CPSC 599/601
A2, Embodied Interaction, and AndroidJava Classes

Slides by
Jessi Stark

Today's Agenda

- A2 Intro
- Sensor input overview
- Example: Gyroscope
- Intro to Android Java development via Unity C#

A2 Intro.

Assignment Two

Build on your A1 Submission. Implement "embodied" interaction. For example, import a speech recognition asset or use built-in sensors on your device like accelerometers, gyroscopes, proximity sensors, etc. Get creative! You might control your character with your own movements, you might cause events to happen if you move closer or farther from your targets or your character. Implement at least two types of embodied interaction that affect the progress of your character.

Assignment Two

Build on your A1 Submission. Implement "embodied" interaction. For example, import a speech recognition asset or use built-in sensors on your device like accelerometers, gyroscopes, proximity sensors, etc. Get creative! You might control your character with your own movements, you might cause events to happen if you move closer or farther from your targets or your character. Implement at least two types of embodied interaction that affect the progress of your character.

Due March 1.

A note about A2 interaction…

- Embodied interaction should happen off the screen
- Any interactions via on-screen UI elements will not count towards your completion of this assignment

A note about A2 submissions…

- Include a readme file describing which interaction methods you've implemented, and what they are used for in your app
- **Include a video of your app running**

Sensor Input Overview.

Android Supported Sensors

- Accelerometer
- Ambient temperature
- Magnetic field sensor
- Gyroscope
- Heart Rate
- Light
- Proximity
- Pressure
- Relative Humidity

Android Supported Sensors

- Accelerometer
  - How fast the device is accelerating along three axes in m/s^2
- Ambient temperature
- Magnetic field sensor
- Gyroscope
- Heart Rate
- Light
- Proximity
- Pressure
- Relative Humidity

Android Supported Sensors

- Accelerometer
- Ambient temperature
  - The temperature of the room in degrees Celsius
- Magnetic field sensor
- Gyroscope
- Heart Rate
- Light
- Proximity
- Pressure
- Relative Humidity

Android Supported Sensors

- Accelerometer
- Ambient temperature
- Magnetic field sensor
  - The ambient magnetic field measured along three axes in micro-Tesla
- Gyroscope
- Heart Rate
- Light
- Proximity
- Pressure
- Relative Humidity

Android Supported Sensors

- Accelerometer
- Ambient temperature
- Magnetic field sensor
- Gyroscope
  - The rate of rotation around three axes in rad/s
- Heart Rate
- Light
- Proximity
- Pressure
- Relative Humidity

Android Supported Sensors

- Accelerometer
- Ambient temperature
- Magnetic field sensor
- Gyroscope
- Heart Rate
  - The heart rate of the user in BPM
- Light
- Proximity
- Pressure
- Relative Humidity

Android Supported Sensors

- Accelerometer
- Ambient temperature
- Magnetic field sensor
- Gyroscope
- Heart Rate
- Light
  - The illumination of the room measure in SI lux units
- Proximity
- Pressure
- Relative Humidity

Android Supported Sensors

- Accelerometer
- Ambient temperature
- Magnetic field sensor
- Gyroscope
- Heart Rate
- Light
  - The illumination of the room measure in SI lux units
- Proximity
- Pressure
- Relative Humidity

Android Supported Sensors

- Accelerometer
- Ambient temperature
- Magnetic field sensor
- Gyroscope
- Heart Rate
- Light
- Proximity
  - The distance to the nearest visible surface, measured in cm
- Pressure
- Relative Humidity

Android Supported Sensors

- Accelerometer
- Ambient temperature
- Magnetic field sensor
- Gyroscope
- Heart Rate
- Light
- Proximity
  - Some proximity sensors provide only binary feedback indicating if the surface is “near” or “far”
- Pressure
- Relative Humidity

Android Supported Sensors

- Accelerometer
- Ambient temperature
- Magnetic field sensor
- Gyroscope
- Heart Rate
- Light
- Proximity
- Pressure
  - Measures the atmospheric pressure in hectaPascal
- Relative Humidity

Android Supported Sensors

- Accelerometer
- Ambient temperature
- Magnetic field sensor
- Gyroscope
- Heart Rate
- Light
- Proximity
- Pressure
- Relative Humidity
  - Relative ambient humidity of the room provided as a percentage value

Android Supported Sensors

Sensors available to you will depend on the device you are using.

