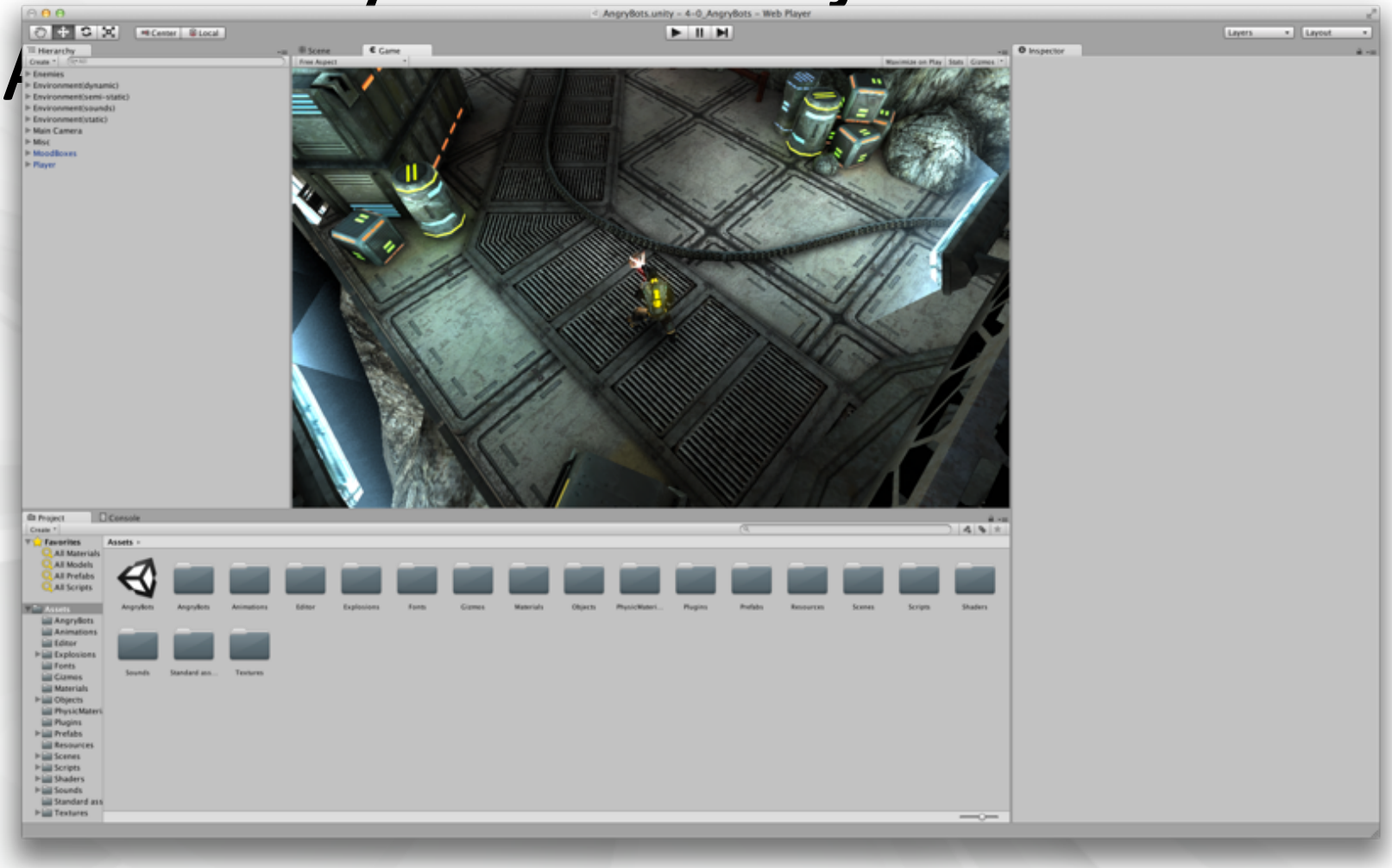
Getting started with Unity & Github

Gibson Ch 16-19

The Unity Demo Project:



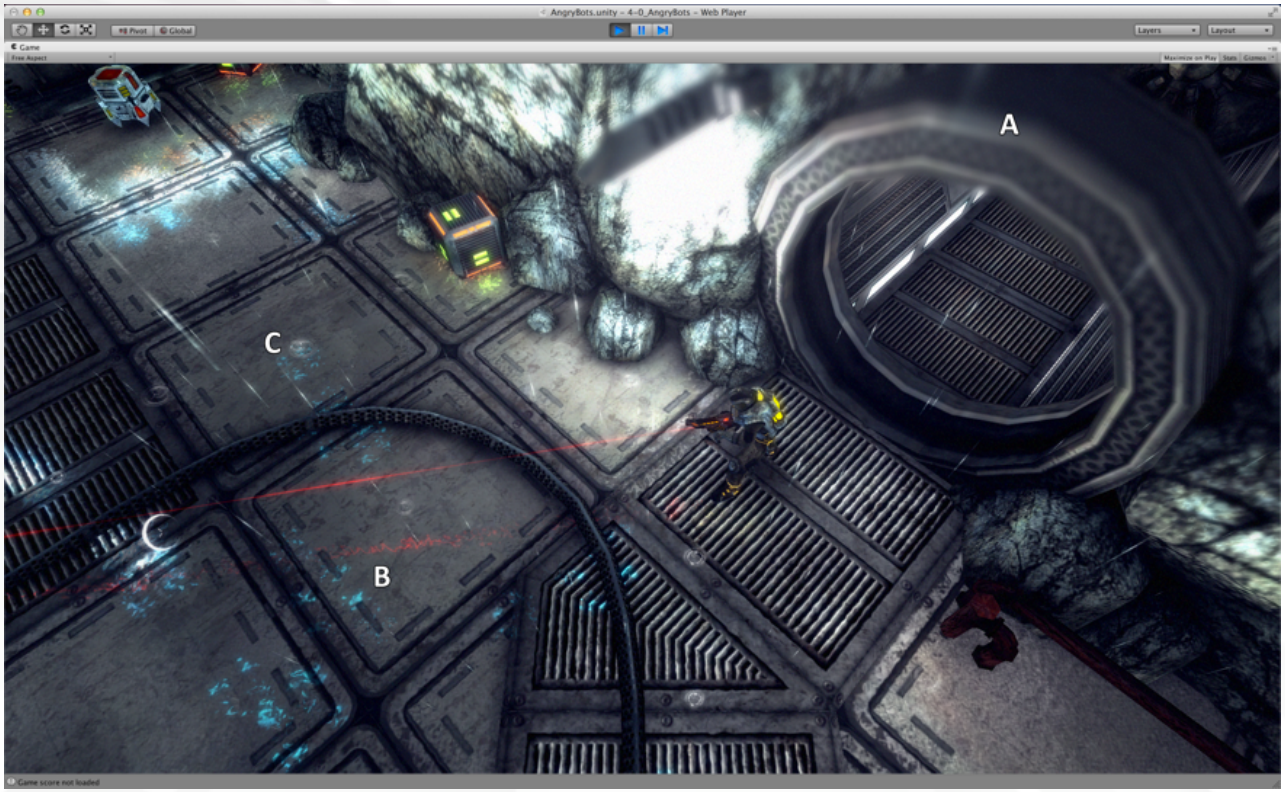
The Unity window when it opens for the first time

The Unity Demo Project: AngryBots

- AngryBots Controls
  - Movement is controlled by WASD or Arrow Keys
  - The gun aims at your mouse pointer
  - Press the left mouse button to fire
  - Environmental awareness
    - Standing next to a circular door will cause it to open
    - Standing next to some computers will change the wires leading from them from red to green and unlock doors

