INFT3960 – Game Production

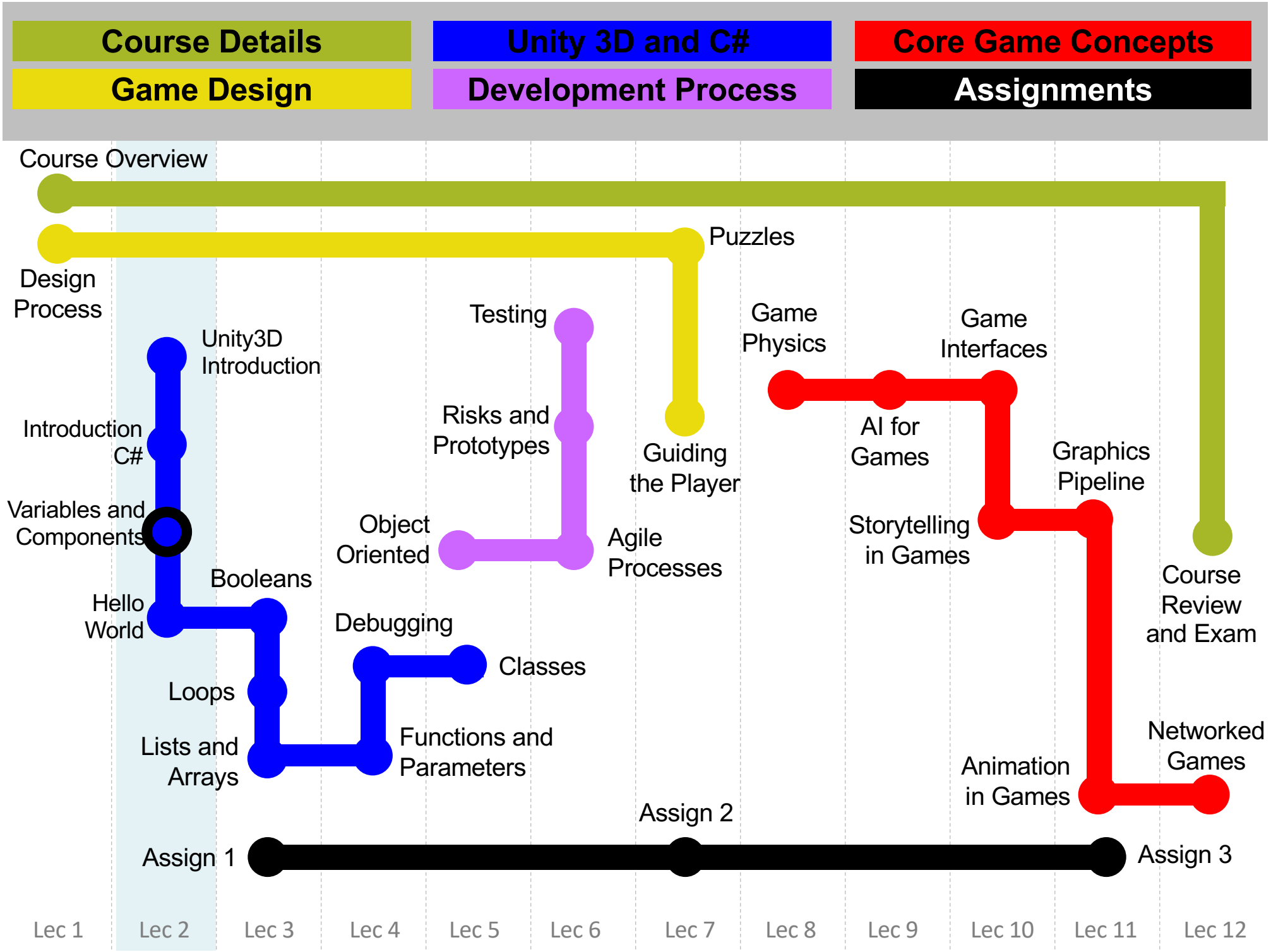
Week 02

Module 2.3

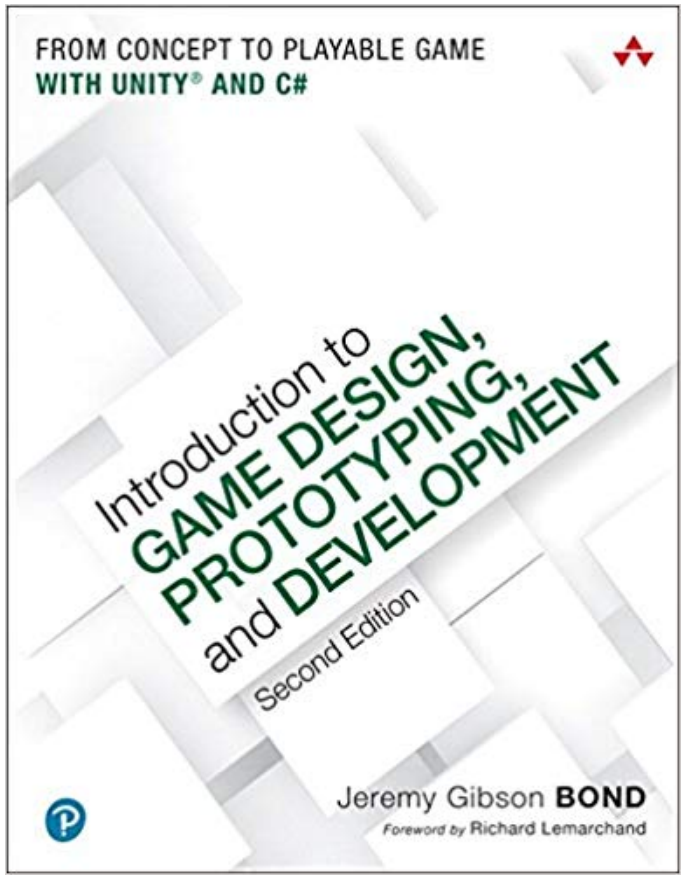
Variables and Components

Course Overview - 2019

Lec	Date	Modules	Assignments
1	Tuesday 30 Jul	Mod 1.1, Mod 1.2	
2	Tuesday 5 Aug	Mod 2.1, Mod 2.2, Mod 2.3, Mod 2.4	
3	Tuesday 12 Aug	Mod 3.1, Mod 3.2, Mod 3.3	 Assign 1 12 Aug, 11:00 pm
4	Tuesday 19 Aug	Mod 4.1, Mod 4.2	
5	Tuesday 26 Aug	Mod 5.1, Mod 5.2	
6	Tuesday 3 Sep	Mod 6.1, Mod 6.2, Mod 6.3	
7	Tuesday 10 Sep	Mod 7.1, Mod 7.2	 Assign 2 12 Sep, 11:00 pm
8	Tuesday 17 Sep	Mod 8.1	
9	Tuesday 24 Sep	Mod 9.1	
10	Tuesday 15 Oct	Mod 10.1, Mod 10.2	
11	Tuesday 22 Oct	Mod 11.1, Mod 11.2	 Assign 3 24 Oct, 11:00pm
12	Tuesday 29 Oct	Mod 12.1, Mod 12.2	


Course Details
Unity 3D and C#
Core Game Concepts
Game Design
Development Process
Assignments
Course Overview
Design Process
Puzzles
Unity3D Introduction
Introduction C#
Variables and Components
Hello World
Booleans
Loops
Lists and Arrays
Debugging
Classes
Functions and Parameters
Testing
Risks and Prototypes
Object Oriented
Agile Processes
Guiding the Player
Game Physics
AI for Games
Storytelling in Games
Game Interfaces
Graphics Pipeline
Animation in Games
Networked Games
Course Review and Exam
Assign 1
Assign 2
Assign 3
Lec 1
Lec 2
Lec 3
Lec 4
Lec 5
Lec 6
Lec 7
Lec 8
Lec 9
Lec 10
Lec 11
Lec 12

More C# – (Chapter 20)



VARIABLES AND COMPONENTS

Variables and Components - Topics

- Declaring and defining variables
- Important C# Variable Types
- Naming Conventions
- Important Unity Variable Types
- Unity GameObject Components

Variables

A variable is a named container for data

Variables in C# are typed, so they can only hold one type of data (e.g., an integer, a float, a string)

Variables need to be declared to be used

```
int x;
```

Assigning a value to a variable is called defining the variable (or initialising)

```
x = 5;
```

Remember Variables

A literal is a value that is entered into your code and can be assigned to a variable

```
x = 5;
```

The 5 above is an integer literal

string literals are surrounded by double quotes:

```
message = "Hello World!"
```

float literals are followed by an f:

```
myPi = 3.14f
```

(otherwise it will be a double by default)

Important C# Variable Types

Core C# variable types start with a lowercase character

- bool
- int
- float
- char
- string
- class

Variable Types - bool

bool – 1-bit True or False Value

Short for Boolean - Named after George Boole (an English mathematician)

bools in C# actually use more than 1-bit of space

- The smallest addressable memory chunk on a 32-bit system is 32 bits.
- The smallest on a 64-bit system is 64 bits.

