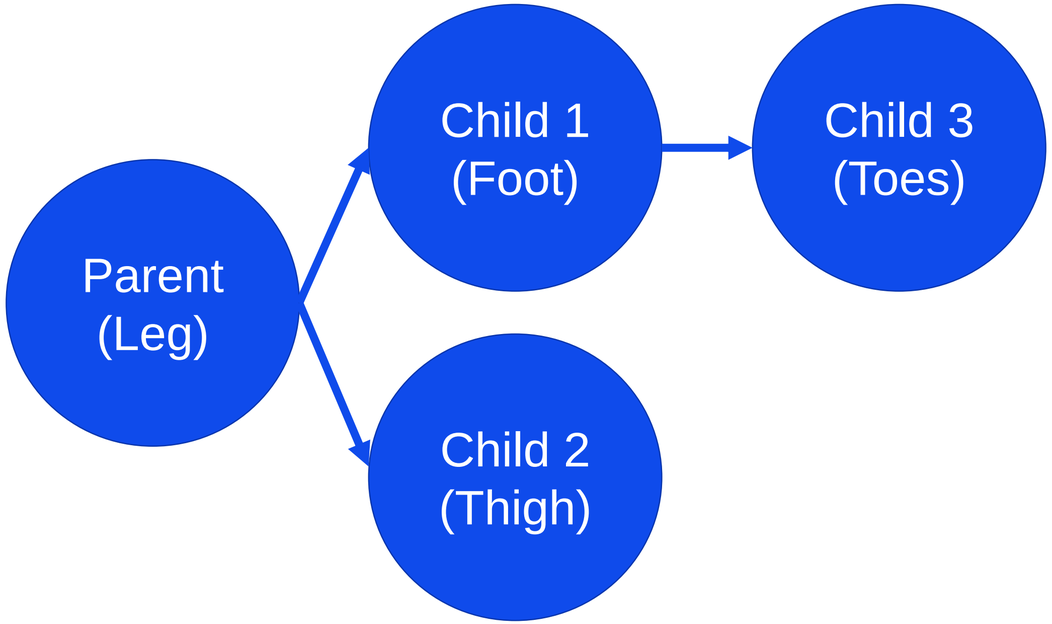
Destroy(this);

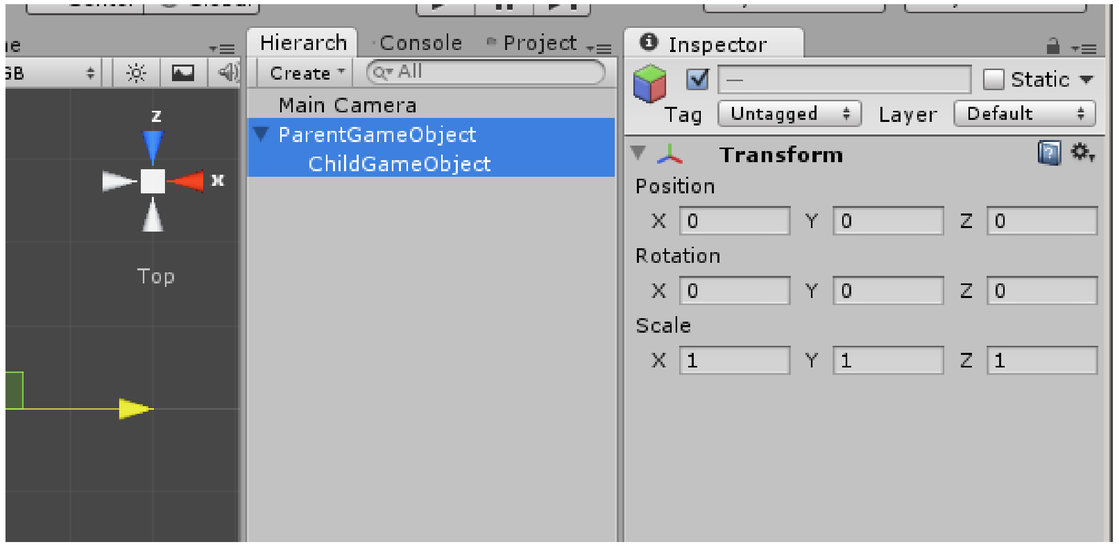
Destroy(componentReference);

Destroy(otherGO.GetComponent<MyComponent>());

Parenting

- Game objects can have a hierarchy structure.
- A game object is a “Child” of another game object if it is below it in the hierarchy.
- Transformations (scale, rotation, and translation) to a parent will also affect all of its children.
- Transformations to a child will not affect its parent.
- A child can be transformed in Local Space or World Space



