UNIVERSITÀ DEGLI STUDI
DI MILANO

Unity 101 - Basics

How to Rub it "The Right Way"

Dario Maggiorini (dario@di.unimi.it)

PLAY
Playlab fOr inNovation in Games

What Unity is NOT

- A 3D editor

- A revision control system

- A development environment

- … and a game engine

What IS Unity

The essence of Unity is its core, *driving* our game during execution

What IS Unity

Everything is hidden from us: all we see is a black box we are supposed to use.
We usually call this box *the runtime*

Runtime

What IS Unity

What IS Unity

THIS! Is Unity

Unity is very like a candy dispenser

What IS Unity

THIS! Is Unity

Unity comes in one piece, and you are supposed to use it “from the outside”

Unity is very like a candy dispenser

What IS Unity

Unity plugins

THIS! Is Unity

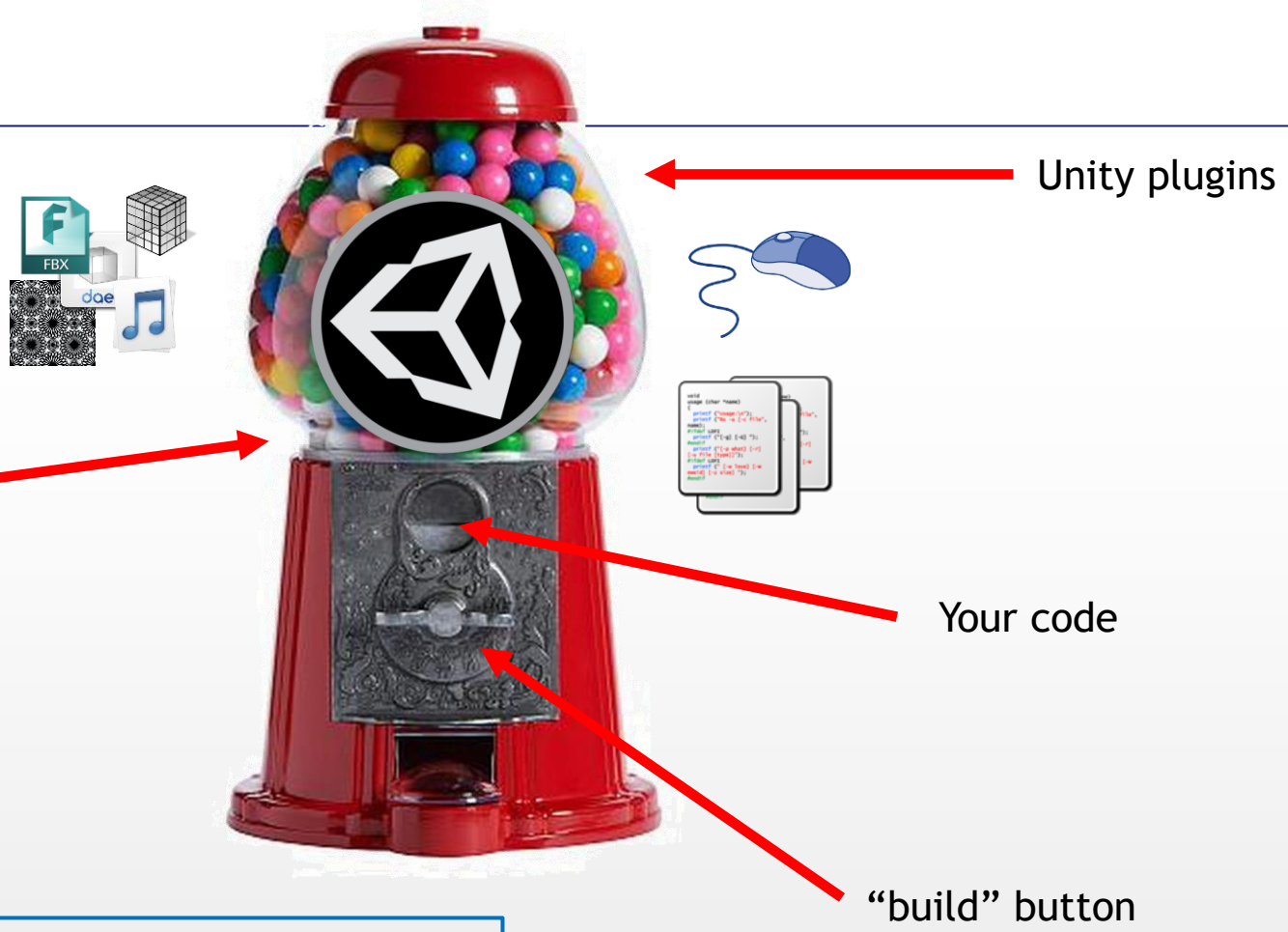
Plugins can be added to the benefit of your project from the asset store.

Like sweet dreams, some of them are free, some of them are cheap, some of them are really expensive