Composite Sensors

- Some composite sensors are provided in the Android API
- These “sensors” are objects in code that combine data from multiple sensors or pre-converted sensor data
  - e.g. Game rotation -> Accelerometer + Gyroscope
  - e.g. Step counter -> Accelerometer data
- Handy!

Accessing Sensor Data

- Unity provides access to some sensors directly through their Input class
  - e.g. Input.acceleration, Input.gyro
- You may have to access certain Android features via Android Platform APIs (more on this later…)
  - Java, but in C#
  - It’s weird

More details about Android sensors

- Sensor types:
  - https://source.android.com/devices/sensors/sensor-types
- All kinds of stuff about sensors and sensing:
  - https://source.android.com/devices/sensors

Example: Gyroscope.

Gyroscope Rotation

- Right-handed rotation
- Unity uses left-handed rotation
- So, flip z and w

```
private static Quaternion GyroToUnity(Quaternion q)
{
    return new Quaternion(q.x, q.y, -q.z, -q.w);
}

void Update()
{
    //rotate on all axes using gyroscope
    Quaternion deviceRotation = GyroToUnity(Input.gyro.attitude);
    transform.rotation = deviceRotation;
}
```

Gyroscope Rotation

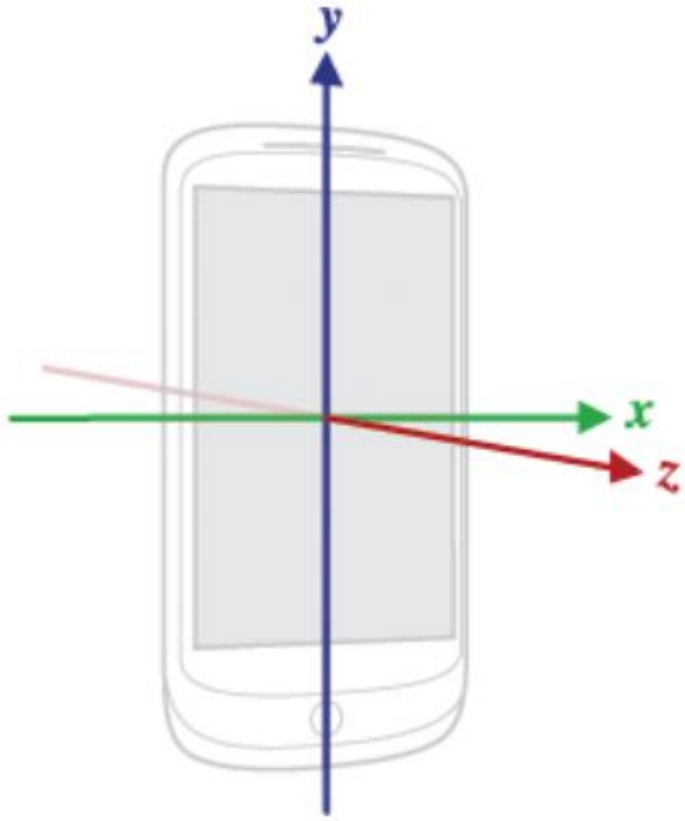
- Doing this will likely cause the game object to wobble around all over the place
- We probably don’t want that

Gyroscope Rotation

- Rotations are represented as Quaternions, a combination of real and imaginary numbers
- Do you remember linear algebra?
  - Me neither.
- Luckily, we can easily convert the Quanternion to a Vector3 to understand the rotation around the x, y, and z axes more easily

