ImpactLab - Game Dev
Lecture 2: Computational Thinking (and C#)

Summer 2024

School of Computing
and Data Science

Wentworth Institute of
Technology

C# (Pronounced C - sharp)

- C# is very similar to Java.
- Much of what you'll see today will be very familiar if you've used Java.
- These days, Unity exclusively uses C#, so no other language will work for this session.
- Unity does not always implement the latest C# standard.

While Oracle is the corporate backer of Java, Microsoft backs C#. They can do largely the same things, it just depends on the ecosystem you want to be a part of.

Common Variables Types

- Integer

```
int myInt = 5;
```

- Float

```
float myFloat = 5.9f;
```

- Double

```
double myDouble = 5.9;
```

- Character

```
char myChar = 'A';
```

- Boolean

```
bool myBool = true;
```

- String

```
string myStr = "Hello";
```

Notes:

- Floats will be very common in Unity, always put an **f** after the number when defining a float.
- **string** does not have a capital S like in Java.

Type Casting

- Implicit Casting (Automatic)

```
char -> int -> long -> float -> double
```

- Explicit Casting (Manual)

```
double -> float -> long -> int -> char
```

Explicit Cast

```
double myDouble = 10.21;
int myInt = (int) myDouble;

Console.WriteLine(myDouble);
Console.WriteLine(myInt);
```

Convert Class

```
int myInt = 10;
double myDouble = 5.5;
bool myBool = true;

Console.WriteLine(Convert.ToString(myInt));
Console.WriteLine(Convert.ToDouble(myInt));
Console.WriteLine(Convert.ToInt32(myDouble));
Console.WriteLine(Convert.ToString(myBool));
```

Operators

Mostly the Same As Any Other Language

```
//Arithmetic Operators
 +  //Addition
 -  //Subtraction
 *  //Multiplication
 /  //Division
 %  //Modulus
 ++ //Increment
 -- //Decrement

//Assignment Operators
 =  //Assignment
 += //x+=5 -> x=x+5
 -= //x-=5 -> x=x-5
 *= //x*=5 -> x=x*5
 /= //x/=5 -> x=x/5
 %= //x%=5 -> x=x%5
 &= //x&=5 -> x=x&5
```

```
//Comparison Operators
 ==  //Equal to
 !=  //Not Equal to
 >   //Greater Than
 <   //Less Than
 >=  //Greater Than or Equal to
 <=  //Less Than or Equal to

//Logical Operators
 &&  //Logical and
 ||  //Logical or
 !   //Logical not
```

There are the normal bitwise operators as well

Control Flow

if-else

```
if (condition1)
{
  //executed if condition1 is True
}
else if (condition2)
{
  //executed if the condition1 is false and condition2 is True
}
else
{
  // executed if the condition1 is false and condition2 is False
}
```

Ternary Operator

```
variable = (condition) ? expressionTrue :  expressionFalse;
```


Control Flow

switch

```
switch(expression)
{
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
    break;
}
```

while

```
while (condition)
{
  // code block to be executed
}
```

```
do
{
  // code block to be executed
}
while (condition);
```

for

```
for (statement 1; statement 2; statement 3)
{
  // code block to be executed
}
```

```
foreach (type variableName in arrayName)
{
  // code block to be executed
}
```

- **default** is optional in the **switch**
- **statement1** is initialization
- **statement2** is condition
- **statement3** is increment
- **foreach** can be used on any array

Strings

Strings always use double quotes and have a number of properties and methods to make your life easier.

```
string txt = "Hello World";
Console.WriteLine("The length of the txt string is: " + txt.Length);

Console.WriteLine(txt.ToUpper());   // Outputs "HELLO WORLD"
Console.WriteLine(txt.ToLower());   // Outputs "hello world"

string firstName = "John ";
string lastName = "Doe";
string name = firstName + lastName;
Console.WriteLine(name);  //outputs "JohnDoe"

string name = $"My full name is: {firstName} {lastName}";
Console.WriteLine(name); //outputs "My full name is John Doe"

string myString = "Hello";
Console.WriteLine(myString[0]);  // Outputs "H"
Console.WriteLine(myString.IndexOf("e"));  // Outputs "1"
```

Escape Characters

\', \", \\, \n, \t, \b

Arrays

Array Creation

```
string[] cars;
string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
int[] myNum = {10, 20, 30, 40};
int[] myNums = new int[10]
```

Lots of ways to create arrays.