What IS Unity

Your code and assets are required to make the clockwork spin

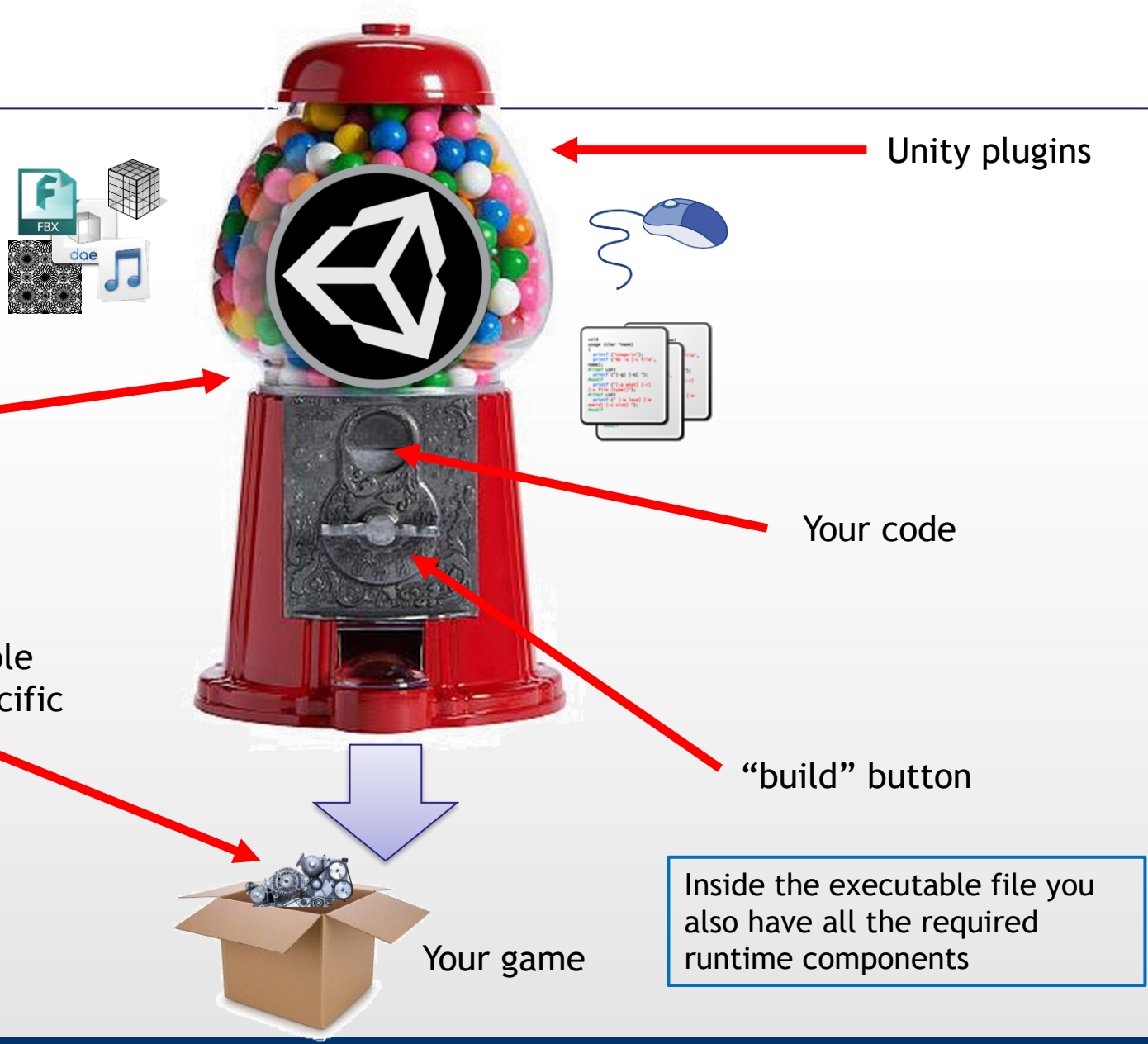
What IS Unity



Starting the build process is like turning the lever: it will trigger the process that puts everything together and produce a candy

What IS Unity

What IS Unity

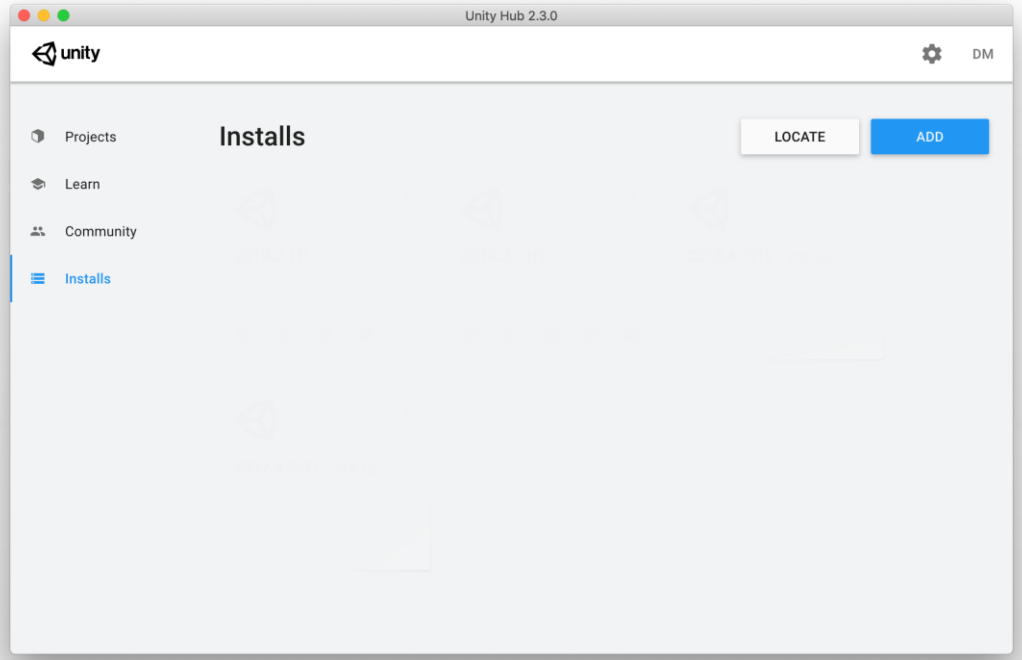


Downloading and Installing Unity

1. Go to http://store.unity.com/
    1. select the “individual” tab
    2. click “Try Personal” (or student, but you will need approval)

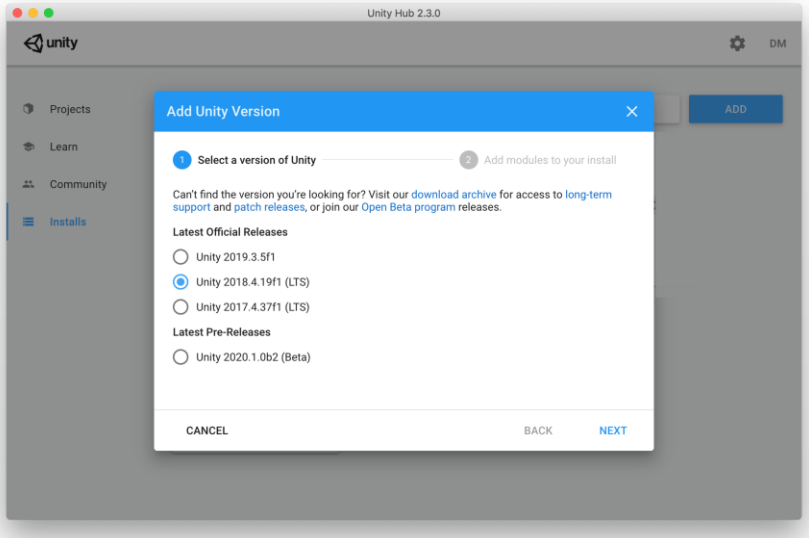
2. Follow the instructions and download the installer for the unity hub

3. From within the hub, install unity via the “Installs” tab (on the left) and the “Add” button (on the right)

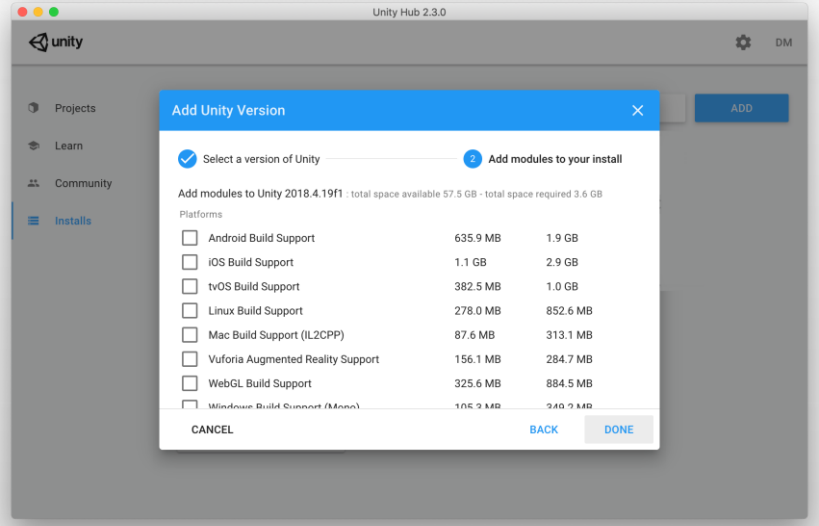


Downloading and Installing Unity

4. Select the version you like



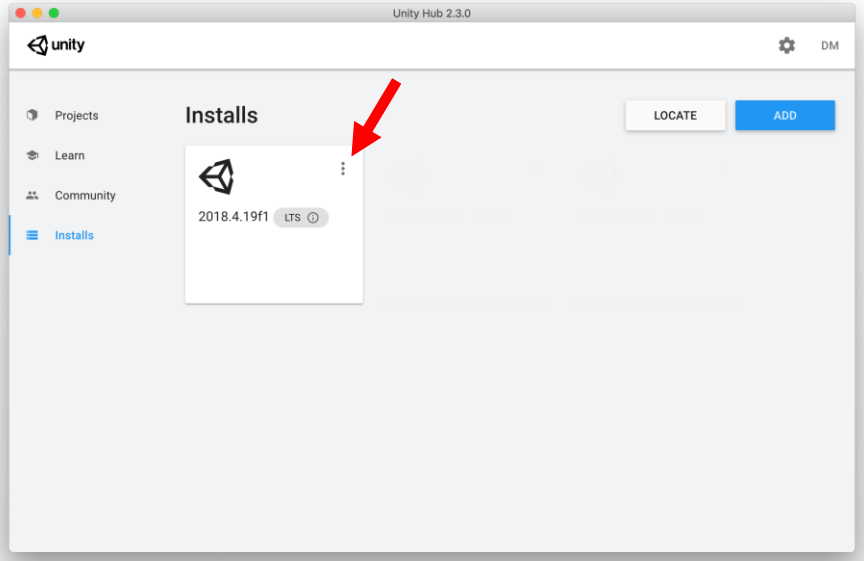
5. Pick the packages you need/want
   - If unsure, just keep the default