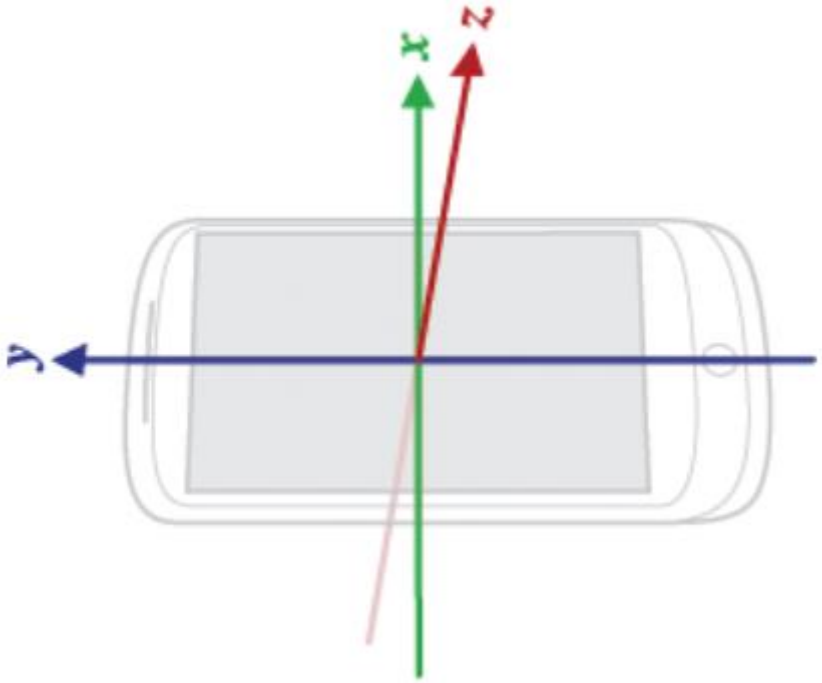
Relative Local Axes

- The application runs in landscape mode



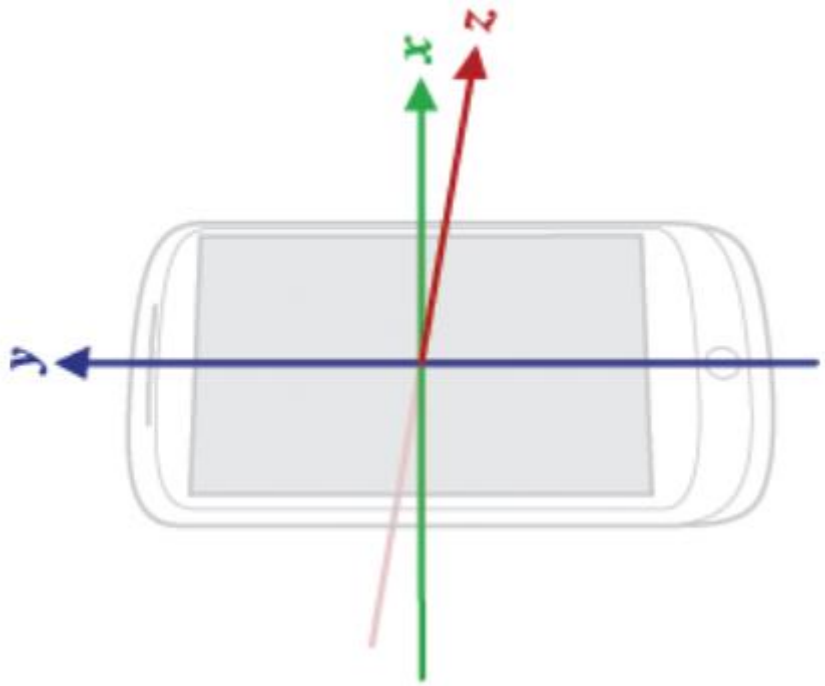
Relative Local Axes

- The application runs in landscape mode



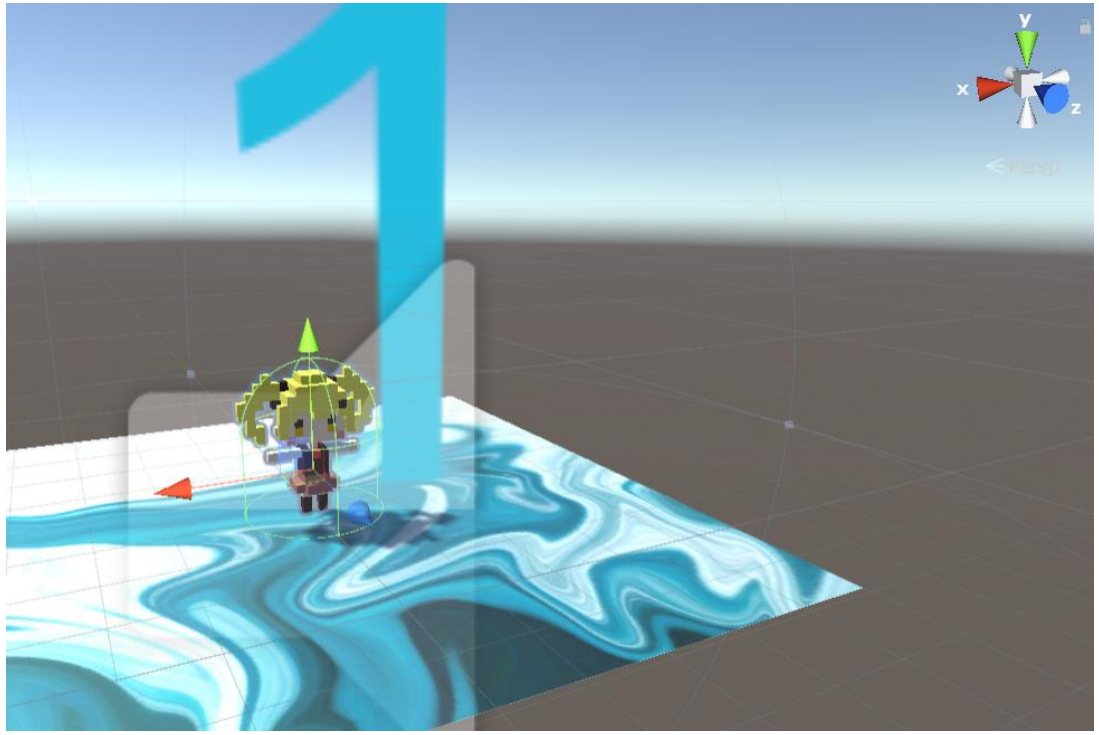
Relative Local Axes

- The application runs in landscape mode
- Since the camera looks down toward the character, the x axis points forward while the z axis points down



Relative Local Axes

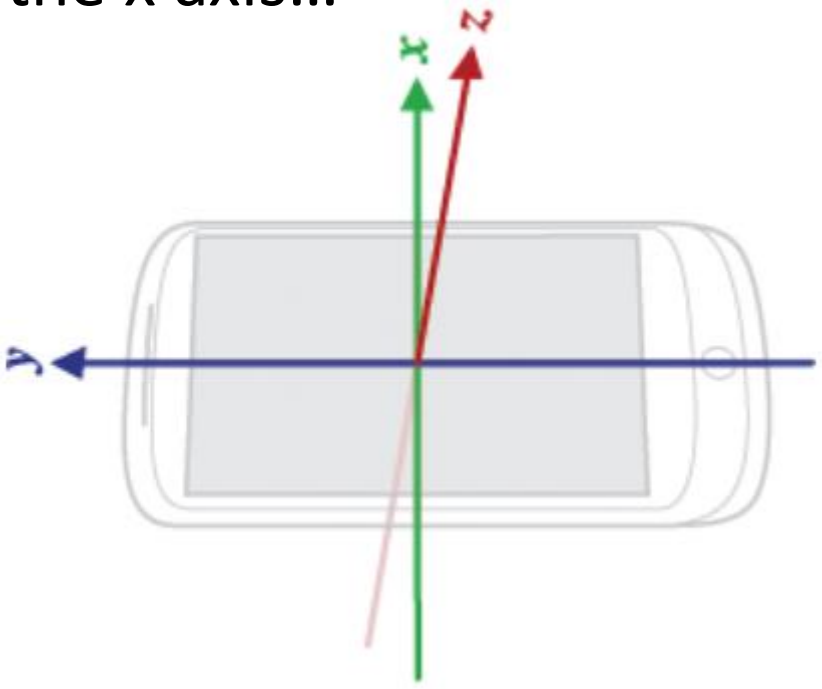
- The character’s local y axis points up
- If we want her to stay upright but be able to turn, we should rotate her only around the y axis



Relative Local Axes

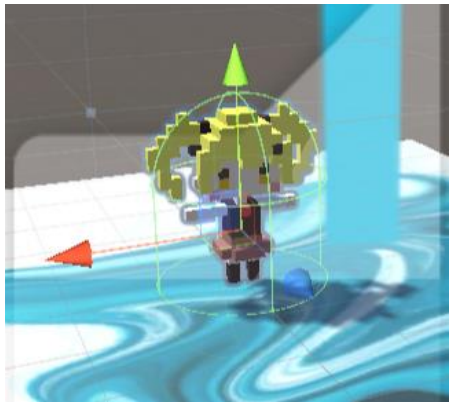
Device

- When I change the direction of the x axis…



Character

- The character should rotate around the y axis (green)



Gyroscope Rotation

- In this case, the device's local x axis corresponds to the character's local y axis
- So, apply a rotation around the character's local y axis using the angle of the device's local x axis
- Keep the value of the character's original x and z rotations
- Multiply by the desired rotation speed

```
void Update()
{
    //rotate on y axis using gyroscope
    Quaternion deviceRotation = GyroToUnity(Input.gyro.attitude);
    Vector3 deviceEulerAngles = deviceRotation.eulerAngles;
    Vector3 transformEulerAngles = transform.rotation.eulerAngles;
    transform.Rotate((new Vector3(transformEulerAngles.x, deviceEulerAngles.x, transformEulerAngles.z)) * (0.1f * Time.deltaTime));
}
```

World Center

- Since we want the movement to occur relative to the camera, we should change the world center to “Camera”
- This setting is on the ARCamera GameObject

Intro to Android Java development via Unity C#.