Looping

```
string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (int i = 0; i < cars.Length; i++)
{
  Console.WriteLine(cars[i]);
}

foreach (string s in cars)
{
  Console.WriteLine(s);
}
```

for and foreach loop examples.

There are lots of methods associated with arrays, like **sort**, **min**, **max**, **sum**, etc.

Methods

C# code for methods typically use **PascalCase** as opposed to **camelCase**. However, you'll often see me use **camelCase**.

It doesn't matter to me which you use, as long as you're consistent.

```
static void MyMethod()
{
  Console.WriteLine("I just got executed!");
}

static void Main(string[] args)
{
  MyMethod();
}
```

```
static void MyMethod(string fname)
{
  Console.WriteLine(fname + " Refsnes");
}
```

static methods can be called using the class name

Parameters are how we send variables into methods.

Methods

```
static void MyMethod(string country = "USA")
{
  Console.WriteLine(country);
}

static void Main(string[] args)
{
  MyMethod("UK");
  MyMethod("Germany");
  MyMethod();
  MyMethod("Canada");
}
```

C# has easy default parameters

```
static int PlusMethod(int x, int y)
{
  return x + y;
}

static double PlusMethod(double x, double y)
{
  return x + y;
}
```

Overloading is allowed.

Basic Classes

```
class Program
{
  static void Main(string[] args)
  {
    Car Ford = new Car("Mustang", "Red", 1969);
    Car Opel = new Car("Astra", "White", 2005);

    Console.WriteLine(Ford.model);
    Console.WriteLine(Opel.model);
  }
}
```

```
class Car
{
  public string model;
  public string color;
  public int year;

 public Car(string modelName, string modelColor, int modelYear)
  {
    model = modelName;
    color = modelColor;
    year = modelYear;
  }
}
```

Same class style as most other OOP languages. Typically just one class per file.

Access modifiers: **public**, **private**, **protected**, **internal**

Classes - Properties

```
class Person
{
  private string name; // field
  public string Name   // property, Capitalized
  {
    get { return name; }
    set { name = value; }
  }
}

class Program
{
  static void Main(string[] args)
  {
    Person myObj = new Person();
    myObj.Name = "Micah";
    Console.WriteLine(myObj.Name);
  }
}
```

private variable are only accessible within the same class, a C# **property** is like a getter/setter method.

New C# Script in Unity

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


public class MyClass : MonoBehaviour
{
  // Start is called before the first frame update
  void Start()
  {

  }

  // Update is called once per frame
  void Update()
  {

  }
}
```

When you create a new script, this is the default code that you are given

Any class that you add to a gameobject in Unity must inherit from **MonoBehavior**.

Start() runs when the object is created and **Update()** runs every frame. These methods are part of **MonoBehavior**.

You can always make regular classes that do not inherit from **MonoBehavior**

New C# Script in Unity

- Each script (that you attach to a gameobject) is a **component** of that object.
- It has access to information about that object (via **this**) and access to other components attached to that object.
- There are lots of little quirks when it comes to accessing and changing these object components via script.