Literal examples: `true` `false`

```
bool verified = true;
```

Variable Types - int

int – 32-bit integer

Stores a single integer number - Integers are numbers with no fractional or decimal element

int math is very fast and accurate - an store numbers between

–2,147,483,648 and 2,147,483,647

31 bits used for number and 1 bit used for sign

Literal examples: `1` `34567` `-48198`

```
int nonFractionalNumber = 12345;
```

Variable Types - float

float – 32-bit decimal number

Stores a floating-point number with a decimal element

A floating-point number is stored in something like scientific notation

Scientific notation is numbers in the format
$a*10^b$: For example, 300 is $3*10^2$

Floating-point numbers are stored in the format $a*2^b$

- 23 bits are used for the significand (the a part)
- 8 bits are used for the exponent (the b part)
- 1 bit determines whether the number is positive or negative

Variable Types - float

float – 32-bit decimal number

Floats are inaccurate for large numbers and for numbers between -1 and 1

e.g. There is no accurate float representation for $\frac{1}{3}$

Literal examples: `3.14f` `123f` `123.456f`

```
float notPreciselyOneThird = 1.0f / 3.0f;
```

Variable Types - char

char – 16-bit character

Single character represented by 16 bits of information

Uses Unicode values for the characters

- Unicode represents 110,000 different characters from over 100 different character sets and languages

Uppercase and lowercase letters are different values!

char literals are surrounded by single quotes

Literal examples: `'A'` `'a'` `'\t'`

```
char theLetterA = 'A';
```

Variable Types - string

string – a series of 16 bit characters

Stores from no characters ("") to an entire novel

Max length is 2 billion chars; ~12,000 times the length of Hamlet

string literals are surrounded by double quotes

Literal examples: `"Hello"` `""` `"\tTab"`

```
string firstLineOfHamlet = "Who's there?";
```

Variable Types - string

string – a series of 16-bit characters

You can access individual characters via bracket access

```
char theCharW = theFirstLineOfHamlet[0];

char questionMark = theFirstLineOfHamlet[11];
```

The length of a string is accessed via .Length

```
int len = firstLineOfHamlet.Length;
```

Sets len to 12

Variable Types - class

class – a collection of Functions and Data

A class creates a new variable type

Covered extensively in Chapter, "Classes"

Already used in the HelloWorld project (chapter 19)

```
public class HelloWorld : MonoBehaviour {
        void Start() {
            print("Hello World!");
        }
    }
```

Everything between the braces { } is part of the class

Naming Conventions

Use camelCase for almost everything

Variable names start with lowercase:

```
thisVariable    anotherVariable    bob
```

Function names start with uppercase:

```
ThatFunction()    Start()    Update()
```

Class names start with uppercase:

```
SomeClass    GameObject    HeroShip
```

Private variables start with underscore:

```
_privateVariable    _hiddenVariable
```

Static variables use SNAKE_CASE:

```
STATIC_VAR    NUM_INSTANCES
```

Important Unity Types

Because they are classes, important Unity variable types all start with an uppercase character

- Vector3
- Color
- Quaternion
- Mathf
- Screen
- SystemInfo
- GameObject

Vector3

Used for position of objects in 3D

```
Vector3 vec = new Vector3( 3, 4, 0 );
```

Instance variables and functions

- vec.x – The x component of the vector
- vec.y – The y component of the vector
- vec.z – The z component of the vector
- vec.magnitude – The length of the vector
- vec.Normalize() – New Vector3 in the same direction at unit length

Static class variables and functions

- Vector3.zero – Shorthand for new Vector3(0, 0, 0);
- Vector3.Dot(vA, vB); – Dot product of vA and vB

Color

A color with transparency information- 4 floats, one for red, green, blue, and alpha (all between 0 and 1)

```
Color col = new Color( 0.5f, 0.5f, 0, 1f );
Color col = new Color( 1f, 0f, 0f );          // Alpha is optional
```

In the Unity color picker, the RGBA values are in the range 0–255. These are then mapped to 0–1f.