Unity Features Shown in AngryBots

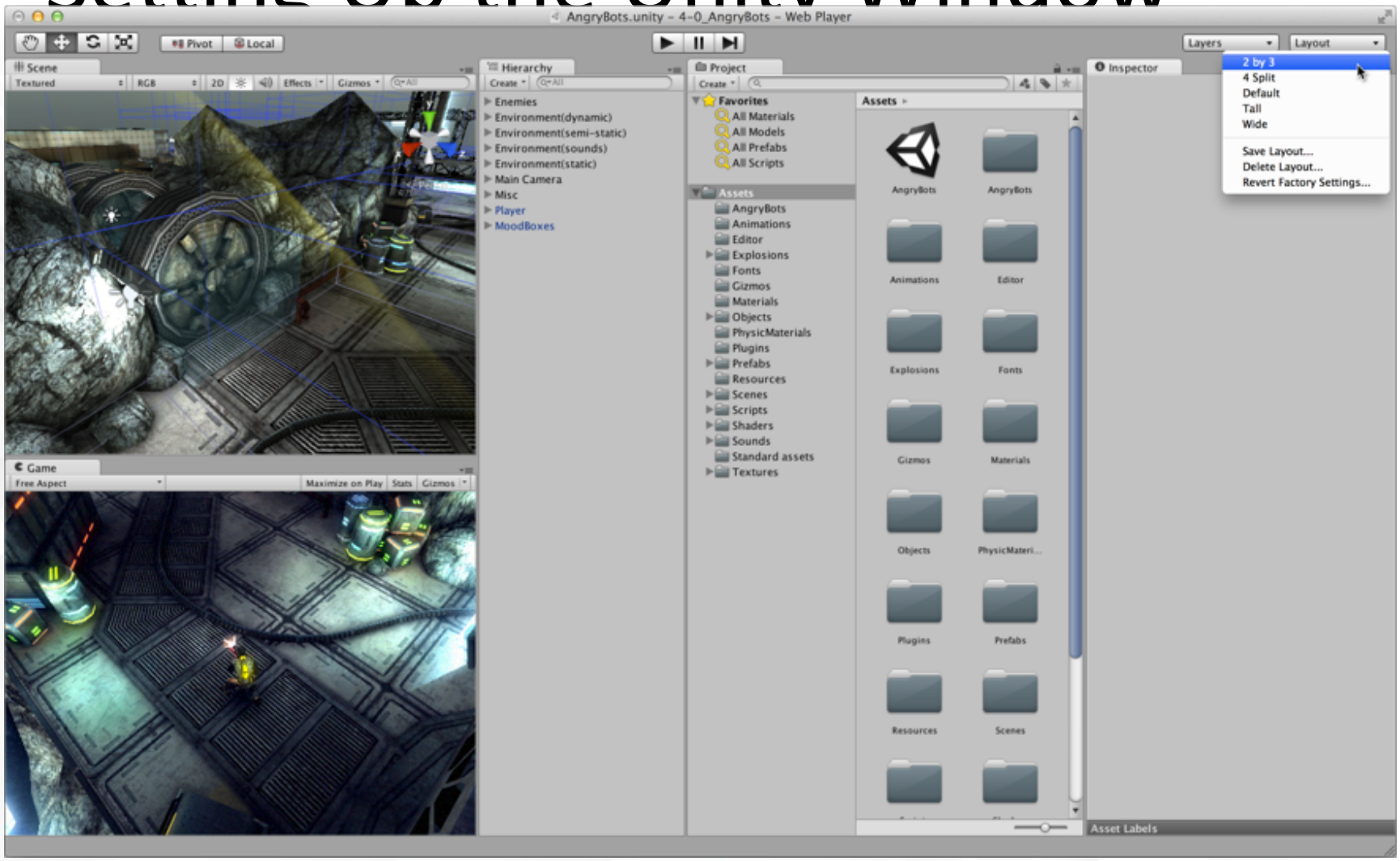
- Shaders
  - A: Depth of field shader (Unity Pro only)



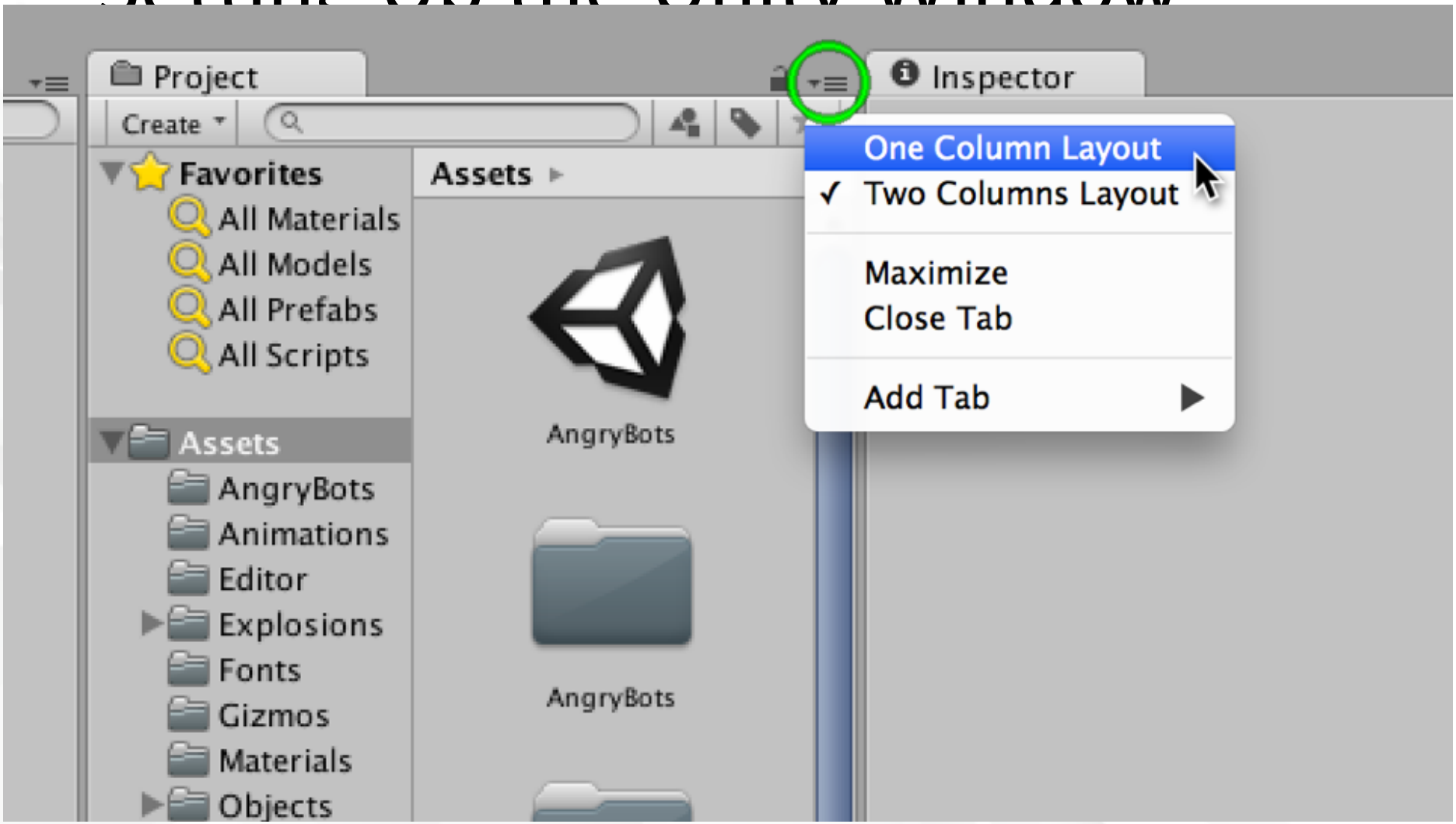
Unity Features Shown in AngryBots

- Shaders
  - A: Depth of field shader (Unity Pro only)
  - B: Reflections (Unity Pro only)
  - C: Animated texture (raindrops)
- Character rigging and animation
  - Animation blending allows the character to move in one direction while looking in another
- Artificial Intelligence-based Pathing
  - Enemies will move around objects in a room to track down the player