Accessing Android Java APIs

- Via three main Unity C# classes:
  - AndroidJavaClass
  - AndroidJavaObject
  - AndroidJavaProxy
- These act as an interface between your C# code and Android's Java APIs
- Add a using directive to your class:
  - using UnityEngine.Android;

AndroidJavaClass

- This C# class can allow you to store a reference to a Java class
- Equivalent to generic class Class in Java
- Handy for using static variables and methods
- Create a new AndroidJavaClass object and define the class name as a String in the constructor
- This example creates a reference to a Java Class object
  - NOT a SpeechRecognizer object

```
AndroidJavaClass recognizerClass = new AndroidJavaClass("SpeechRecognizer");
```

AndroidJavaObject

- This C# class can allow you to store a reference to a Java object
- Equivalent to generic class Object in Java
- Access its instance variables and methods
- Create a new AndroidJavaObject object and define the class name as a String in the constructor
- This example creates a reference to a Java Intent object

```
AndroidJavaObject intent = new AndroidJavaObject("Intent");
```

Method Arguments and Return Values

- Example:
- Java method signature for method called in first line:
- `public static Context getApplicationContext()`
  - In Context class
- The static method takes no arguments and returns a Context object
- This code stores the return value as an AndroidJavaObject

```
AndroidJavaObject context = contextClass.CallStatic<AndroidJavaObject>("getApplicationContext", new object[] { });
AndroidJavaObject recognizer = recognizerClass.CallStatic<AndroidJavaObject>("createSpeechRecognizer", new object[] { context });
```

Method Arguments and Return Values

- Example:
- Java method signature for method called in second line:
- `public static SpeechRecognizer createSpeechRecognizer(Context c)`
  - In SpeechRecognizer class
- The static method takes one argument (a Context object) and returns a SpeechRecognizer object
- This code passes the Context object to the method as an AndroidJavaObject and stores the return value as an AndroidJavaObject

```
AndroidJavaObject context = contextClass.CallStatic<AndroidJavaObject>("getApplicationContext", new object[] { });
AndroidJavaObject recognizer = recognizerClass.CallStatic<AndroidJavaObject>("createSpeechRecognizer", new object[] { context });
```

Method Arguments and Return Values

- If the Java method return type can be mapped directly to a C# type, we can accept it as that
  - e.g. primitives
- Same goes for passing arguments
- Example:

```
AndroidJavaObject javaArray = results.Call<AndroidJavaObject>("getStringArrayList", "SpeechRecognizer.RESULTS_RECOGNITION");
if (javaArray.Call<Boolean>("contains", new object[] { "start" })) {
```

- Here, we directly pass a String to the Java method and directly accept the returned Boolean value (no need to use AndroidJavaObject for these values)

AndroidJavaProxy

- Used to reference Java interfaces
- Implement this interface in your class the same way you'd implement any C# interface
- Constructor must define the base interface type

```
public class SpeechListener : AndroidJavaProxy
{
    public SpeechListener() : base("android.speech.RecognizerListener")
    {
        //stuff happens here
    }
}
```

AndroidJavaProxy

- When you implement an interface, you must define method bodies for all of its abstract methods in the implementing class
  - (This is always true, not just here)
- In this case, the method names need to exactly match the Java method names, including casing
- Primitive arguments can be primitives, but Objects should be of type AndroidJavaObject in your method signature

```
public void onEvent(int eventType, AndroidJavaObject param)
{

}
```

Extending Classes

- As in all C# programs, you can extend AndroidJavaObject classes the same way you implement AndroidJavaProxy interfaces.
- Extend AndroidJavaObject in the class definition, and create a constructor that defines the name of the base class

Useful Links

- Android API: https://developer.android.com/docs/
- Unity AndroidJava…: https://docs.unity3d.com/ScriptReference/AndroidJavaClass.html
- Android Sensors: https://source.android.com/devices/sensors
- Speech regonition github repo (not mine): https://github.com/gsssrao/UnityAndroidSpeechRecognition

The End.

Questions?