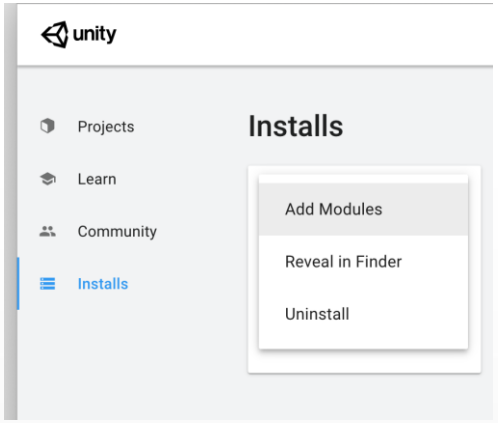
6. And, after a while, you are good to go

Downloading and Installing Unity

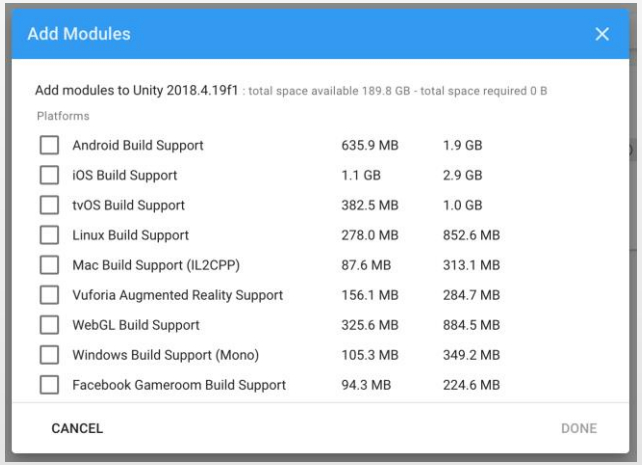
7. Forgot a module?
   You can install it later!



dots menu

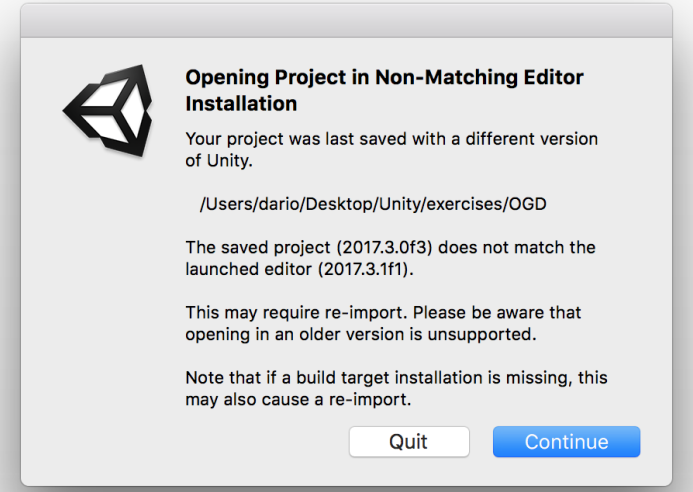


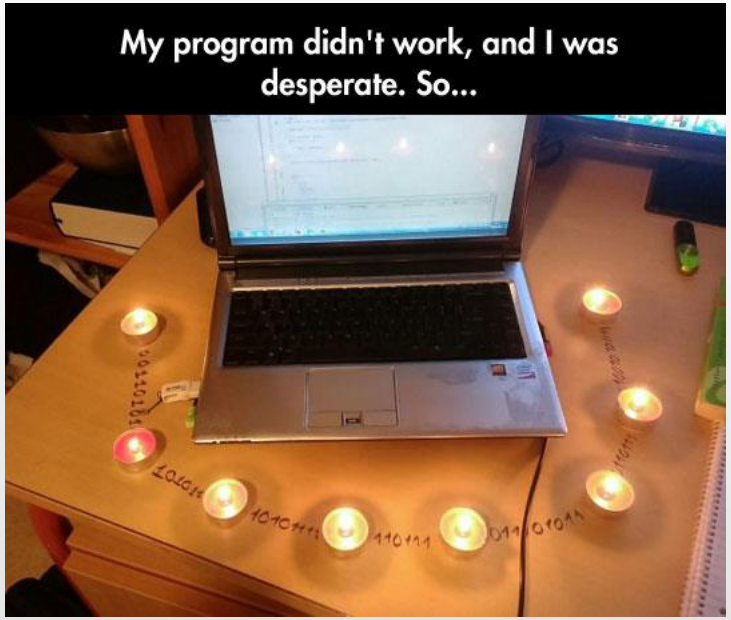
Right click



A Warning About Versions

- Never, EVER, change your unity version during development
- Even minor versions can insert (subtle) incompatibilities and break all your work
  - **Mind about keeping a backup!**