```
// Update is called once per frame
void Update()
{
  Vector3 pos = this.transform.position;
  pos.x += 0.5f;
  this.transform.position = pos;
}
```

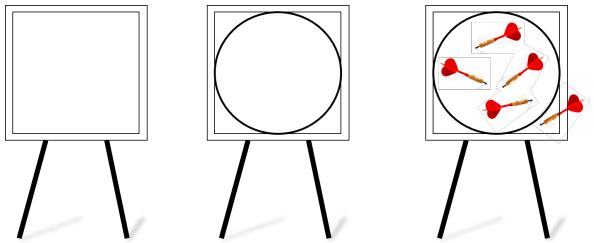
Grab the full position (**Vector3**), update the x value, save the full position back.

Updates **this** object's x position every frame.

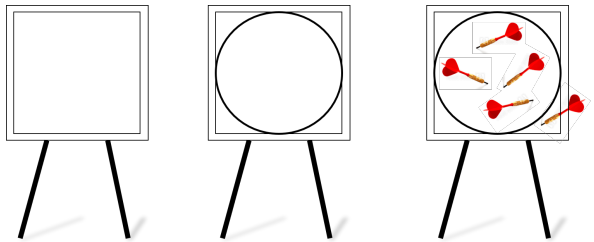
Computational Exercise 1: Discovering PI

- PI is the ratio of a circle's circumference to its diameter.
- Its approximate value is 3.14159265...

There are many ways to estimate its value, but today, we're going to use the "Dart Board" approach.



Discovering PI



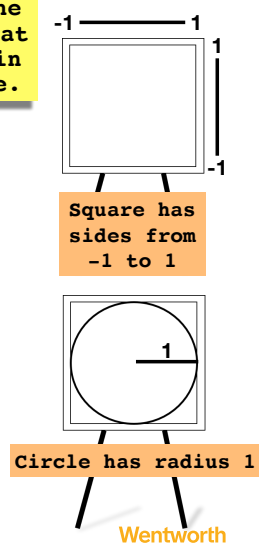
The Idea is simple:

- Start with a square.
- Add a circle inside (exactly touching the edges).
- Throw darts at the square randomly.
- Count how many land in the circle vs how many total darts you throw.
- The result (hits in the circle/total darts) will be related to PI.

Throw a Dart

```
float x=Random.Range(-1.0f,1.0f);
float y=Random.Range(-1.0f,1.0f);
```

"Throw" the dart so that it lands in the square.

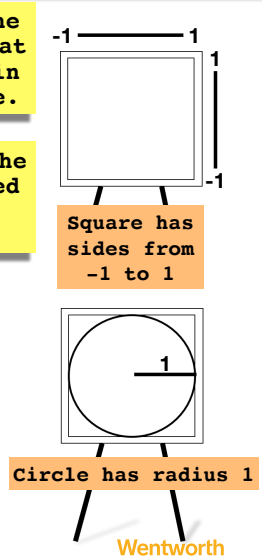


Throw a Dart

```
float x=Random.Range(-1.0f,1.0f);
float y=Random.Range(-1.0f,1.0f);


if(Mathf.Sqrt(x*x+y*y)<=1){
  inCircle+=1;
}
```

"Throw" the dart so that it lands in the square.

Check if the dart landed in the circle



Compare to Circle

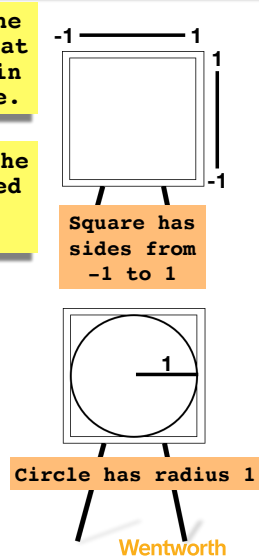
```
float x=Random.Range(-1.0f,1.0f);
float y=Random.Range(-1.0f,1.0f);


if(Mathf.Sqrt(x*x+y*y)<=1){
  inCircle+=1;
}
```

"Throw" the dart so that it lands in the square.

Check if the dart landed in the circle

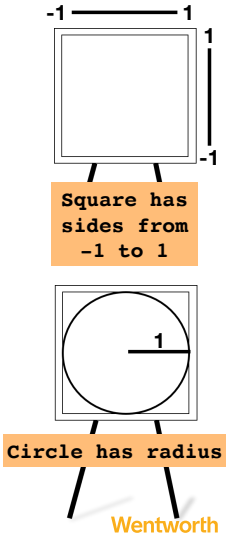
Do this many times (in a loop) and report how many landed in the circle.

Divide by the total number of darts thrown.



Do Many Times