Setting Up the Unity Window

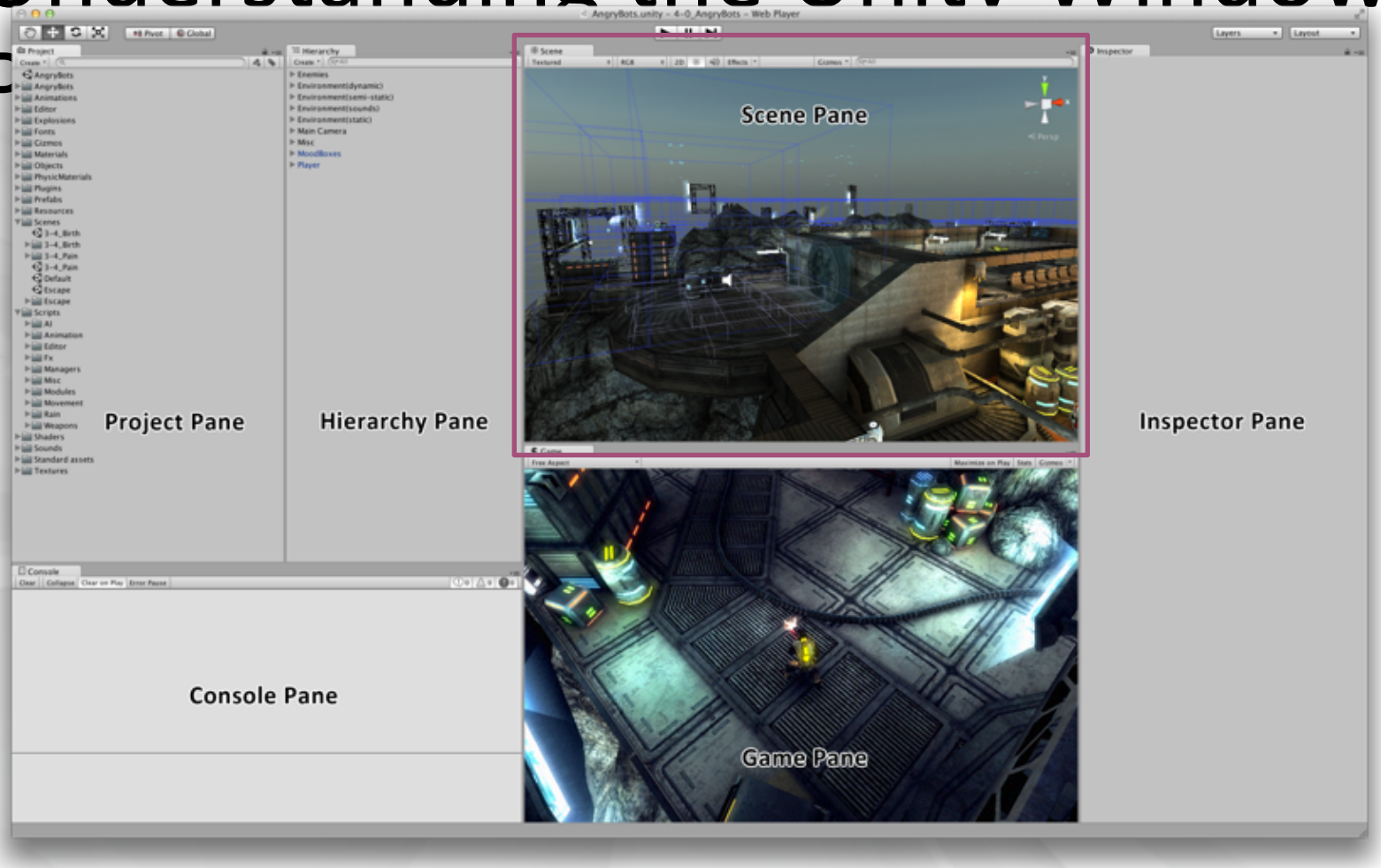


Setting Up the Unity Window



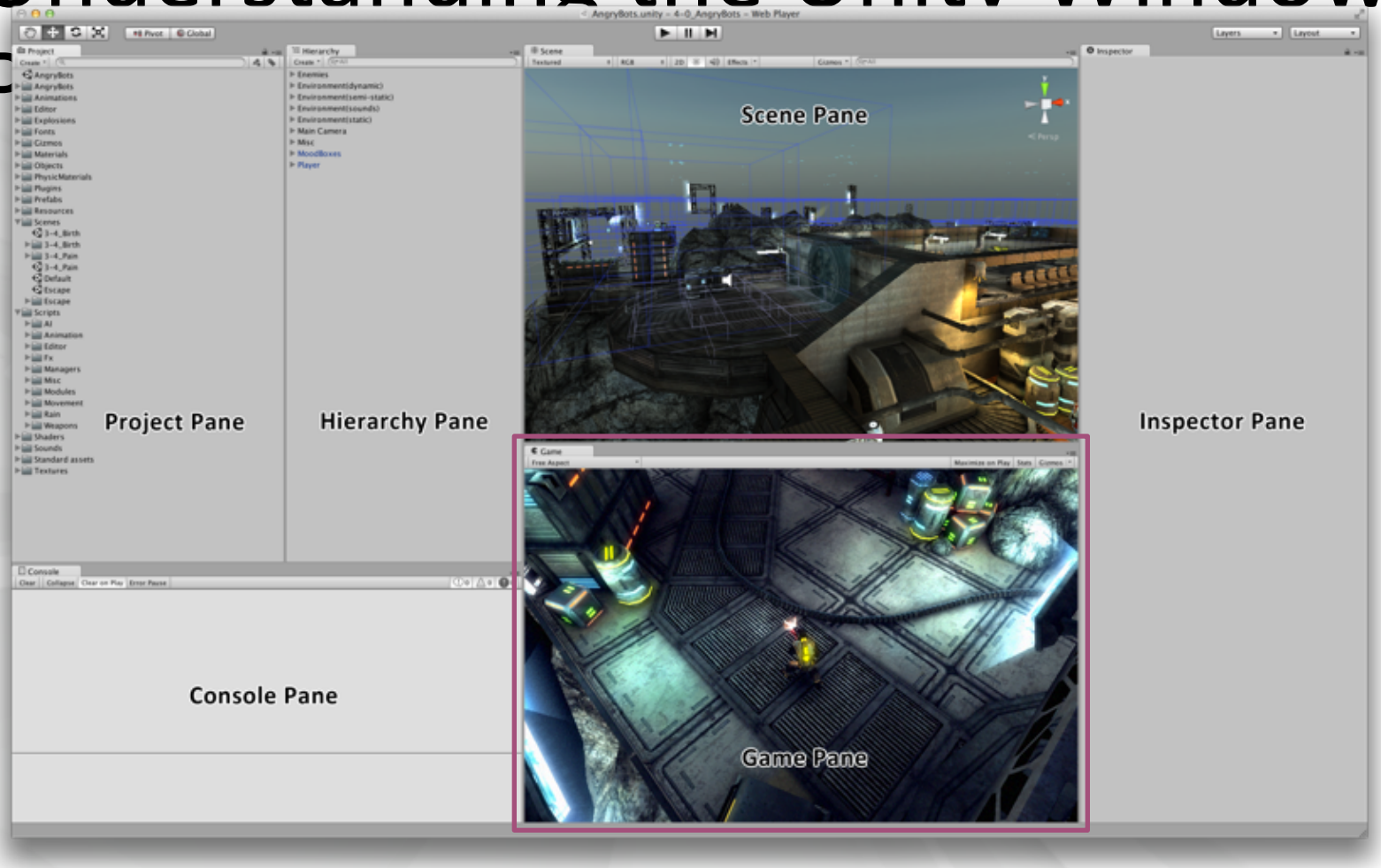
Set the Project pane to *One Column Layout*

Understanding the Unity Window P



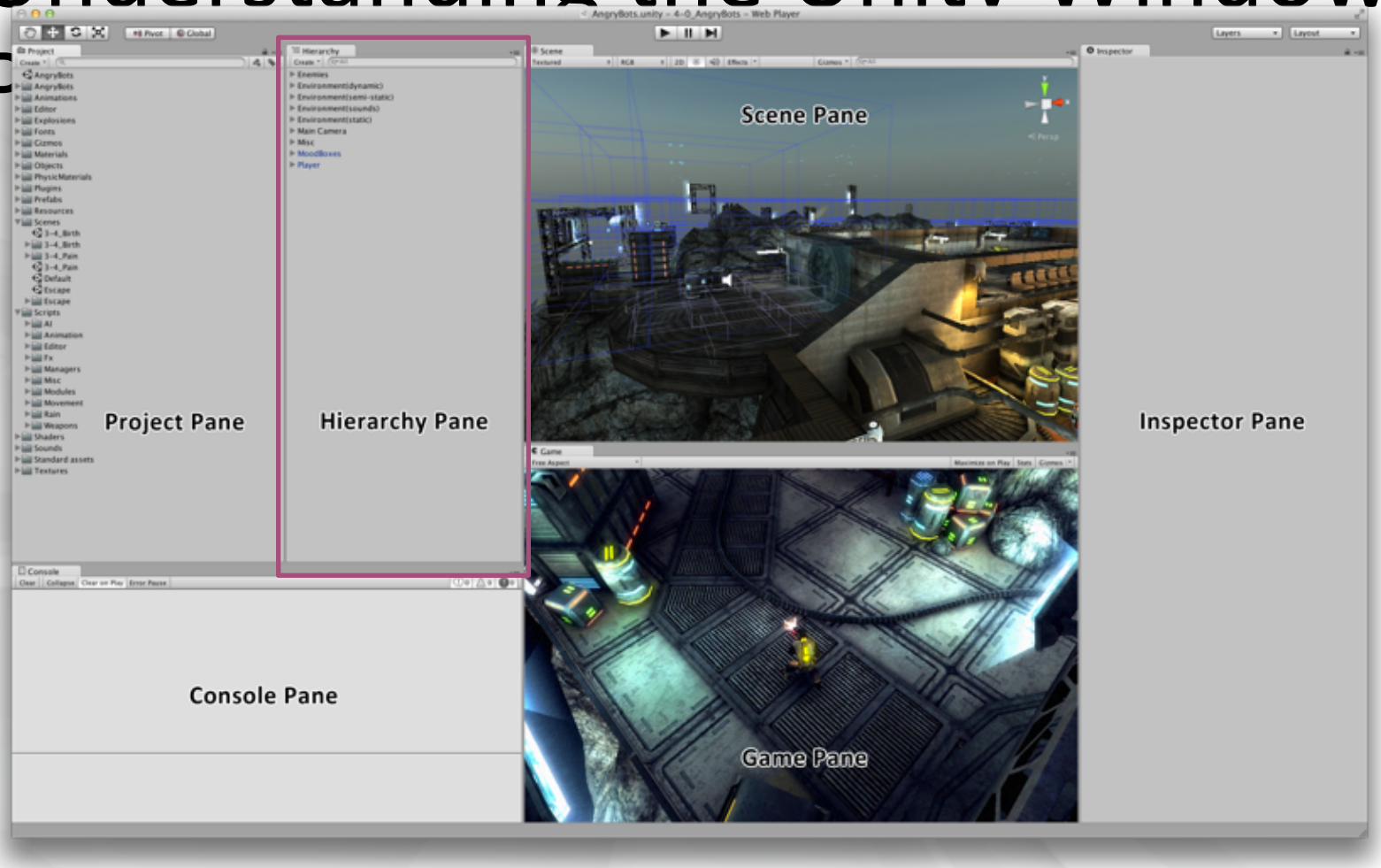
Scene Pane: **Allows you to move around the 3D scene and select, move, rotate, and scale GameObjects.**