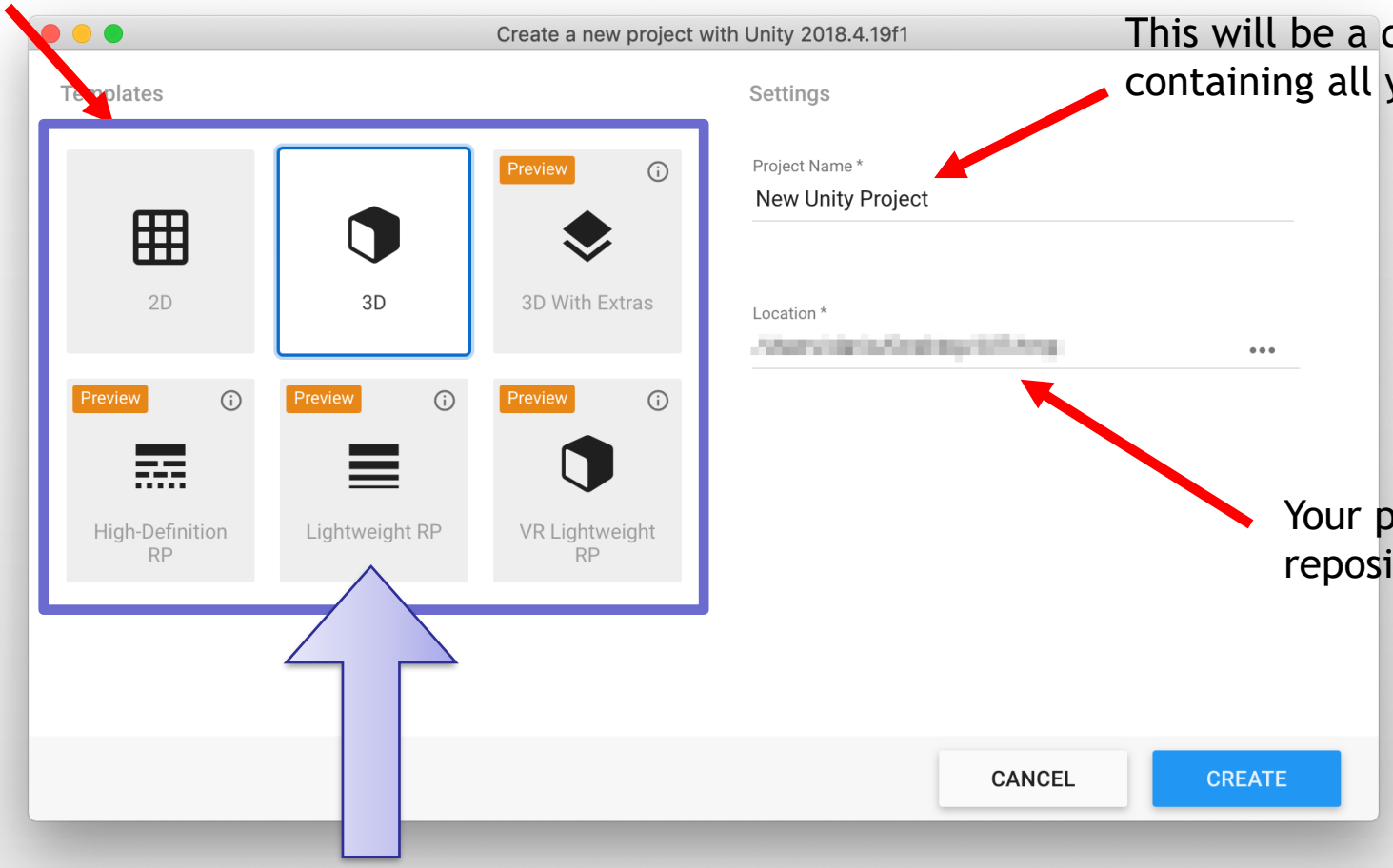




Starting Up

(Your hub might look slightly different depending on your version)