```
int N=1000000;
int inCircle=0;

for(int i=0;i<N;i++)
{
  float x=Random.Range(-1.0f,1.0f);
  float y=Random.Range(-1.0f,1.0f);


  if(Mathf.Sqrt(x*x+y*y)<=1){
    inCircle+=1;
  }
}
```



Why Does This Work (And What to Print)

Area of a Circle:

$$A_c = \pi r^2$$

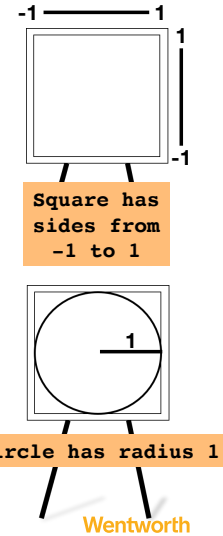
Area of a Square:

$$A_s = D^2 = 4r^2$$

Ratio:

$$\frac{A_c}{A_s} = \frac{\pi r^2}{4r^2} = \frac{\pi}{4}$$

So, if we throw enough darts, they approximate the area of both the square and circle. → $\pi = 4\frac{H}{N}$



Computational Exercise 2: Let's Make a Deal

Let's Make a Deal is a game show that started in 1963 and has appeared in various forms up to today.

The final segment of the original show was "The Big Deal" and it's what we're going to be simulating today.

Monte Hall

Our goal today is to gain **insight** about this final game show segment.

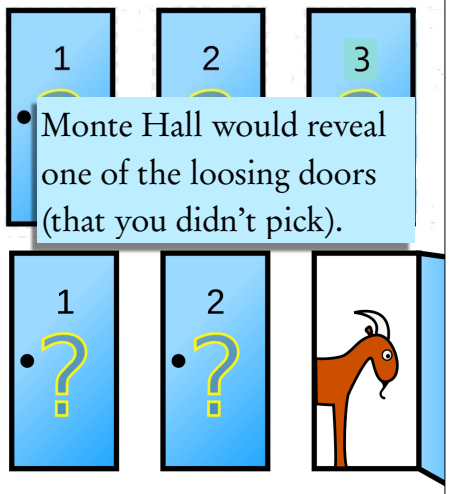
The Big Deal

The final segment of the show always contained a big item, like a car, vacation, etc.

The player would select one of three doors.

Behind two would be a goat (or similar), but behind one was the prize.

He then gives you the chance to change your pick.



The Big Deal

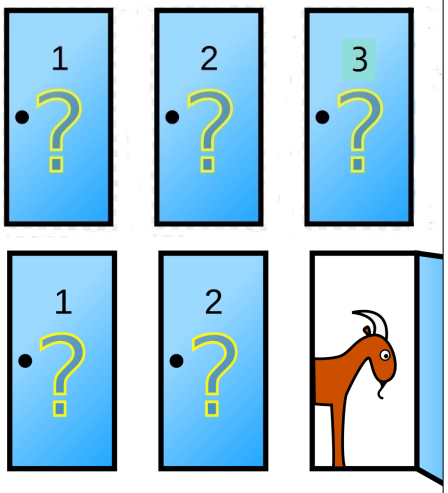
Let's say you pick door 2.

Monte Hall shows you what's behind door 3.

Do you change your pick to door 1?

Is it always better to switch or stay?

What is the chance of winning if you stay or if you switch?



The Big Deal

First, let's think about how we would run this game once on our computer:

```
int winningDoor = Random.Range(0,3);
int playerDoor = Random.Range(0,3);

if(winningDoor == playerDoor) {
 print("Win");
} else {
  print("Lose");
}
```

Generate two random numbers, one for the winning door and one for the door that the player picks, labeled 0, 1, and 2.

If the player selects the winning door, we print "Win", otherwise we print "Lose"

We'll add in switching in a moment

The Big Deal

But this is only one run of the game, how can we "see" that we have a 1/3 chance to win and a 2/3 chance to lose?