Instance variables and functions

col.r – The red component of the vector
col.g – The green component of the vector
col.b – The blue component of the vector
col.a – The alpha component of the vector

Color – Static

Static class variables and functions

```
// Primary Colors: Red, Green, and Blue
Color.red     = new Color(1, 0, 0, 1);    // Solid red
Color.green = new Color(0, 1, 0, 1);    // Solid green
Color.blue   = new Color(0, 0, 1, 1);     // Solid blue

// Secondary Colors: Cyan, Magenta, and Yellow
Color.cyan    = new Color(0, 1, 1, 1);              // Cyan, a bright greenish blue
Color.magenta = new Color(1, 0, 1, 1);              // Magenta, a pinkish purple
Color.yellow  = new Color(1, 0.92f, 0.016f, 1); // A nice-looking yellow

// As you can imagine, a standard yellow would be new Color(1,1,0,1), but
//  in Unity's opinion, this color looks better.
```

Color – Static

Static class variables and functions

```
// Black, White, and Clear
Color.black   = new Color(0, 0, 0, 1);           // Solid black
Color.white   = new Color(1, 1, 1, 1);           // Solid white
Color.gray    = new Color(0.5f, 0.5f, 0.5f, 1)   // Gray
Color.grey    = new Color(0.5f, 0.5f, 0.5f, 1)   // British spelling of gray
Color.clear   = new Color(0, 0, 0, 0);           // Completely transparent
```

Quaternion

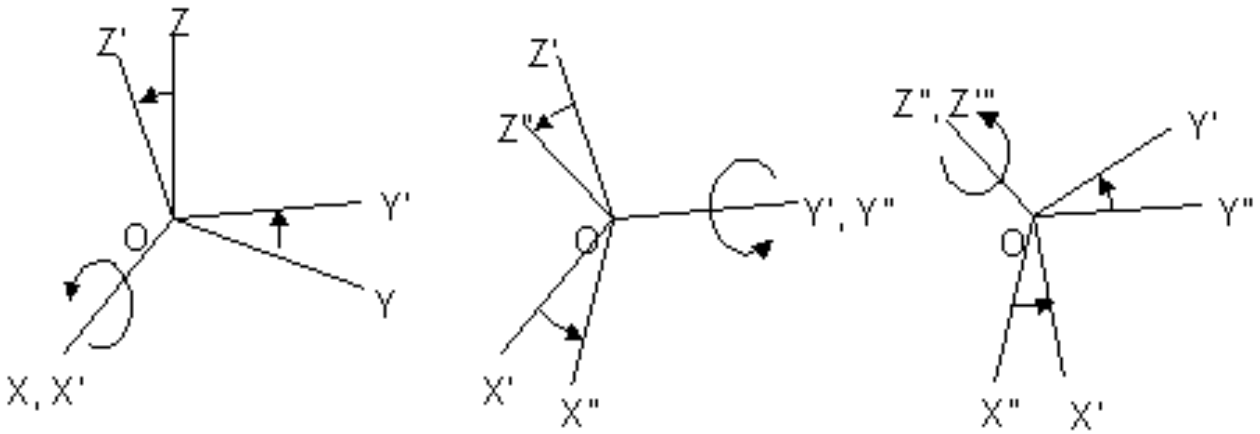
Quaternion – **Rotation information**

Based on three imaginary numbers and a scalar

So, everyone uses Euler angles (e.g., x, y, z) to input rotation

```
Quaternion up45Deg = Quaternion.Euler(-45,0,0);
```

In Euler (pronounced "oiler") angles, x, y, & z are rotations about those respective axes



Quaternion

Quaternion – **Rotation information**

Quaternions are much better for interpolation and calculations than Euler angles

They also avoid Gimbal Lock (where two Euler axes align)

Instance variables and functions

`up45Deg.eulerAngles`

a Vector3 of the Euler rotations

Mathf

Mathf – **A collection of static math functions**

Static class variables and functions

```
Mathf.Sin(x);    // Computes the sine of x
Mathf.Cos(x);    // .Tan(), .Asin(), .Acos(), & .Atan() also available

Mathf.Atan2( y, x );  // Gives you the angle to rotate around the z-axis to
                      //  change something facing along the x-axis to face
                      //  instead toward the point x, y.

print(Mathf.PI);    // 3.141593; the ratio of circumference to diameter
```