Understanding the Unity Window P



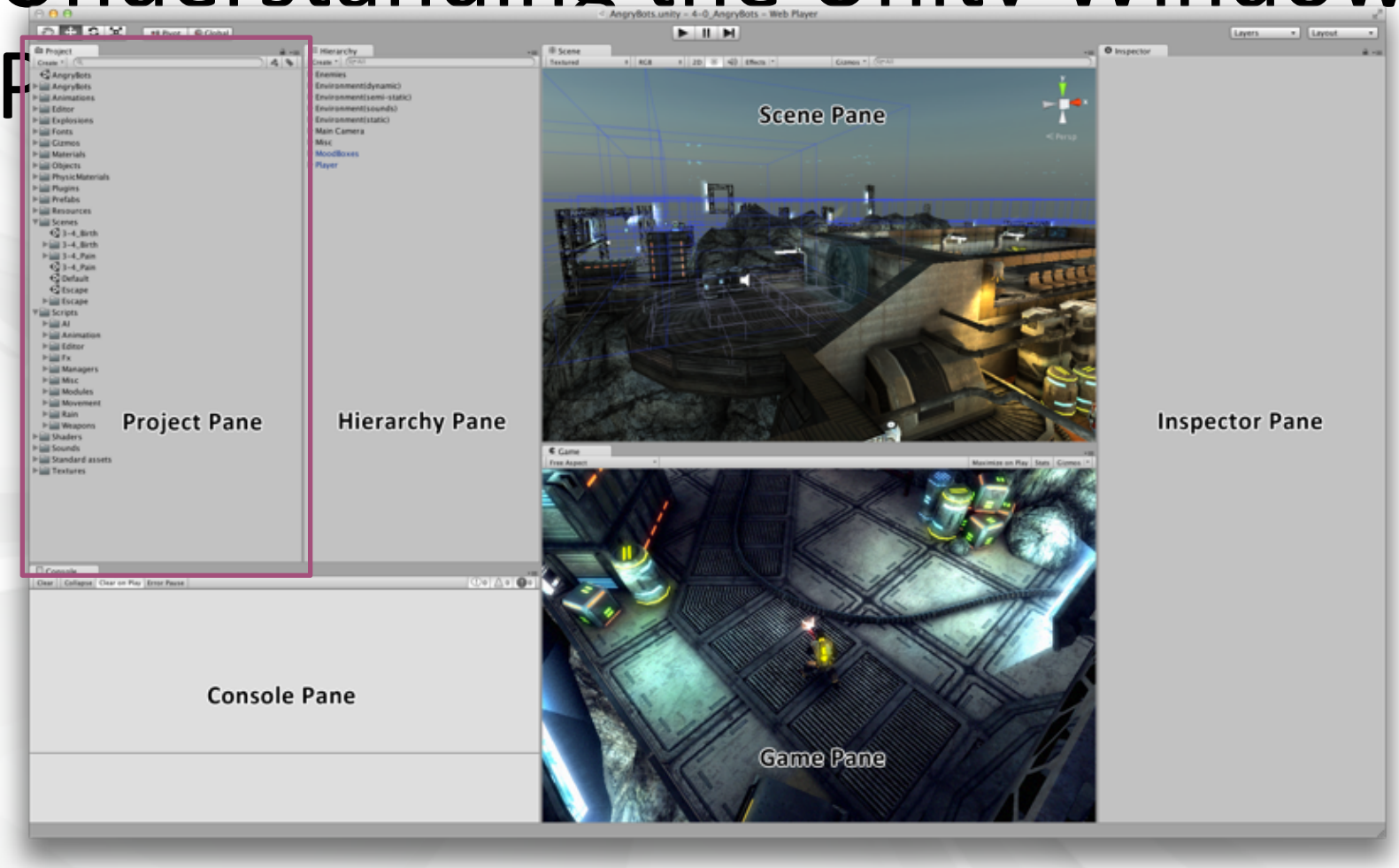
Game Pane: **Shows you a preview of the gameplay. Shows the view from the Main Camera in the scene.**

Understanding the Unity Window P



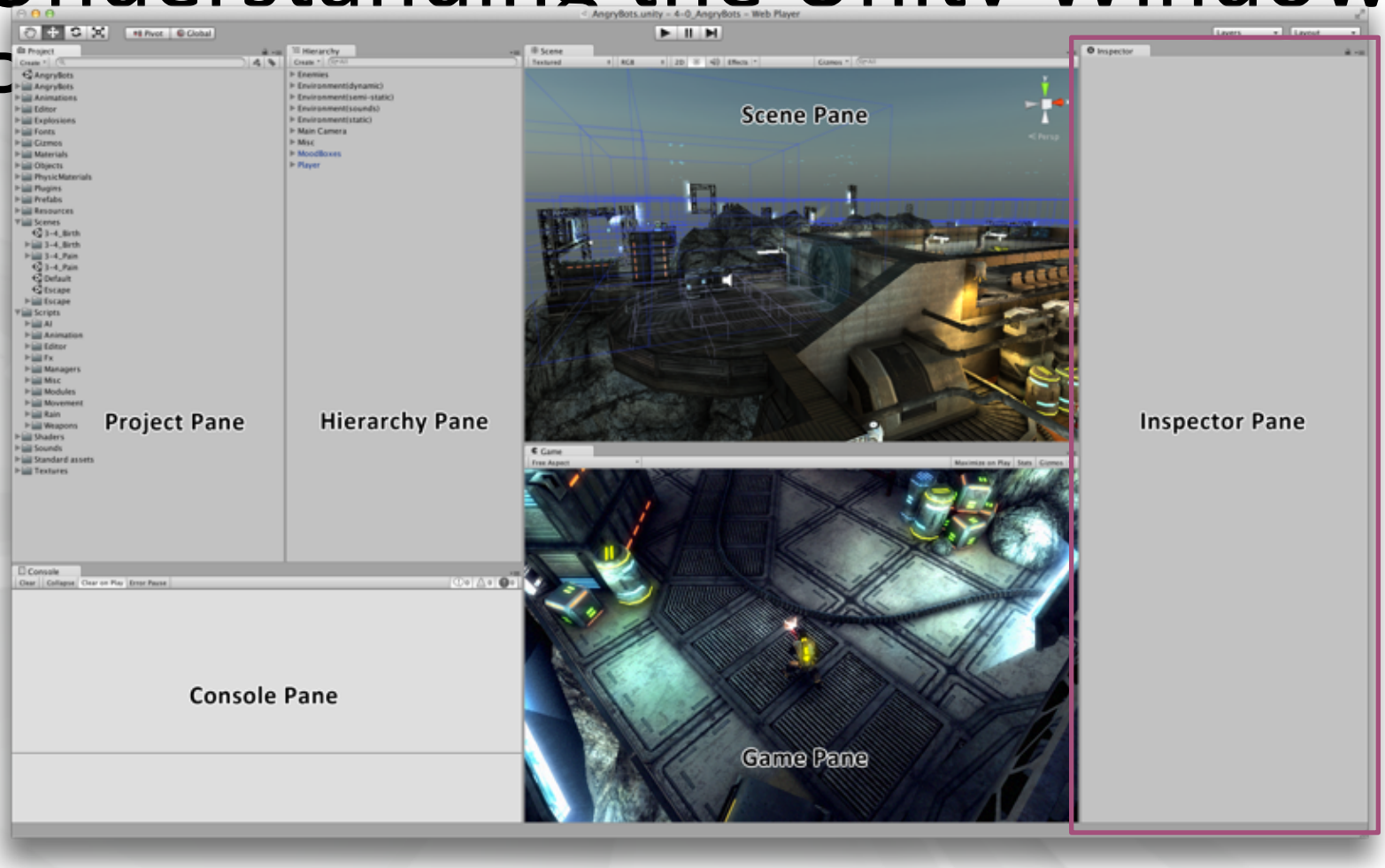
Hierarchy Pane: **A list of every GameObject in the scene. Maintains a hierarchy of parent and child GameObjects.**