```
int winningDoor = Random.Range(0,4);
int playerDoor = Random.Range(0,4);

if(winningDoor == playerDoor) {
 print("Win");
} else {
  print("Lose");
}
```

Monte Carlo Simulation

Run the game many times, counting the wins and loses, and compute the probabilities at the end.

The Big Deal

But this is only one run of the game, how can we "see" that we have a 1/3 chance to win and a 2/3 chance to lose?

Monte Carlo Simulation

Run the game many times, counting the wins and loses, and compute the probabilities at the end.

```
int runs = 1000;
int winCount = 0;
int loseCount = 0;

for(int i=0;i<runs;i++){
  int winningDoor = Random.Range(0,4);
  int playerDoor = Random.Range(0,4);

  if(winningDoor == playerDoor) {
    winCount++;
  } else {
    loseCount++;
  }
}
```

The Big Deal

```
int runs = 1000;
int winCount = 0;
int loseCount = 0;

for(int i=0;i<runs;i++){
  int winningDoor = Random.Range(0,4);
  int playerDoor = Random.Range(0,4);

  if(winningDoor == playerDoor) {
    winCount++;
  } else {
    loseCount++;
  }
}
print("Win: "+(float)winCount/runs);
print("Lose: "+(float)loseCount/runs);
```

What's the Probability?

The number of wins or loses divided by the total number of runs that we did.

Revealing

```
int runs = 1000;
int winStay = 0;
int winSwitch = 0;

for(int i=0;i<runs;i++){
  int winningDoor = rand.nextInt(3);
  int playerDoor = rand.nextInt(3);

  //reveal door
  int revealDoor = 0
  if(revealDoor==playerDoor || revealDoor==winningDoor)
{
    revealDoor = 1;
  }
  if(revealDoor==playerDoor || revealDoor==winningDoor)
{
    revealDoor = 2;
  };


  if(winningDoor == playerDoor){
    winStay++;
  }
```

Let's change up the code slightly, we are really only interested in how often we win if we switch vs if we stay.

There are many ways to do this, we'll take a simple approach:

Pick a door, see if it works, then try another door.

Switching

```
for(int i=0;i<runs;i++){
  int winningDoor = Random.Range(0,4);
  int playerDoor = Random.Range(0,4);

  //reveal door
  int revealDoor = 0
  if(revealDoor==playerDoor || revealDoor==winningDoor) {
    revealDoor = 1;
  }
  if(revealDoor==playerDoo
    revealDoor = 2;
  }

  //Switch door
  int switchDoor = 0
  if(switchDoor==playerDoor || switchDoor==revealDoor) {
    switchDoor = 1;
  }
  if(switchDoor==playerDoor || switchDoor==revealDoor) {
    switchDoor = 2;
  }
```

For switching, we'll do the same thing, except that we can only switch to a door that hasn't been revealed and isn't currently chosen.

Lastly

```
for(int i=0;i<runs;i++){
  ...