Depends on your game

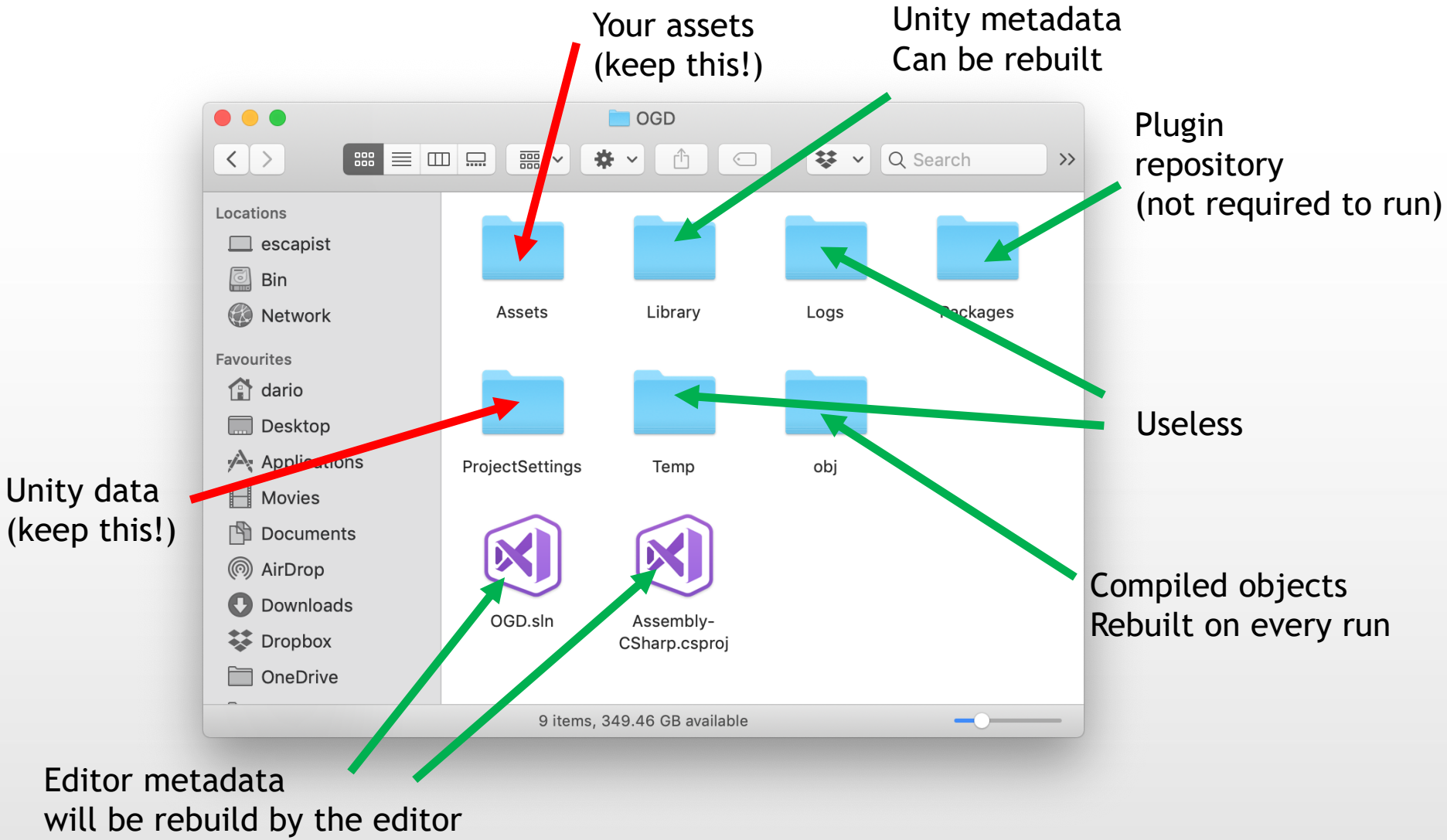


This will be a directory containing all your stuff

Your projects repository

These are just presets.
You can manually rebuild them from the editor

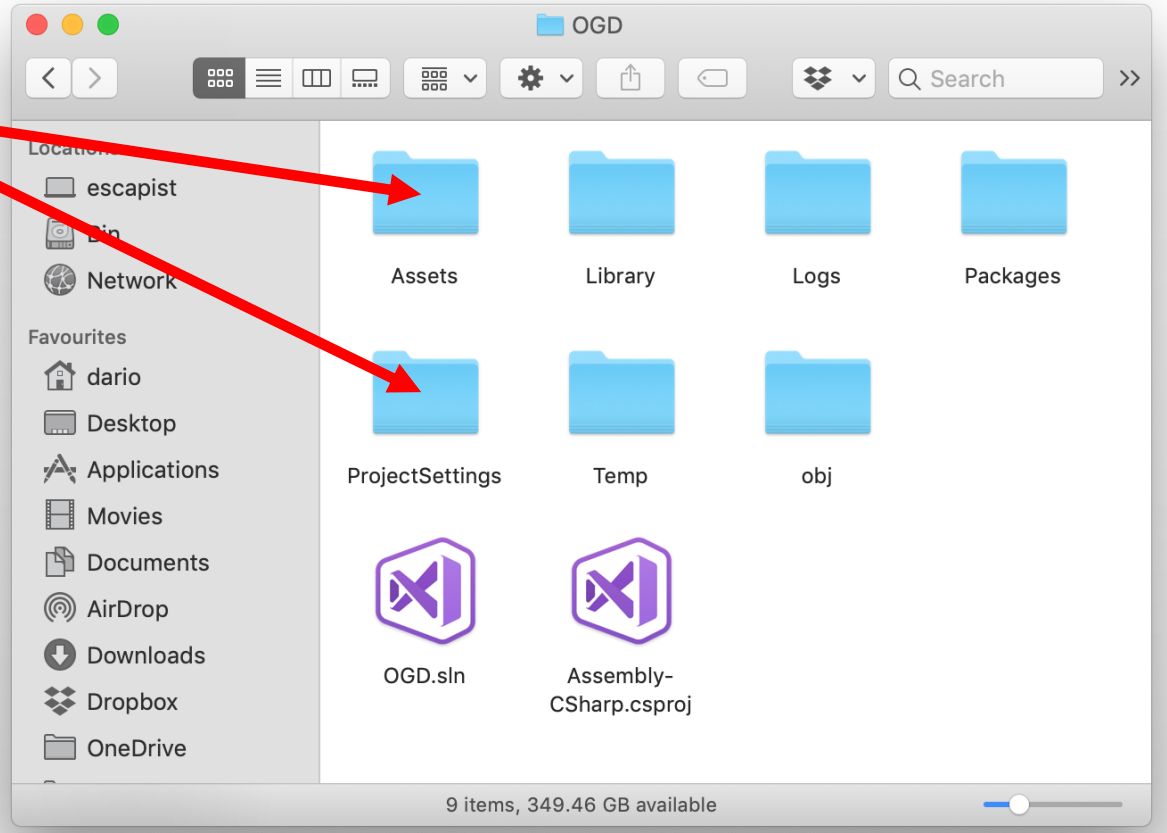
Files and Folders



Files and Folders

This is all you need to save/backup/share of your project (GIT included!)

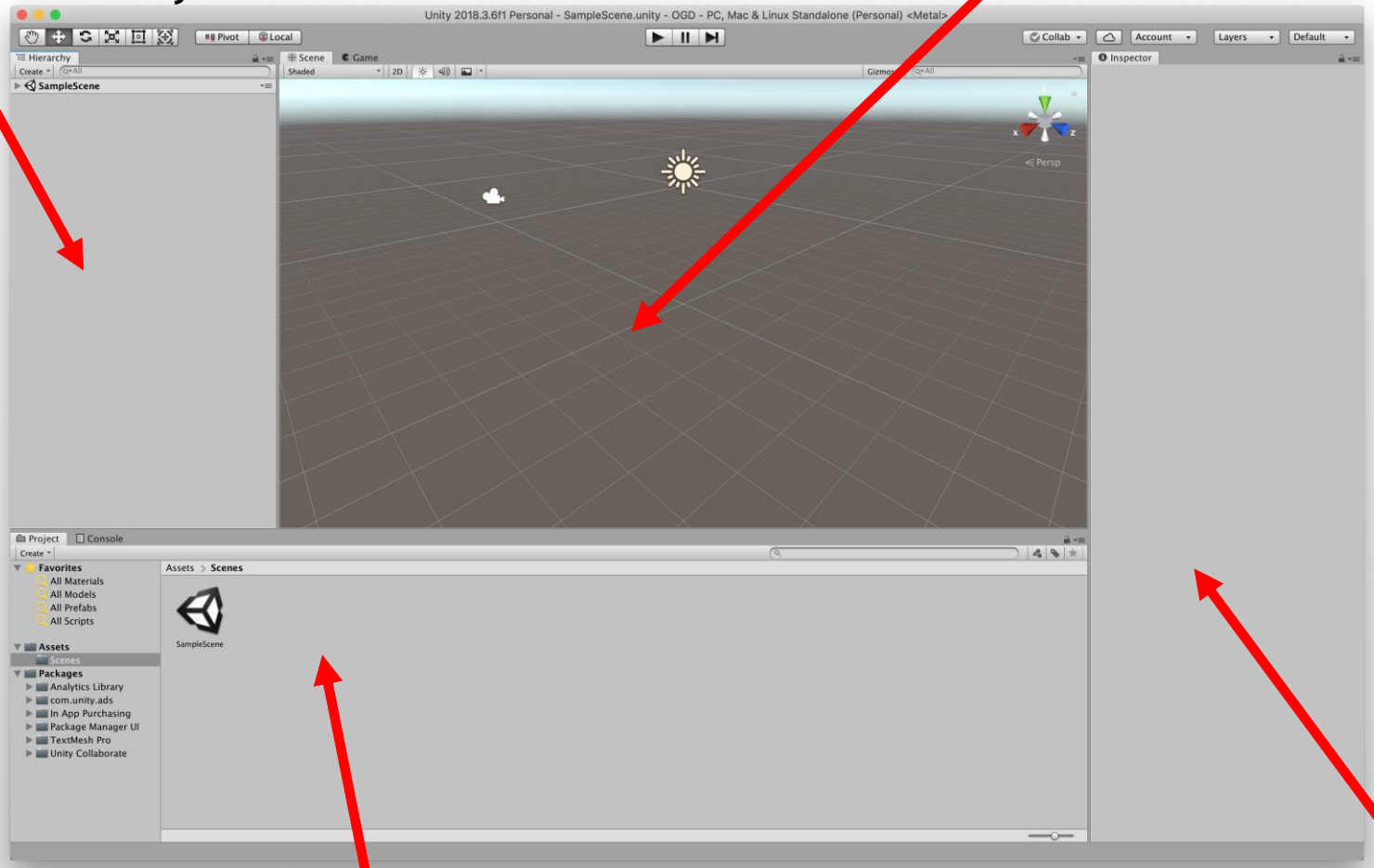
Moreover, they will be only around 10% of the total storage space