Understanding the Unity Window



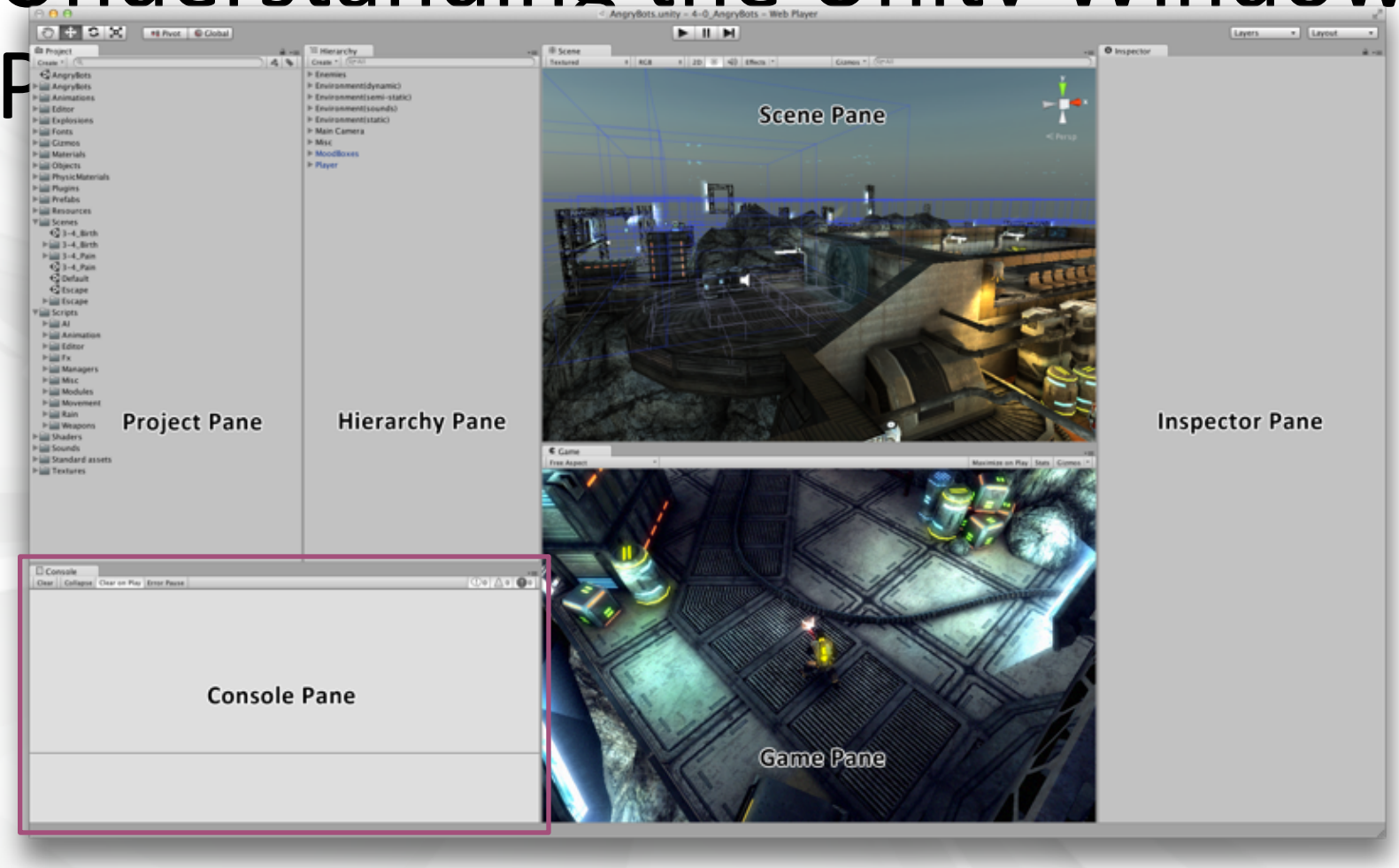
Project Pane: Collection of all assets in the Unity project: everything from models to C# code, images, and sounds.

Understanding the Unity Window P



Inspector Pane: **Shows details of any selected asset. Allows you to edit the details of any GameObject.**

Understanding the Unity Window P



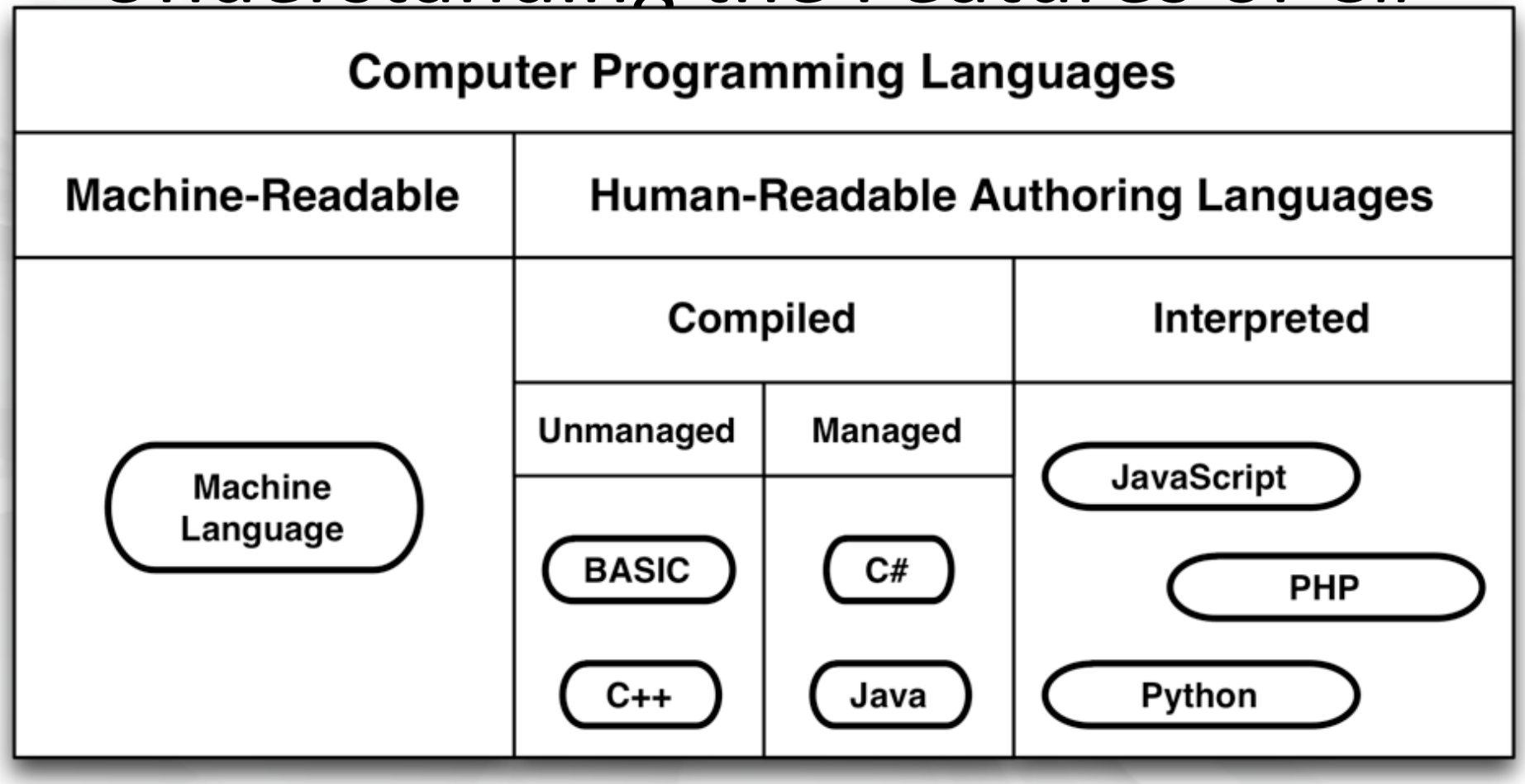
Console Pane: **Shows messages from Unity and from the C# scripts that you write. Used extensively in Chapter 18.**

Dealing with C#...