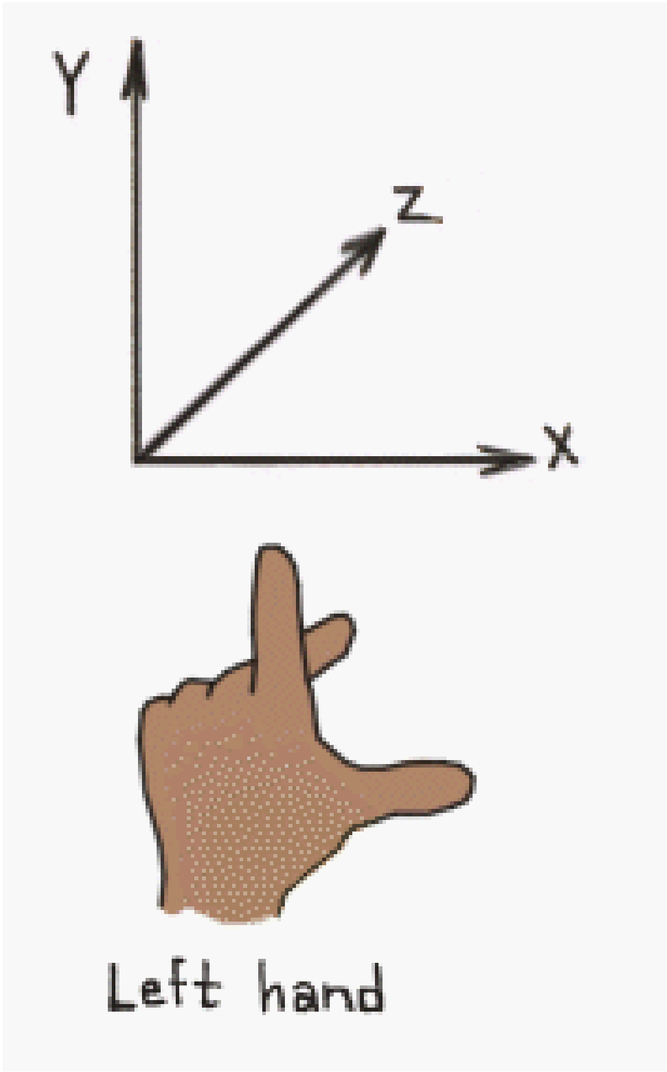


3D Coordinate Space (Reminder)

- Unity uses a **left-handed coordinate** system

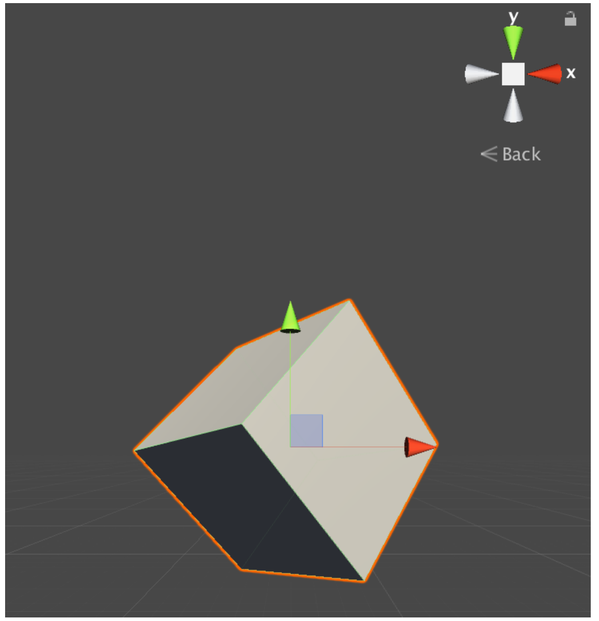
- From the default camera view
  - +x is to the right, -x to the left
  - +y is up, -y is down
  - +z is into the screen, -z is out of the screen

- There is a **world origin at x = 0.0, y = 0.0, z = 0.0**
  - Best practice: build your game around this origin
  - E.g. Player start at origin or center of level is at origin

