Session 03 - 01/21/20

Introducing `C#`

Learning a new programming language doesn't always have to be like starting over

- `C#` and `JavaScript` both have "C-Like" syntax:
    - `{` and `}` to define blocks of scope
    - functions defined with `(` and `)`, optionally with parameters passed inside
    - `;` at end of expressions
    - whitespace doesn't (usually) matter
    - `if`, `else`, `while`, `for`, `switch` and many other basic logical constructs work exactly the same
- Even when syntax is different...
    - Python
        - no `{`, or `}`
        - whitespace matters
    - Haskell
        - "declarative" language
    - BrainFuck
        - ???
- ...it doesn't change the fact that programming is **how you think about the problem**.
    - APIs change over time - by the time you've memorized it, the definition changes
        - For example: Unity recently unified all `AR` and `VR` classes into a single `XR` name space
    - Looking up the API isn't "cheating"
        - Unity Scripting Reference
            - Look under `UnityEngine > Classes`
        - Most frustrating thing about learning new language is not knowing what's already there for you.

C# vs JavaScript

- Most immediate difference you'll notice is that `C#` is **statically typed** while `JavaScript` is **dynamically typed**
    - variables start off as a specific thing, and have to stay that way.
    - Unlike `JavaScript` where you can do `var x = 12` and then later say `x = "hello"`
- This means we have to be a little more specific when we do things in `C#`
- Basic Programming Concepts in `C#`
    - variables
        - "Primitive" types like `int`, `float`, `bool`
        - Also "instances" of more complex types like `Vector3`,
        - Can *cast* into other types with `()`
            - `int myInt = 4;`
            - `float myFloat = 25.0f;`
            - `int myOtherInt = myInt / (int) myFloat;`
    - functions
        - passing parameters
            - overloading (3 floats vs a `Vector3`)
        - "returning" a value
    - scope (inside of `{` and `}`)
    - operations and assignment (from function)
    - The `.` operator
        - `Random.Range`

C# in Unity

- Scripts must be placed in hierarchy to become active
    - Multiple copies of scripts act independently
- File name and Class name **must be the same**
- `Debug.Log()` **is your friend!**
- `public` vs `private`

- Working with **Components** in your scripts
    - `GameObject` vs `gameObject`
    - `GetCompnent<>()` vs `GetComponent()`
        - The `<>` represent generic `Type`
        - You (the programmer) indicate type at runtime and the returned object immediately is cast correctly.
        - This means you can do something like this:
            - `gameObject.GetComponent<Rigidbody>().useGravity;`
            - i.e. access the `useGravity property right away
        - Alternatively, the non-generic version requires you to know the String name of the component
            - `gameObject.GetComponent("Rigidbody");`
            - Prone to misspellings & does not benefit from autocorrect;
            - Cannot access properties (i.e. `useGravity`) until assigned to correct variable type
            - Therefore I suggest not to use
- The `Input` class
    - Used to capture mouse/keyboard/gamepad input