Topics

- The Features of C#
  - C# is a Compiled Language
  - C# is Managed Code
  - C# is Strongly Typed
  - C# is Function Based
  - C# is Object-Oriented
- Reading and Understanding C# Syntax

Understanding the Features of C#



Hierarchy of Computer Languages

Understanding the Features of C#

- C# is *Managed Code*
  - C, C++ - must allocate and deallocate memory
  - In *managed code*, allocation and deallocation are handled automatically
  - This makes it less likely that you will accidentally claim all of the memory

VARIABLES IN COMPUTER LANGUAGES

- A variable is a name that can hold a value
- Variables in computer languages can Numbers
  - Words, sentences, paragraphs, novels…
  - Images, sounds, 3D models, animations…
  - Functions and methods
  - Classes and GameObjects

Understanding the Features of C#

- C# is *Strongly Typed*
  - non-strongly typed
    - Here's a box. Throw whatever you want in it…
  - *strongly typed* language
    - Custom 'boxes'
    - Must put items in proper location

Understanding the Features of C#

- C# is *Function Based*
  - Computer programs used to be just a series of commands
  - Functional languages allow the encapsulation of commands
    - "If you see a store on the way, please `BuySomeMilk()`."
    - The `BuySomeMilk()` function encapsulates the many actions involved in finding and purchasing milk.

Understanding the Features of C#

- C# is *Object-Oriented*
  - Functions and data used to be separate
  - Object-oriented programming introduced *classes*
  - Classes combine functions and data into a single *object*
    - Variables in classes are called *fields*
    - Functions in classes are called *methods*

Reading and Understanding C# Syntax

- C# statements have syntax rules
  - int x = 5;
    - *Declares* a variable named x of the type int
      - If a statement starts with a type, the second word of the statement becomes a new variable of that type
    - *Defines* the value of x to be 5
      - The = is used to assign values to variables
    - All C# statements end with a semicolon (;)
      - A semicolon is used because the period is already used in decimal numbers

Reading and Understanding C# Syntax

- C# statements also have syntax rules
  - int x = 5;
  - int y = x * (3 + x);
    - Declares a variable named y of the type int
    - Adds 3 + x for a value of 8 (because x = 5)
      - Just as in algebra, order of operations follows parentheses first
    - Multiplies x * 8 for a value of 40 (5 * 8 = 40)
    - Defines the value of y to be 40
    - Ends with a semicolon (;)

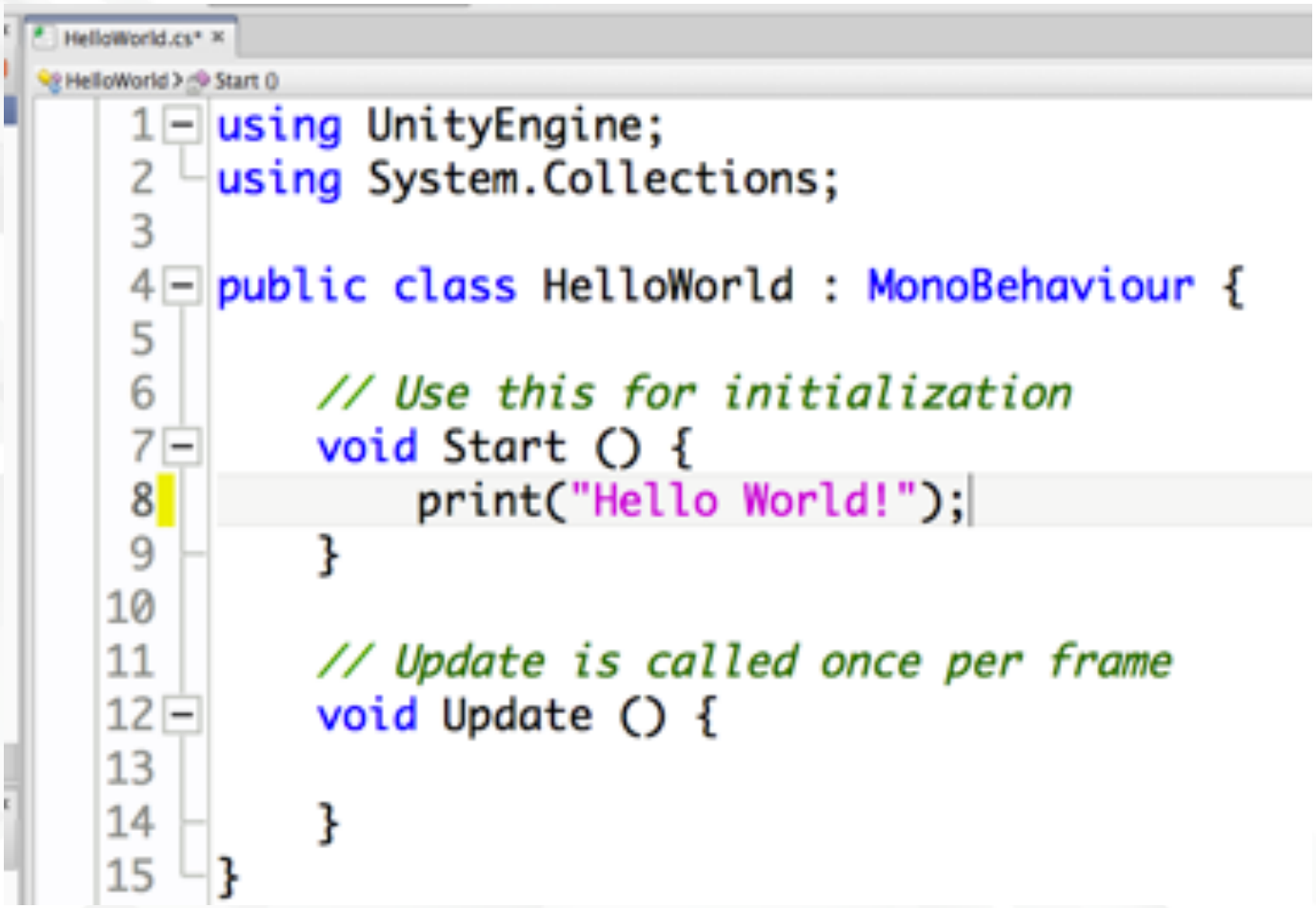
Reading and Understanding C# Syntax

- C# statements also have syntax rules
  - string greeting = "Hello World!";
    - Declares a variable named greeting of the type string
      - *strings* can hold a series of characters like a word or novel
    - Defines the value of greeting to be "Hello World!"
      - Anything between double quotes is a *string literal*, a value to be assigned to a string variable
    - Ends with a semicolon (;)