  if(winningDoor == playerDoor){
    winStay++;
  }
  if(winningDoor == switchDoor){
    winSwitch++;
  }
}
print("Win Stay: "+(float)winStay/runs);
print("Win Switch: "+(float)winSwitch/runs);
```

Now we just need to keep track of the times when we would have won if we stayed with our original choice of if it was better to switch.

Win Stay: 0.33
Win Switch: 0.67

Our new insight:

We win twice as often if we switch.

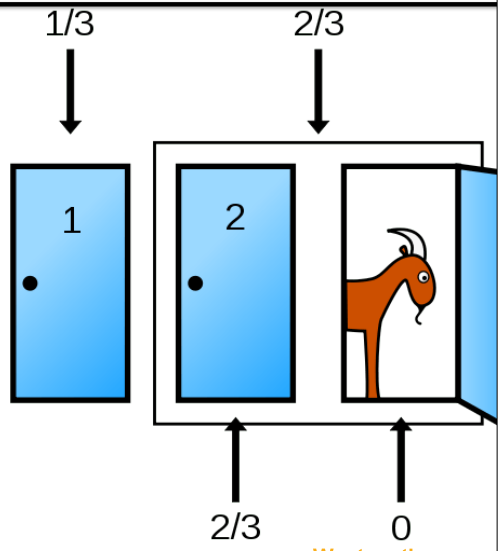
Always Switch!

Why Does it Work This Way?

If we pick door 1, we have 1/3 chance to win, with 2/3 chance that the winning door is 2 or 3.

When door 3 is revealed, we know it was not a winner, but the probability of those two door stay the same.

So, door 2 has 2/3 chance to be a winner.



Wentworth Computing & Data Science

Unity Important Classes, Short List

- https://docs.unity3d.com/Manual/ScriptingImportantClasses.html
- **`MonoBehavior`**
  - Contains functions and events for controlling your gameobjects.
- **`Transform`**
  - Contains the gameobject's position, rotation, and scale.
- **`Rigidbody`** and **`Rigidbody2D`**
  - Allows the gameobject to interact with the Unity physics engine.

There are many useful classes in Unity, but for now, these are the most common.

Wentworth Computing & Data Science

Unity Vectors

- https://docs.unity3d.com/Manual/VectorCookbook.html
- **`Vector4/Vector3/Vector2/`**
  - Unity's representation of vectors and points.

```
                                  //x      y      z
Vector3 pos = new Vector3(1.0f, 3.0f, 5.0f);
```

You'll use vectors or vector-like objects to describe most of the positional, rotational, and scale information about your gameobjects.

Linear Algebra plays a big part in any game. Much of it is behind the scenes, but many times, you will need to do your own vector/ matrix operations.

Wentworth Computing & Data Science

Vector Operations

- There are two main ways to think of **`Vector`** objects.
  - As points in space.
  - As "arrows" with magnitude and direction.

As Points

```
  Vector3 start = new Vector3(1.0f, 3.0f, 5.0f);
  Vector3 end = new Vector3(0.0f, 5.0f, 0.0f);

//Addition
  Vector3 res1 = start + end; //(1.0, 8.0, 5.0)

//Subtraction
  Vector3 res2 = start - end; //(1.0, -2.0, 5.0)

//Scalar Multiplication
  Vector3 res3 = 3*start; //(3.0, 9.0, 15.0)

//Scalar Division
  Vector3 res4 = start/2; //(0.5, 1.5, 2.5)
```

Wentworth Computing & Data Science

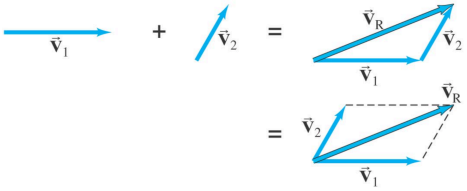
Vector Operations

Vectors As "Arrows"

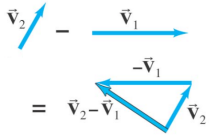
```
  Vector3 v1 = new Vector3(1.0f, 3.0f, 5.0f);
  Vector3 v2 = new Vector3(0.0f, 5.0f, 0.0f);

//Addition
  Vector3 res1 = v1 + v2; //(1.0, 8.0, 5.0)

//Subtraction
  Vector3 res2 = v1 - v2; //(1.0, -2.0, 5.0)
```

It can be very useful to draw out the arrows on paper to help understand what is happening.

Addition Picture



Subtraction Picture



Vector Operations

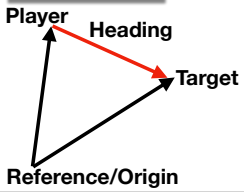
Direction and Distance

```
  Vector3 target = new Vector3(1.0f, 3.0f, 5.0f);
  Vector3 player = new Vector3(0.0f, 5.0f, 0.0f);

//Compute Heading
  Vector3 heading = target - player; //(1.0, -2.0, 5.0)