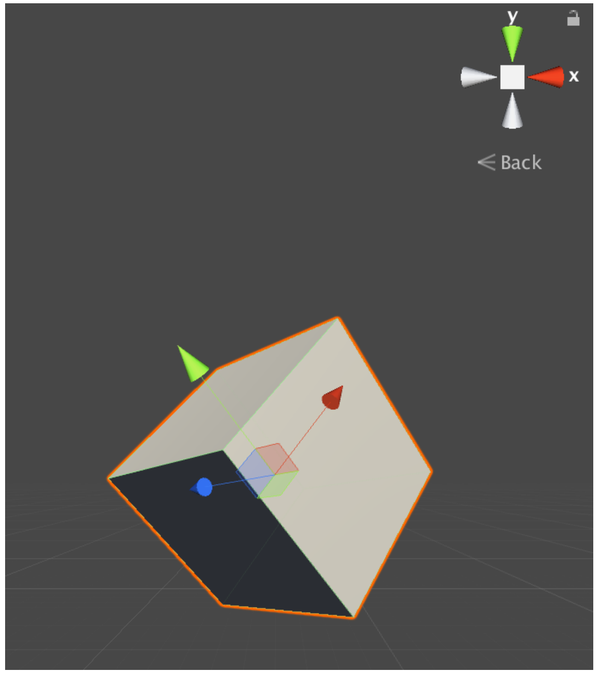


Spaces

- There are TWO main coordinate spaces:
  - **World/Global Space:** relative to the origin of the world
  - **Local/Self Space:** relative to the position, rotation, and scale of a parent object.



- World directions: **Vector3.right, Vector3.up, Vector3.forward**
  - The world (x, y, z) axes as unit vectors. E.g. Vector3.right = (1.0f, 0.0f, 0.0f)



- Local directions: **transform.right, transform.up, transform.forward**
  - The object’s local (x, y, z) axes specified as a World Vector

Position / Translation

Transform.position

Relative to world origin.

For example:

transform.position = new Vector3(2.0f, 3.0f, 4.0f);

if (transform.position.x > 5.0f)…

Transform.Translate(Vector3 distanceVector)

Relative to current position.

For example:

transform.Translate(2.0f, 3.0f, 4.0f, Space.World);

otherGameObject.transform.Translate(-1.0f, 0.0f, 157.2f, Space.Self);

Rotation

- Unity uses two different systems for rotation:
  - In the Inspector: **Euler Angles** (pronounced oy-ler) – transform.eulerAngles
  - In code: **Quaternions** – transform.rotation

- Quaternions are… complicated. They involve imaginary numbers, complex numbers, and arbitrary vectors, and a fourth dimension.
  - So why are they used?
  - They prevent **“Gimble Lock”** – An order of operations problem for rotation
  - Issues of Euler rotations and gimble lock - https://youtu.be/zc8b2Jo7mno
  - Use Unity functionality to convert from one to another

Rotation

Transform.Rotate(Vector3 eulerAngles)

Transform.RotateAround(Vector3 point, Vector3 customAxes, float angle)

Transform.LookAt(Vector3 point)

Just a few ways to rotate an object directly, all taking Euler Angles

For example:

transform.Rotate(180.0f, 90.0f, 0.0f);

transform.RotateAround(grappleTrans.position, grappleTrans.right, 45.0f);

transform.LookAt(enemyObject.transform.position);

Scale

- “Scale” = size
- **Import scale** is dependent on:
  - 2D: resolution of the sprite
  - 3D: measurement unit of a modelling program software
- All primitive Unity shapes have an import scale of 1.
  - E.g. a **cube is (1unit, 1unit, 1unit),** a sphere has diameter of 1unit, etc.
- Changing “scale” is relative to the “import size” of an object
  - Changing scale values simply multiplies the import scale

Transform.localScale

Transform.localScale = new Vector3(0.5f, 0.5f, 0.5f)

Transform.localScale *= 0.5f;

Color

There is a specific **Color class** for accessing color through code

- **Color** is wrapper around a **Vector4**
- Color in the **editor is represented as a value from (0 – 255)**
- **In code, it is a value from (0.0 – 1.0)**
- **“Alpha” is transparency**

Color(float red, float g, float b, float alpha)

For example:

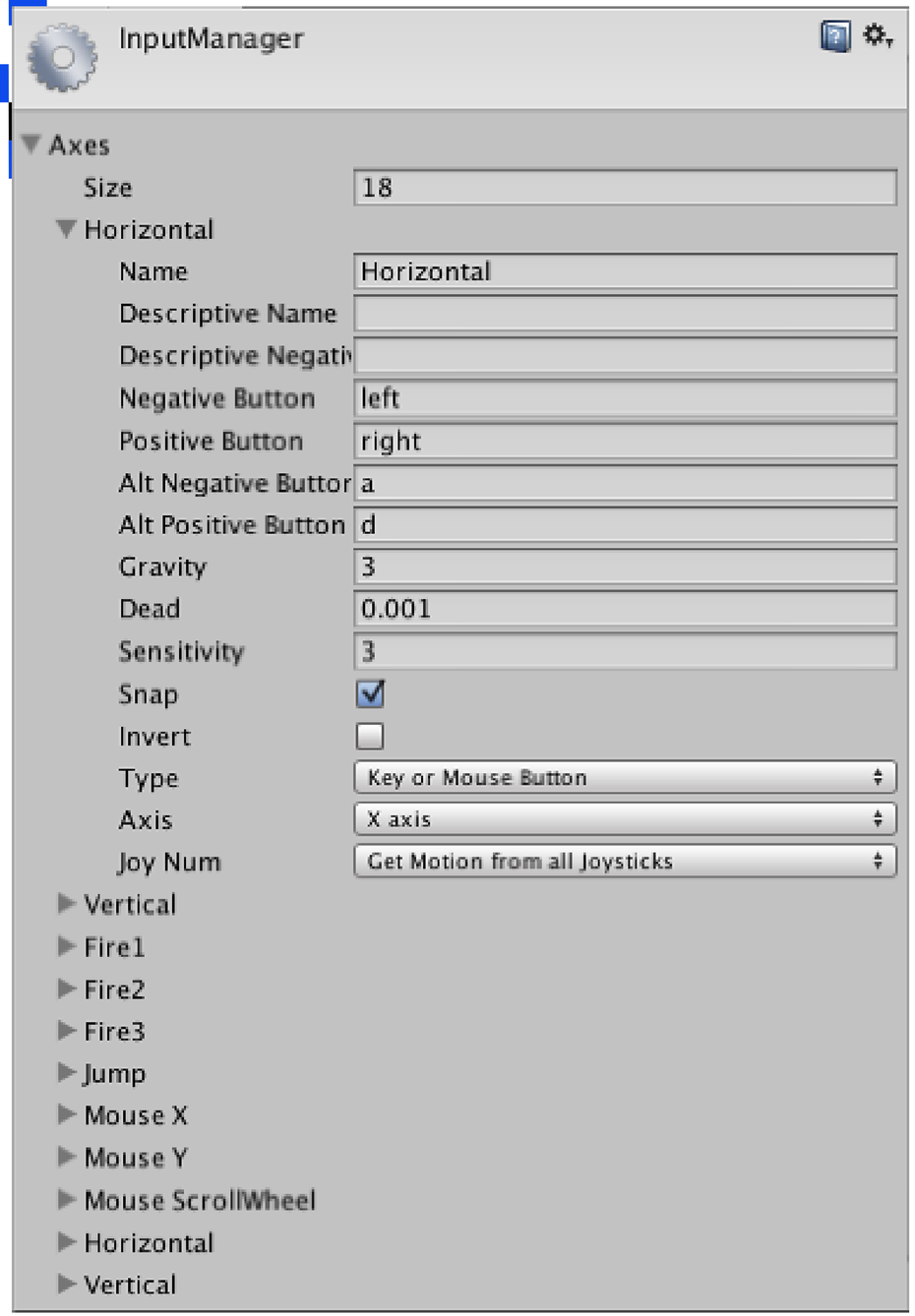
Color myColor = new Color (0.8, 0.3, 0.5, 1.0);

myMaterial.color = Color.red;

Input Manager

- Edit -> Project Settings -> Input

- These can be changed in code
  - Allows for players to set their own key bindings



Input Buttons

- **Key** – raw keyboard reference)
  - **bool Input.GetKeyDown(“a”);**
  - Not flexible for player changing their input

- **Button** (specified by Input Manager)
  - **bool Input.GetButtonDown(“InteractKey”)**
  - GetButtonDown(…) – on the frame when it is first pressed
  - GetButton(…) - when it is being held down over multiple frames
  - GetButtonUp(…) – on the frame when it is released

Input Axes

- Specified by Input Manager
- **float Input.GetAxis(“HorizontalMovement”)**
  - Specifies a positive and negative key
  - Continuous value between (-1.0, 1.0)
  - E.g. float moveSpeed = Input.GetAxis(“Horizontal”);
  - Here, holding “a” will ramp the value to -1.0f **over multiple frames**
  - **Sensitivity –** how quickly the axis reaches -1 or 1 when a key is held down
  - **Gravity –** how quickly the axis returns to 0 when the key is released
- This is important to **smooth control over objects** vs. instant acceleration
  - Also useful for **joysticks** and **triggers** that can be partially moved / held in

Mouse Input

- Mouse buttons use GetMouseButtonDown(string) or the same GetAxis() functions as before.

- Mouse position is specified in Vector2 (x,y) **Pixel Space**
  - These are the pixels of your monitor
  - Gives the exact position on the monitor that the player is clicking
  - Usually needs to be **converted to world space** by using the current perspective of the in-game camera before it is used

Vector3 Input.mousePosition

For example:

```
Vector3 worldPoint = Camera.main.ScreenToWorldPoint(Input.mousePosition)
```

Note that ScreenToWorldPoint expects a Vector3, where z is the distance from the camera due to the non-parallel view of perspective cameras (see Week 2 lecture). With a Vector2, you are getting the world-point on the same z-plane as the camera, which may not be what you want. This is NOT the best way to know what gameobject a player is clicking on, see RayTracing in a few weeks time.