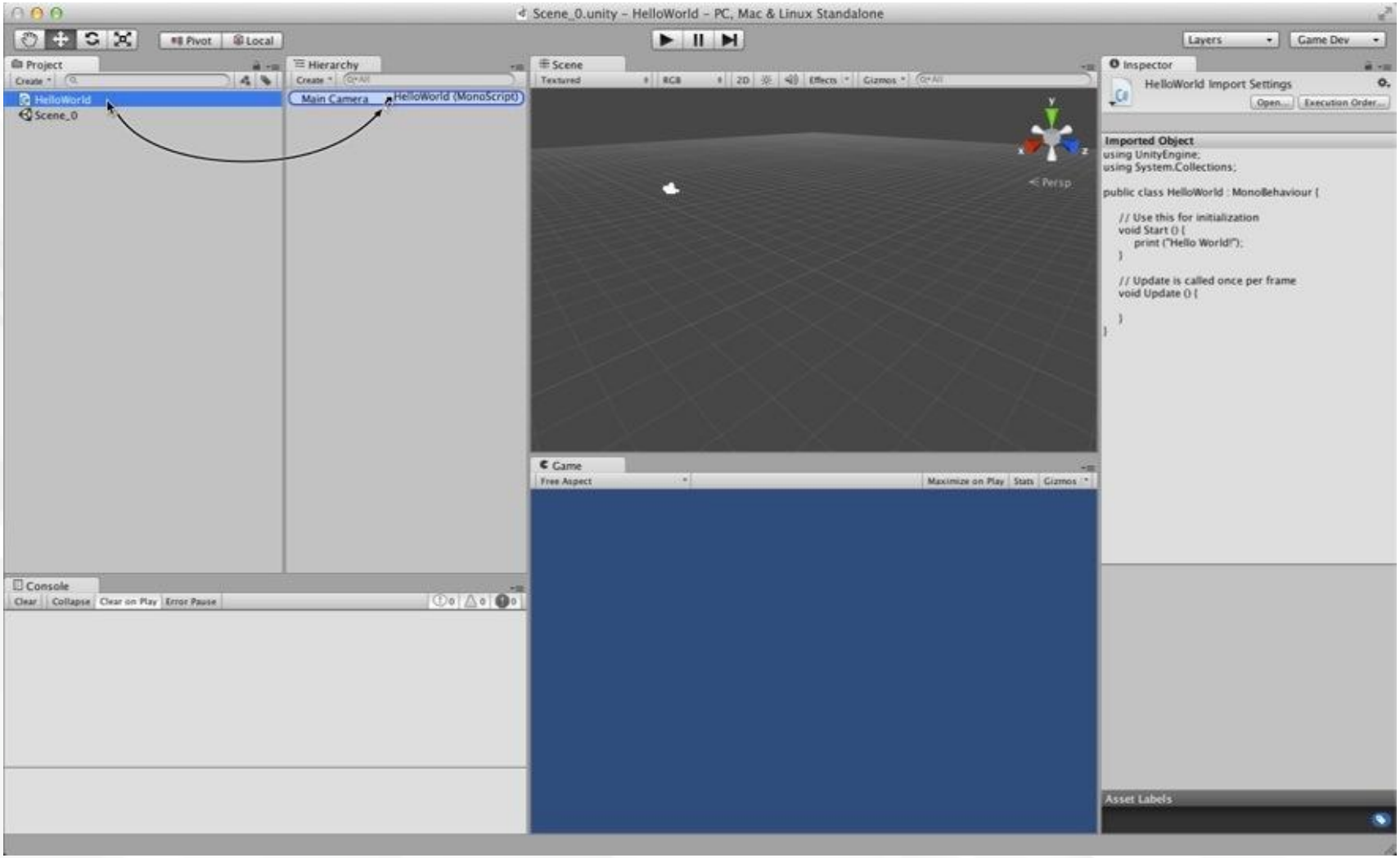
Reading and Understanding C# Syntax

- C# statements also have syntax rules
  - string greeting = "Hello World!";
  - print(greeting);
    - *Calls* the function print()
      - When a function is called, it executes its actions
    - Passes the *argument* greeting into the function print()
      - Some functions take *arguments*: data that changes how the function acts
      - The print() function will print the string greeting to the Console pane in Unity
      - The Console pane will display "Hello World!"
    - Ends with a semicolon (;)

Hello World?

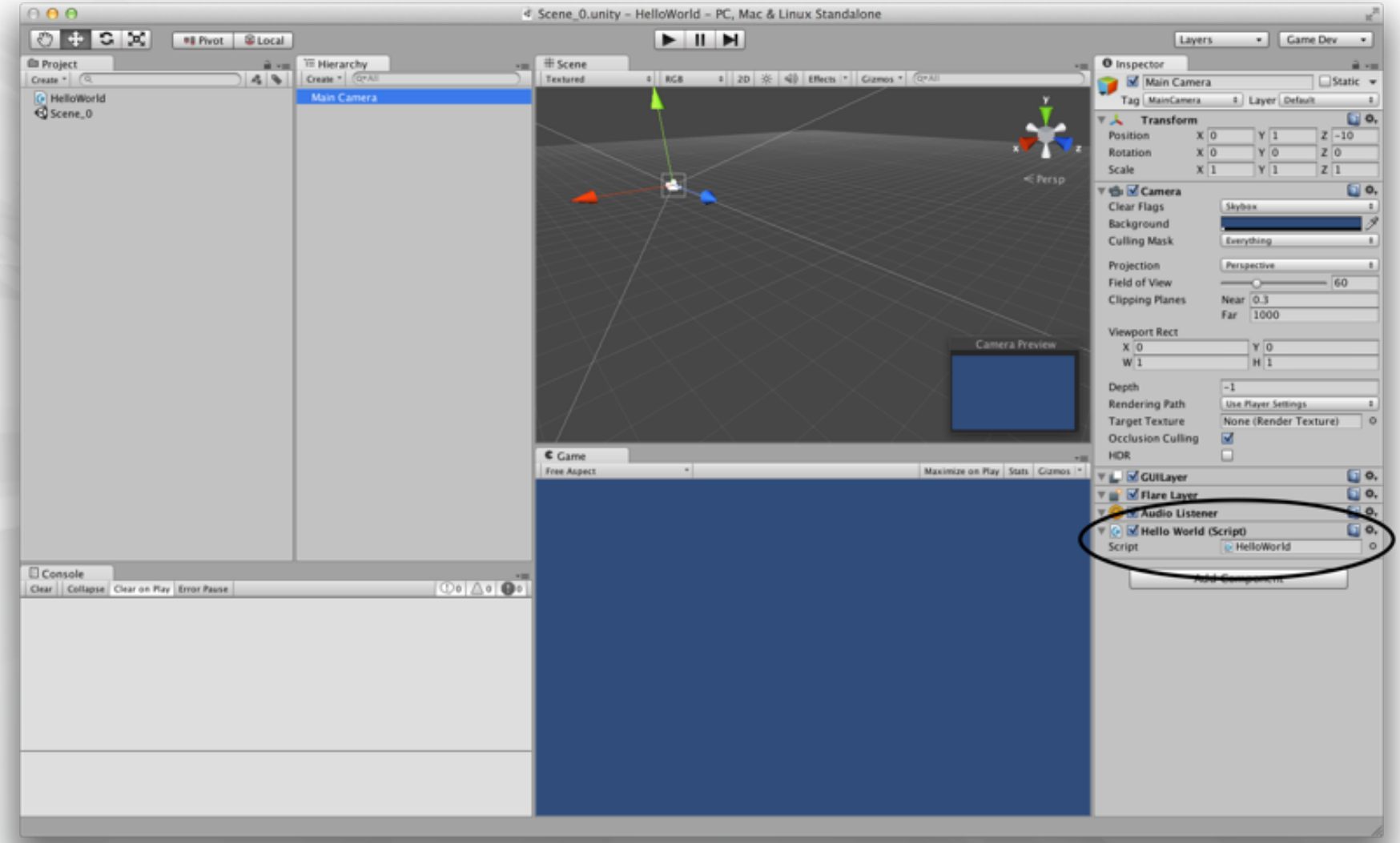

HelloWorld.cs*
HelloWorld > Start ()
1 using UnityEngine;
2 using System.Collections;
3
4 public class HelloWorld : MonoBehaviour {
5
6 // Use this for initialization
7 void Start () {
8 print("Hello World!");
9 }
10
11 // Update is called once per frame
12 void Update () {
13
14 }
15 }

Attaching Scripts to GameObjects



To work in Unity, a C# script must be *attached* to a GameObject

Attaching Scripts to GameObjects



This makes the script a *component* of the GameObject

Start() and Update()