//Compute Distance
  float dist = heading.magnitute; //5.47

//Compute Direction
  Vector3 dir = heading/dist; //normalized vector
                              //length 1 vector
```

Common problem:

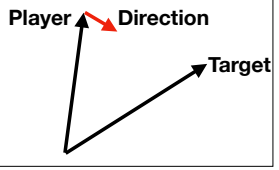
How far is the target from the player and in what direction?

Heading



Distance

Direction

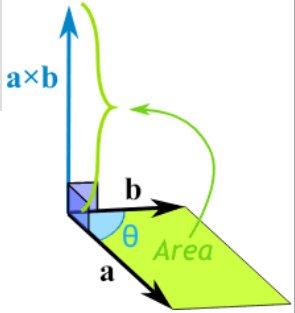


Vector Operations

Normal Vector

```
  Vector3 a = new Vector3(1.0f, 3.0f, 5.0f);
  Vector3 b = new Vector3(0.0f, 5.0f, 0.0f);

//Perpendicular Vector
  Vector3 perp = Vector3.Cross(a, b);

//Perpendicular Vector Length
  float perpLength = perp.magnitude;

//Perpendicular Unit Vector (length 1)
  Vector3 perp = perp/perpLength;
```

Normal vectors are perpendicular to a plane.



Normal vectors are often used in computer graphics (e.g. dynamic mesh generation) and in 3D path finding.

Vector Operations

The point of these vector operations isn't for you to remember all the math and use cases.

It's to show you some of the built in capabilities of Unity, there's no need to reinvent the wheel.

It also shows that understanding the math and interpretation behind these concepts is just as important as remembering the syntax.

Unity has many useful classes that you'll see in action throughout this class.

Back to Simple Scripts

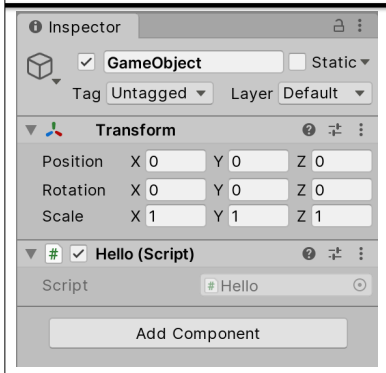
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Hello : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

Create a new script in Unity (Right Click in Project->Create->C# Script)

Create a new empty object in the Hierarchy. (Right Click->Create Empty

Drag the script from the Project view to the empty you just created.

Simple Scripts



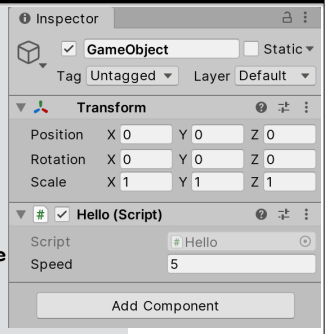
Notice that this gameobject (GameObject) has two components, **`Transform`** and **`Hello`** (my script).

Your scripts can access any component via code!

The other elements, Tag, Layer, etc., are adjustments that we can make to the gameobject itself, we'll talk about them in the future.

Simple Scripts

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

public class Hello : MonoBehaviour
{
    public float speed = 5f;
    private float health = 10f;

    // Start is called before the first frame
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }
}
```



Any public variable in a MonoBehavior (attached to an object) shows up in the inspector.

Simple Scripts

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

public class Hello : MonoBehaviour
{
    public float speed = 5f;
    private float health = 10f;

    // Start is called before the first frame update
    void Start()
    {
        Vector3 pos = this.transform.position;
        Debug.Log(pos);  //prints (0.0, 0.0, 0.0)
    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

You can access the transform (and thus the position, rotation, and scale) directly using **`this.transform`**

Simple Scripts

Add a Rigidbody component to this object.

Other components need to be accessed via the **`GetComponent()`** call.

```
Type Name = GetComponent<Type>();
```

With this reference, you can access any element of the component via code.