Unity Interface Anatomy

Scene hierarchy

Scene window

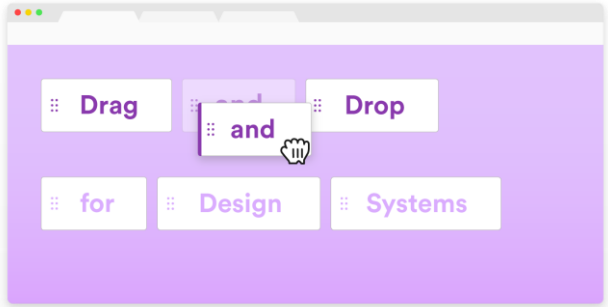


Asset database

Objects inspector

Asset Database

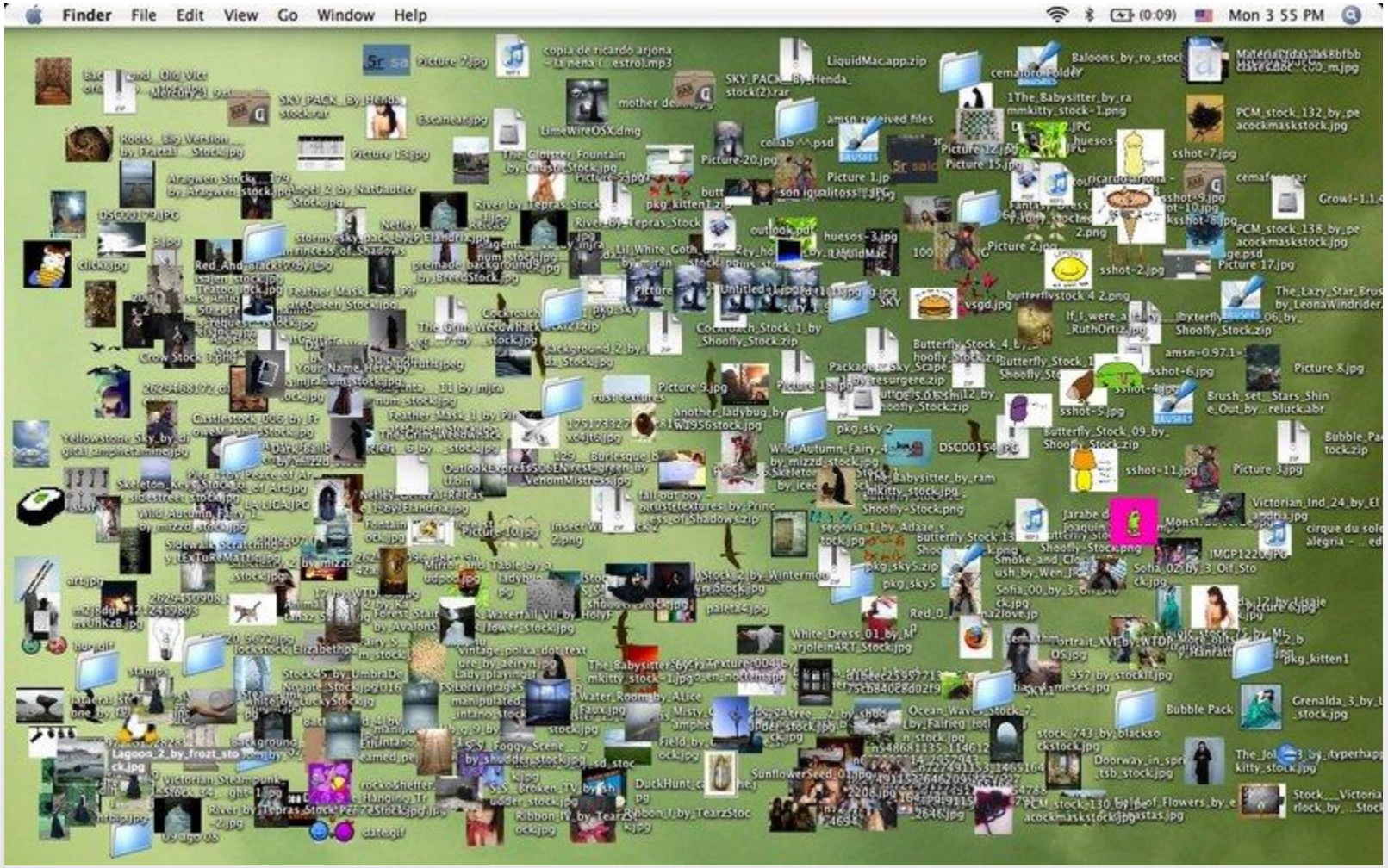
- This is where you store all data (any kind of data) to be used to build your game
  - Just drag and drop stuff inside



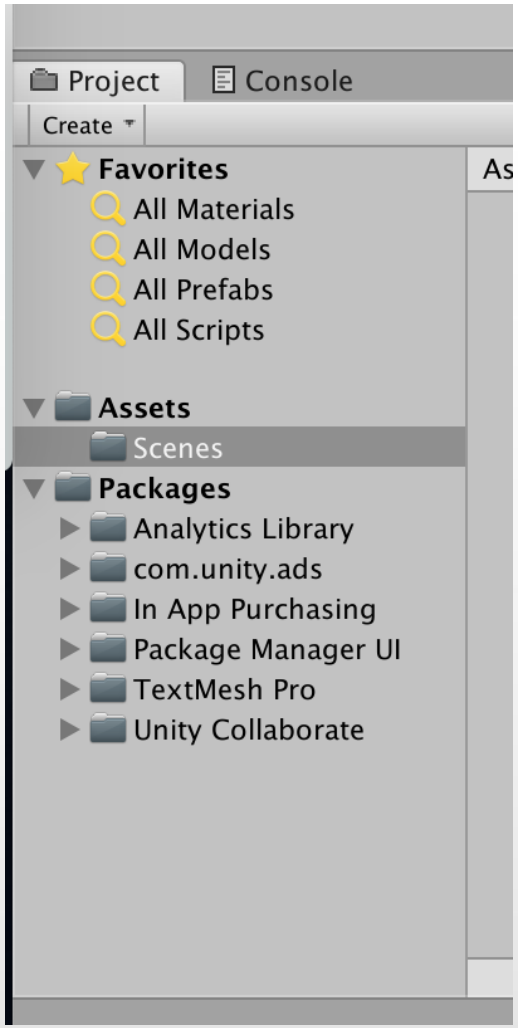
- From external sources
  - Images
  - 3D models
  - Sounds

- Created using Unity
  - Scripts
  - Materials
  - Animations
  - Visual and audio effects

Warning, if You do not Like Using Folders ...

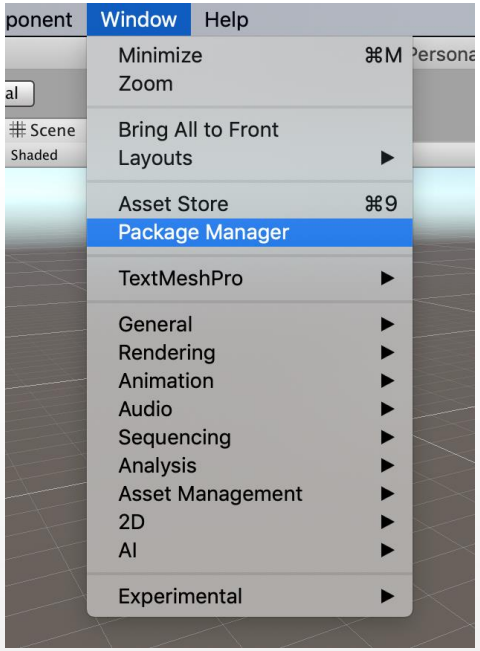


Daf***?

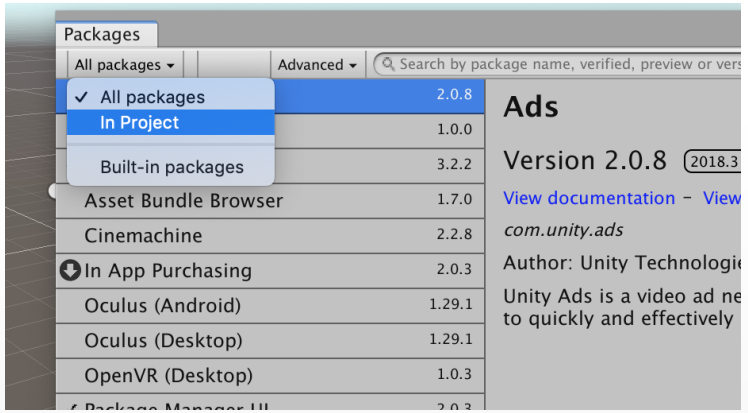


- Pre-loaded packages, courtesy of Unity Technologies
- They are NOT part of your project
  - You can delete everything but the *Package Manager UI*