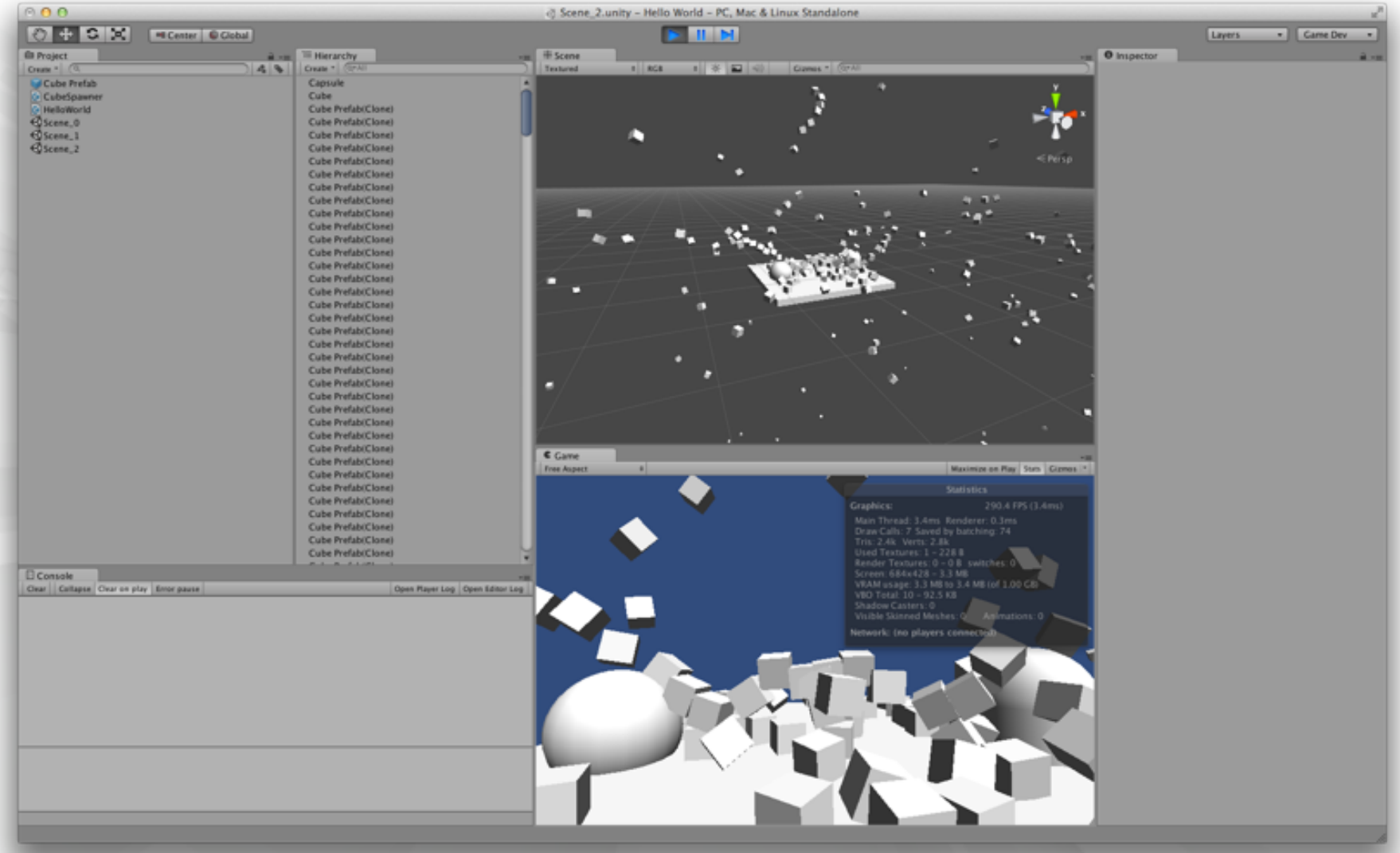
- You make use of Start() and Update() in the HelloWorld project
  - void Start() {…}
    - Called once
    - Called immediately before the first Update() is called
  - void Update() {…}
    - Called every frame
    - This can happen over 200 times per second!
  - void Awake() {…} (not used in HelloWorld, but important)
    - Called once
    - Called at the moment the GameObject is created
    - Guaranteed to be called before Start()

GameObject Prefabs and Instantiation

- A *prefab* is a mold from which GameObject *instances* can be made
  - Created by dragging a GameObject from the Hierarchy pane into the Project pane
  - Can be assigned to a script variable in the Inspector pane
    - `public GameObject    gameObjectPrefab;`
  - Then, an instance of the prefab can be created in code
    - `Instantiate( gameObjectPrefab );`
- This is used in HelloWorld to create thousands of instances of a Cube GameObject prefab

The HelloWorld Project

- Output "Hello World!" to the Console pane
  - Once using Start()
  - Many times using Update()
- Create a Cube prefab that reacts to gravity & physics
- Instantiate an instance of the Cube prefab
  - Once using Start()
  - Many times using Update()
    - This will create a cascade of thousands of Cube instances
  - Over other physically-modeled objects



The final HelloWorld scene

VARIABLES AND COMPONENTS

Variables in C#

- Quick recap:
  - A *variable* is a named container for data
  - Variables in C# are *typed*, so they can only hold one type of data
  - Variables need to be *declared* to be used
  - Assigning a value to a variable is called *defining* the variable
  - A *literal* is a value that is entered into your code and can be assigned to a variable
    - The 5 above is an integer literal
    - string literals are surrounded by double quotes: `"Hello World!"`
    - float literals are followed by an f: `3.14f`

Important C# Variable Types

- Core C# variable types start with a lowercase character
  - bool - A 1-bit True or False Value
    - `bool verified = true;`
  - int – whole number –2,147,483,648 and 2,147,483,647
    - `int nonFractionalNumber = 12345;`
  - float - A 32-bit Decimal Number
    - `float notPreciselyOneThird = 1.0f / 3.0f;`
  - Char - A 16-bit Character (Unicode)
    - `char theLetterA = 'A';`

Important C# Variable Types

- Core C# variable types start with a lowercase character
  - String - A Series of 16-bit Characters
    - `string theFirstLineOfHamlet = "Who's there?";`
    - `char theCharW = theFirstLineOfHamlet[0];`
    - `char questionMark = theFirstLineOfHamlet[11];`
    - `int len = theFirstLineOfHamlet.Length;`
  - Class - A Collection of Functions and Data
    - Creates a new variable type
    - Everything between the braces `{ }` is part of the class

C# 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 Variable Types

- Because they are classes, important Unity variable types all start with an uppercase character
  - Vector3
  - Color
  - Quaternion
  - Mathf
  - Screen
  - SystemInfo
  - GameObject

Important Unity Variable Types

- Vector3 – A collection of 3 floats
  - 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

Important Unity Variable Types

- Color – A color with transparency information
  - 4 floats 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

Important Unity Variable Types

- Color – A color with transparency information
  - 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.

// 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
```

Important Unity Variable Types

- 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
  - 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

Important Unity Variable Types

- 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.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
```

Important Unity Variable Types

- 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
```

Important Unity Variable Types

- 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
```

Important Unity Variable Types

- GameObject – Base class for all objects in scenes
  - Composed of *components*

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

  - Always has a Transform component
  - Instance variables and functions

```
go.name     // The name of the GameObject ("MyGO")
go.GetComponent<Transform>()     // The Transform component
go.transform          // A shortcut to the Transform component
go.SetActive(false)     // Make this GameObject inactive
go.name     // The name of the GameObject ("MyGO")
```

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

Unity GameObject Components

GameObject name, tag, and layer

Transform Component

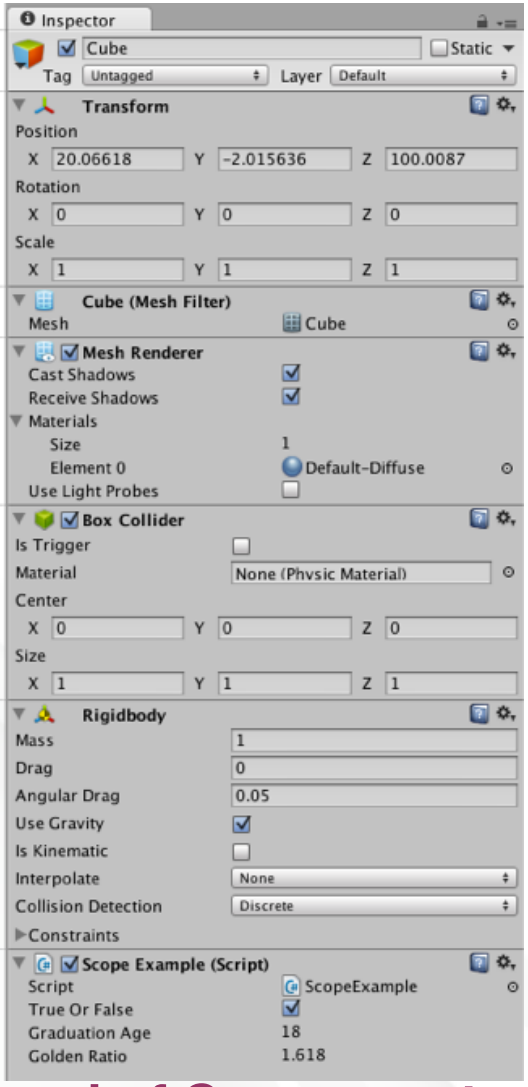
MeshFilter Component

Renderer Component

Collider Component

Rigidbody Component

Script Component



GameObjects are composed of Components

Unity GameObject Components

- Transform component
  - 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)
```

Unity GameObject Components

- MeshFilter component
  - 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

Unity GameObject Components

- Renderer component
  - 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)

Unity GameObject Components

- Collider component
  - 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

Unity GameObject Components

- Rigidbody component
  - 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

Unity GameObject Components

- (Script) components
  - 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 throughout the book
  - 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.