Mathf

Mathf – **A collection of static math functions**

Static class variables and functions

```
Mathf.Min( 2, 3, 1 );    // 1, the smallest of the numbers (float or int)
Mathf.Max( 2, 3, 1 );    // 3, the largest of the numbers (float or int)
Mathf.Round( 1.75f );    // 2, rounds up or down to the nearest number
Mathf.Ceil( 1.75f );     // 2, rounds up to the next highest integer number
Mathf.Floor( 1.75f );    // 1, rounds down to the next lowest integer number
Mathf.Abs( -25 );        // 25, the absolute value of -25
```

Screen

Screen – **Information about the display**

Static class variables and functions

```
Screen.width     // The width of the screen in pixels
Screen.height    // The height of the screen in pixels

Screen.showCursor = false;  // Hide the cursor – ver 4 Unity
Cursor.visible = false;     // Hide the cursor – ver 5 Unity
```

SystemInfo

SystemInfo – **Information about the device/computer**

Static class variables and functions

```
SystemInfo.operatingSystem  // The width of the screen in pixels
                            //   e.g., Mac OS X 10.9.3

SystemInfo.systemMemorySize         // Amount of RAM

SystemInfo.supportsAccelerometer  // Has accelerometer

SystemInfo.supportsGyroscope        // Has gyroscope
```

GameObject

GameObject – **Base class for all objects in scenes**

Static class variables and functions

Composed of components

```
GameObject go = new GameObject("MyGO");
```

GetComponent<>() is a generic method that can be used to access any component attached to a GameObject

```
go.GetComponent<Transform>()      // The Transform component
```

Always has a Transform component

GameObject

GameObject – **Base class for all objects in scenes**

Static class variables and functions

Instance variables and functions

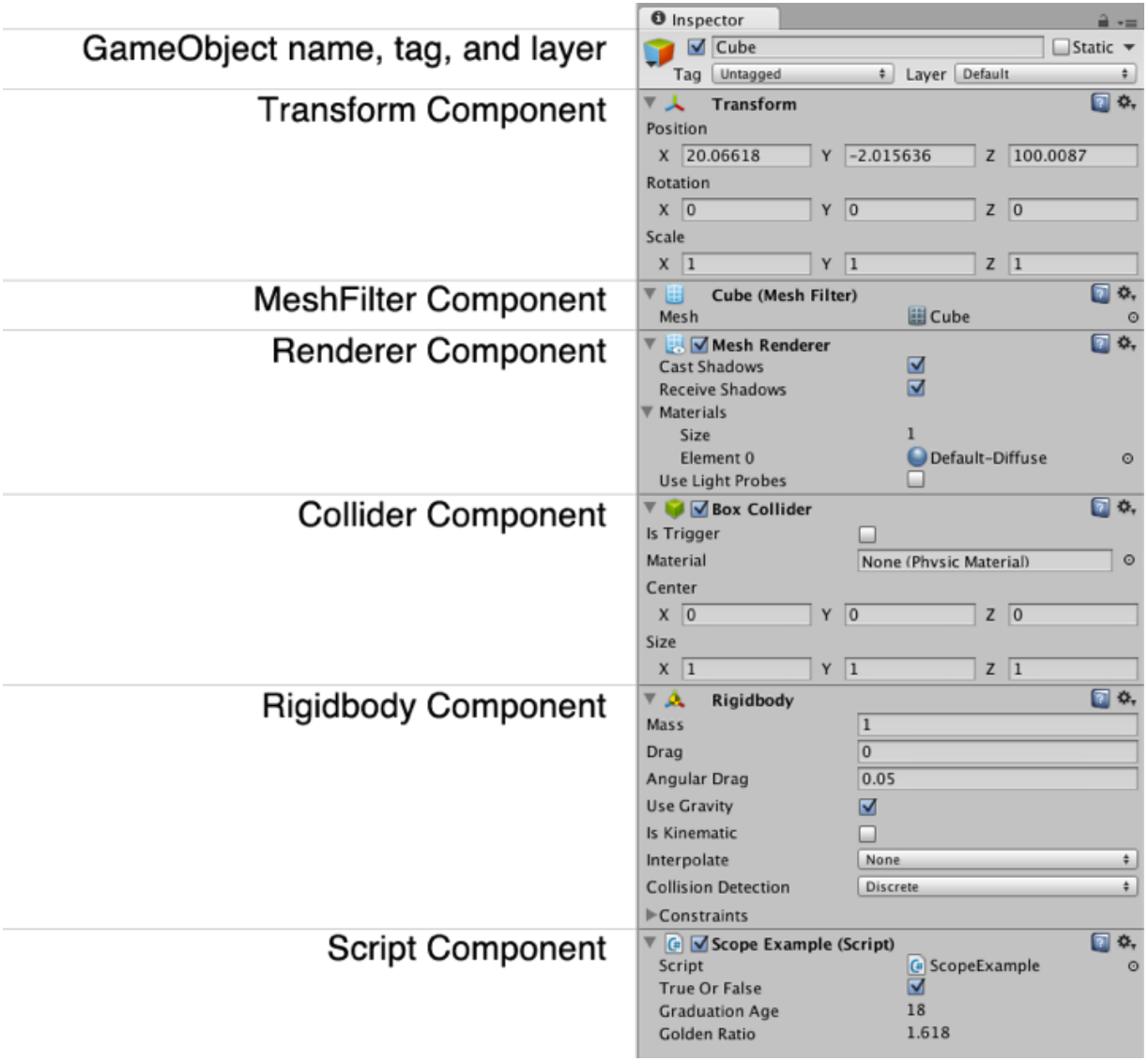
```
go.name    // The name of the GameObject ("MyGO")

go.GetComponent<Transform>()      // The Transform component

go.SetActive(false)          // Make this GameObject inactive
```