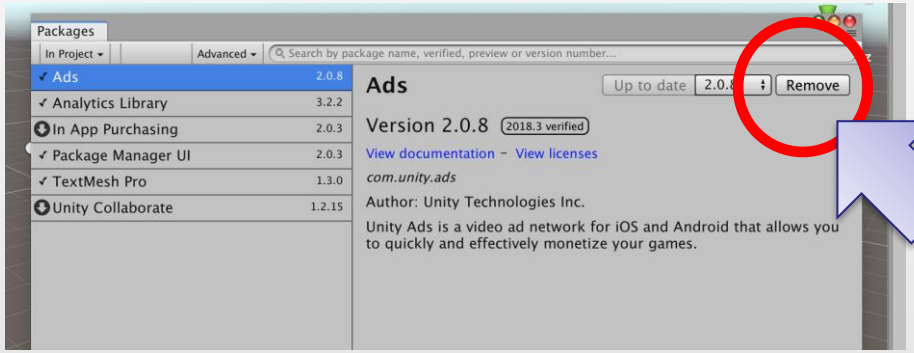
Getting Rid of Unwanted Packages



Right click



Right click

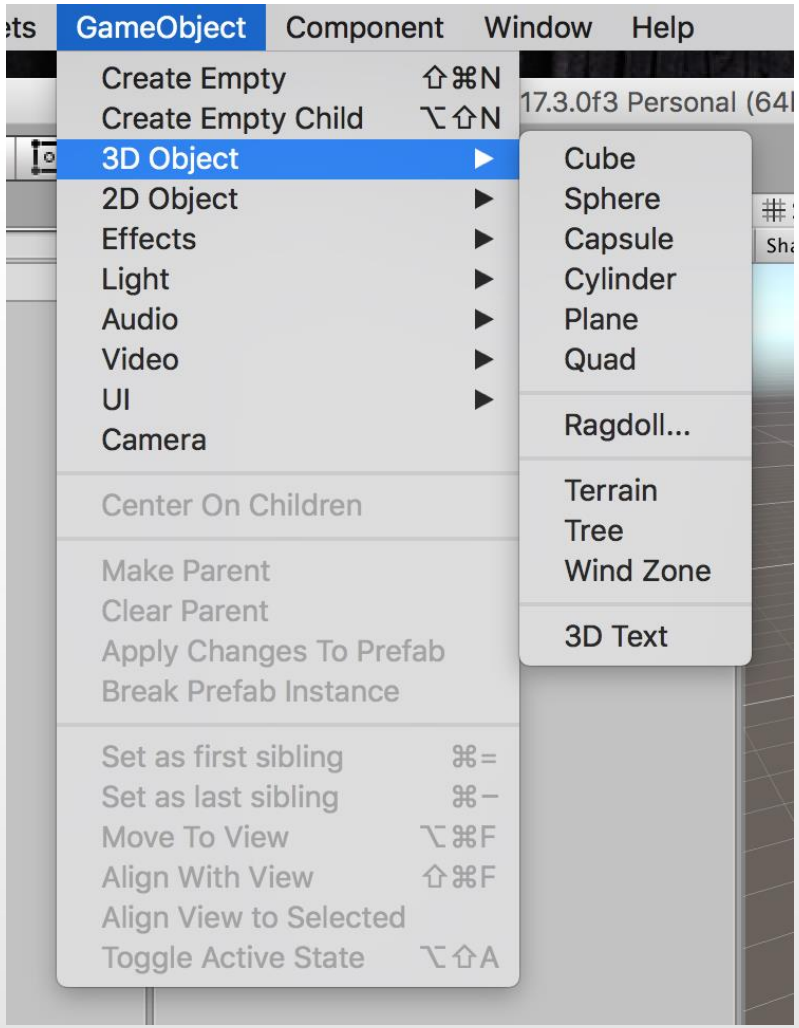


Right click

Object Hierarchy

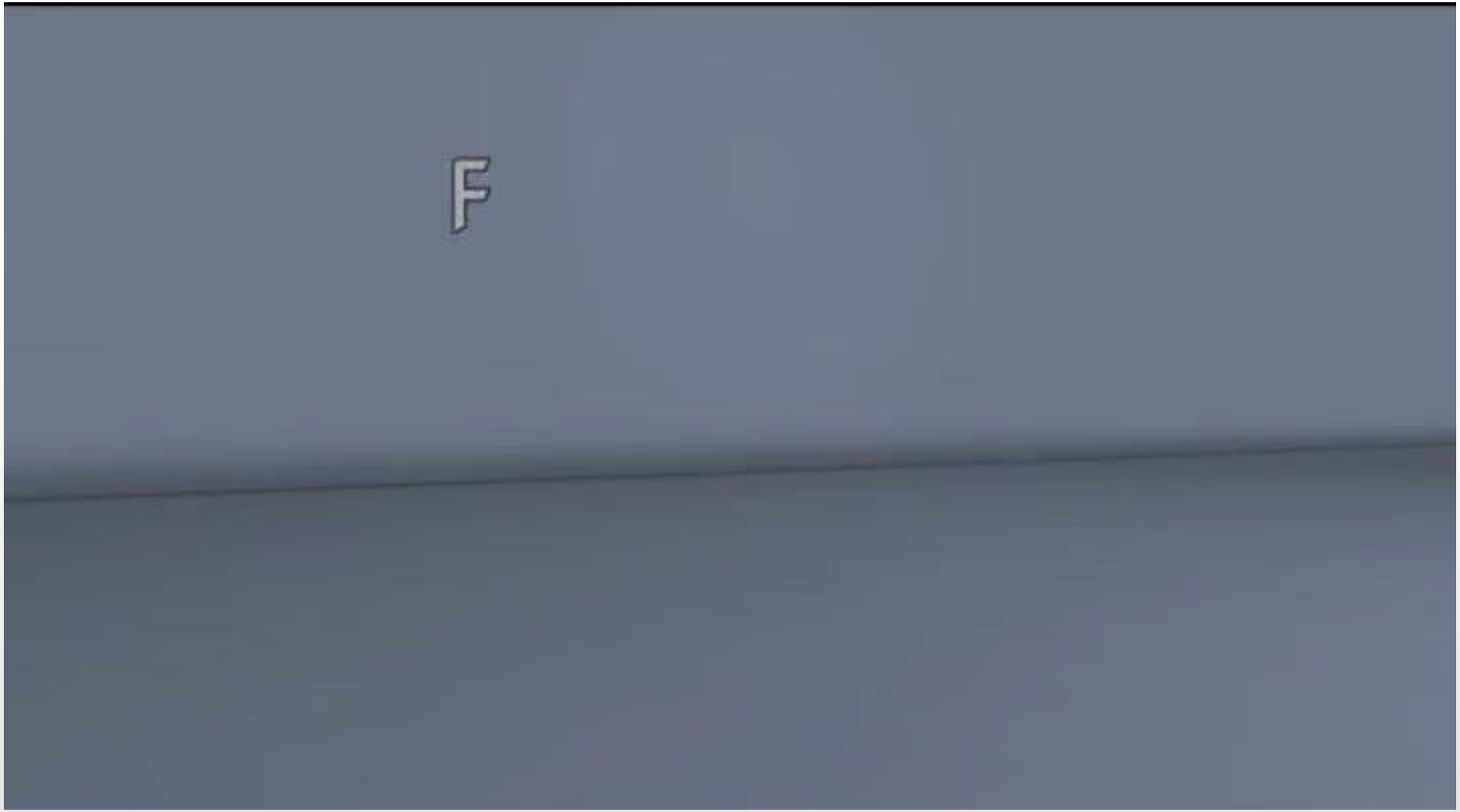
- This is a tree-based representation of your scene
- The tree root is the scene itself
- All scene children (main objects) are independent objects in the scene
- All sub-children are regrouped under a main object to create complex shapes and behaviours
- Saving your scene records your hierarchy in the asset database (will become an asset itself)