Simple Scripts

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

public class Hello : MonoBehaviour
{
    public float speed = 5f;
    private float health = 10f;
    private Rigidbody rb;

    // Start is called before the first frame update
    void Start()
    {
        Vector3 pos = this.transform.position;
        Debug.Log(pos);  //prints (0.0, 0.0, 0.0)
        rb = GetComponent<RigidBody>();
    }

    // Update is called once per frame
    void Update()
    {

    }
```

If the component doesn't exist, you will get an error on the **`GetComponent()`** line

There are workarounds for this that we'll discuss later.

Simple Scripts

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

public class Data
{

}
```

You are not restricted to creating **`MonoBehaviour`**s,

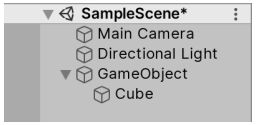
Feel free to create any regular classes, however, they cannot be added as a component to a gameobject.

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

public class Hello : MonoBehaviour
{
...
    Data d = new Data();
...
}
```

You can access any regular classes without doing anything special.

Movement Example

- Add a Cube to the GameObject that we created. Right Click in the Hierarchy->3D Object->Cube and drag it onto the GameObject.




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

public class Hello : MonoBeha
{

  Update()
  {

  }

}
```

Clean out the script that we've been working on, we'll only need **`Update()`** for now.

Also, remove the **`Rigidbody`** from the object.

Movement Example

Clean out the script that we've been working on, we'll only need **`Update()`** for now.

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

public class Hello : MonoBehaviour
{

  public float speed = 10f;

  Update()
  {
    Vector3 pos = this.transform.position;
    if(Input.GetKey(KeyCode.UpArrow)){
      pos.y += speed * Time.deltaTime;
    }
    this.transform.position = pos;
  }
}
```

Grab the current position as **`pos`**

If the Up Arrow key is pressed...

...change **`pos.y`**

Update the **`transform.position`**

Movement Example

```
...
    Vector3 pos = this.transform.position;
    if(Input.GetKey(KeyCode.UpArrow)){
      pos.y += speed * Time.deltaTime;
    }
...
```

How is this updating the y position?

`speed` is a constant that we set

`pos.y` is being increased

What is **`Time.deltaTime`**?

`Time.deltaTime` is the amount of time that has passed between this frame and the previous frame.

For a 60fps game, this time is about 16ms

Movement Example

```
...
    Vector3 pos = this.transform.position;
    if(Input.GetKey(KeyCode.UpArrow)){
      pos.y += speed;
    }
...
```

What if we didn't have it? How does our movement behave?

Physics Lesson

$$\frac{dx}{dt} = v$$

Velocity is change in distance over change in time

$$\frac{x(t+dt) - x(t)}{dt} = v$$

Using definition of the derivative

$$x(t+dt) = v \cdot dt + x(t)$$

Rearrange

Movement Example

$$x(t+dt) = v \cdot dt + x(t)$$

Position Now

Position **`dt`** time from now

`dt` (small time)

`speed` (Velocity)

Look Familiar?

Unity has its own physics system (based on **`Rigidbody`**) that we'll use in the future, but at a fundamental level, it's based on numerically solving differential equations.

So, what is **`Time.deltaTime`**?

It's **`dt`** in our equation.

Which means we're using real physics to move our object.

Movement Example

```
...
  Update()
  {
    Vector3 pos = this.transform.position;
    if(Input.GetKey(KeyCode.UpArrow)){
      pos.y += speed * Time.deltaTime;
    }
    if(Input.GetKey(KeyCode.DownArrow)){
      pos.y -= speed * Time.deltaTime;
    }
    if(Input.GetKey(KeyCode.RightArrow)){
      pos.x += speed * Time.deltaTime;
    }
    if(Input.GetKey(KeyCode.LeftArrow)){
      pos.x -= speed * Time.deltaTime;
    }
    this.transform.position = pos;
  }

}
```

Add the rest of the directions, pay attention to the axis (**x** or **y**) and the sign (**-=** or **+=**).

Keep in mind:

This is moving the GameObject, not the cube.

But, since the cube is a child (in the Hierarchy) of the GameObject, it moves too.

Next Time:

There are a myriad of functions, types, and settings for interacting with GameObjects. I'll introduce them as needed when we are working on our games.