GameObject Components

GameObjects are composed of Components



Component - Transform

Controls position, rotation, and scale

```
Transform tr = go.GetComponent<Transform>();
```

Also controls hierarchy of objects in the scene

```
tr.parent // The parent of this transform in the hierarchy
```

Children can be iterated over with a foreach loop

```
foreach (Transform tChild in tr) {…}
```

Instance variables and functions

```
tr.position        // The position in world coordinates
tr.localPosition   // The position relative to its parent
tr.rotation        // The rotation in world coordinates
tr.localScale      // The scale (always in local coordinates)
```

Component - MeshFilter

The model that you see

```
MeshFilter mf = go.GetComponent<MeshFilter>();
```

Attaches a 3D model to a GameObject

Is actually a 3D shell of the object (3D objects in games are hollow inside

This MeshFilter is rendered on screen by a MeshRenderer component

Component - Renderer

Draws the GameObject on screen

```
Renderer rend = go.GetComponent<Renderer>();
```

Usually, this is a MeshRenderer

- Renderer is the superclass for MeshRenderer
- So, Renderer is almost always used in code

Combines the MeshFilter with a Material (which contains various Textures and a Shader)

Component - Collider

The physical presence of the GameObejct

```
Collider coll = go.GetComponent<Collider>();
```

There are four types of collider (in order of complexity)

- Sphere Collider – The fastest type. A ball or sphere.
- Capsule Collider – A pipe with spheres at each end. 2nd fastest.
- Box Collider – A rectangular solid. Useful for crates, cars, torsos, etc.
- Mesh Collider – Collider formed from a MeshFilter. Much slower!

  - Only convex Mesh Collider can collide with other Mesh Colliders
  - Much, much slower than the other three types

Unity physics are performed by the NVIDIA PhysX engine

Colliders will not move without a Rigidbody component

Component - RigidBody

The physical simulation of the GameObject

```
Rigidbody rigid = go.GetComponent<Rigidbody>();
```

Handles velocity, bounciness, friction, gravity, etc.

Updates every FixedUpdate()

- This is exactly 50 times per second

If Rigidbody isKinematic == true, the collider will move, but position will not change automatically due to velocity

```
rigid.isKinematic = true;   // rigid will not move on its own
```

Colliders will not move without a Rigidbody component

Component - Script

Any C# class that you write

```
HelloWorld hw = go.GetComponent<HelloWorld>();
```

Because C# scripts are handled as components, several can be attached to the same GameObject

- This enables more object-oriented programming
- You'll see several examples in the textbook etc.

Public fields in your scripts will appear as editable fields in the Unity Inspector - However, Unity will often alter the names of these fields a bit

- The class name ScopeExample becomes Scope Example (Script).
- The variable trueOrFalse becomes True Or False.
- The variable graduationAge becomes Graduation Age.
- The variable goldenRatio becomes Golden Ratio.

Summary

Learned about declaring and defining C# variables

Learned several important C# variable types

- These all start with lowercase letters

Learned naming conventions used in this book

Important Unity Variable Types

- These all start with uppercase letters

Learned several Unity GameObject components