Populating Your Scene

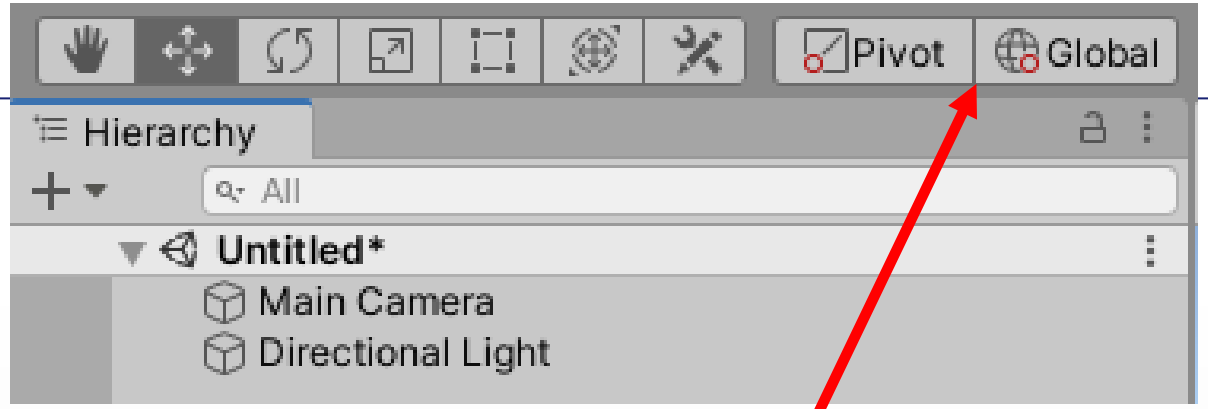


- Basic objects are available via menu
- Effects, lights, and multimedia are included
- Shapes are fine for a quick test or prototype, but forget using them for a real game

UNLESS ...



Moving Around



- Look at the upper-left corner
  - Hotkeys are Q W E R T Y

- Buttons are for:
  1. Pan
     - Only your visual
  2. Move
     - Click on the arrow or the small cube inside the object
  3. Rotate
     - Click on the thin circles
  4. Scale
     - Click on the cubic handles
  5. Plane-based manipulation
     - Useful when you have a 2D orthographic view
  6. Move + Rotate + Scale
     - 2 + 3 + 4 together

Axis reference

- Global (to the scene)
- Local (to the object)

Navigating the Scene (With Any Tool Selected)

- Pan

  - Hold CTRL-ALT and left click-drag to pan the camera around

- Orbit
  - Hold ALT and left click-drag to orbit the camera around the current pivot point (the pivot point is the geometric center of the current selection)

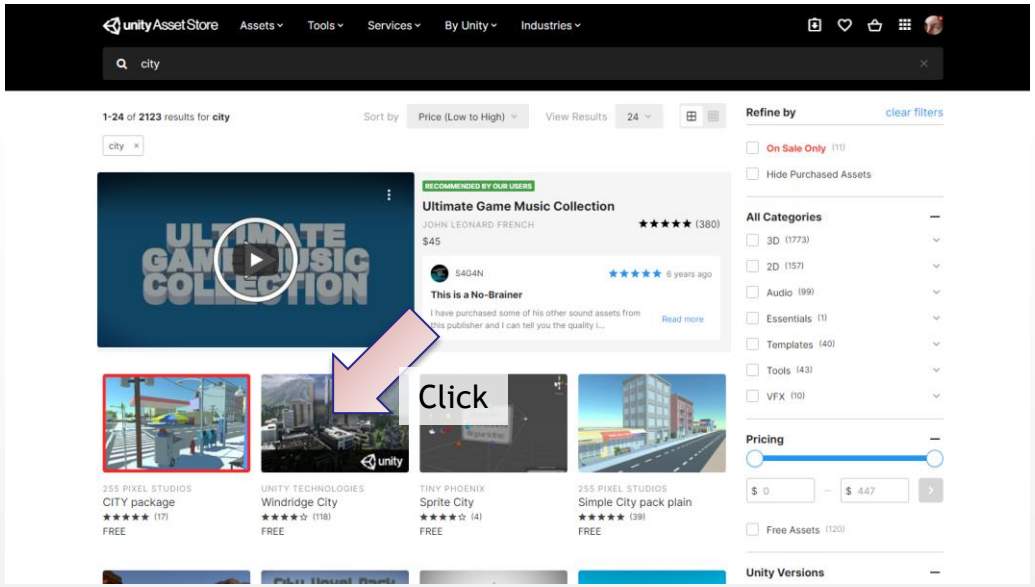
- Zoom
  - Hold ALT and right click-drag to zoom the Scene View (mouse wheel also, but will be bumpy)
- Centering
  - Double clicking on an object (either in the scene or in the scene hierarchy) will center your view on it
- Fixing the focus
  - Select an object and hit the ‘f’ key

Populating Your Scene (With Nice Assets)

- Assets can be added:
  - By drag and drop in your asset database
    - This is the usual way if you created them
  - By importing a package into your scene
    - Like when you get some of them from the unity asset store

- You can drag and drop any kind of asset
  - For 3D shapes an import procedure will start
    Verify in advance if Unity supports your 3D editor
  - Textures (images) will be stored as TIFF (lossless) with sizes in power of two (2, 4, 8, 16, 32, 64 ...) up to 2048 pixels.
    They are NOT required to be square
    Better create originals with the right size to avoid artifacts

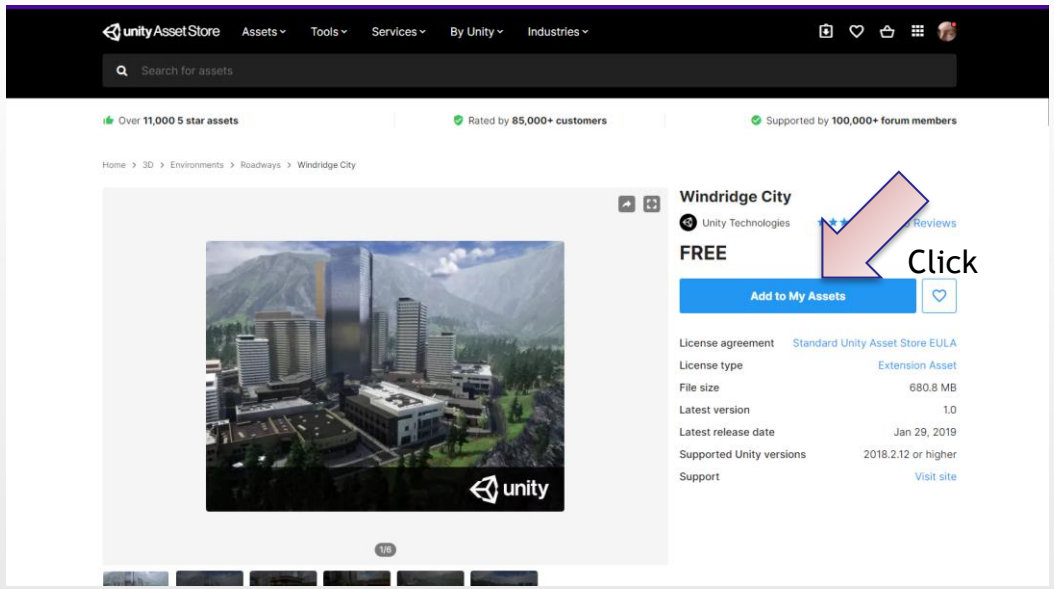
Getting Assets From the Store



1. Search

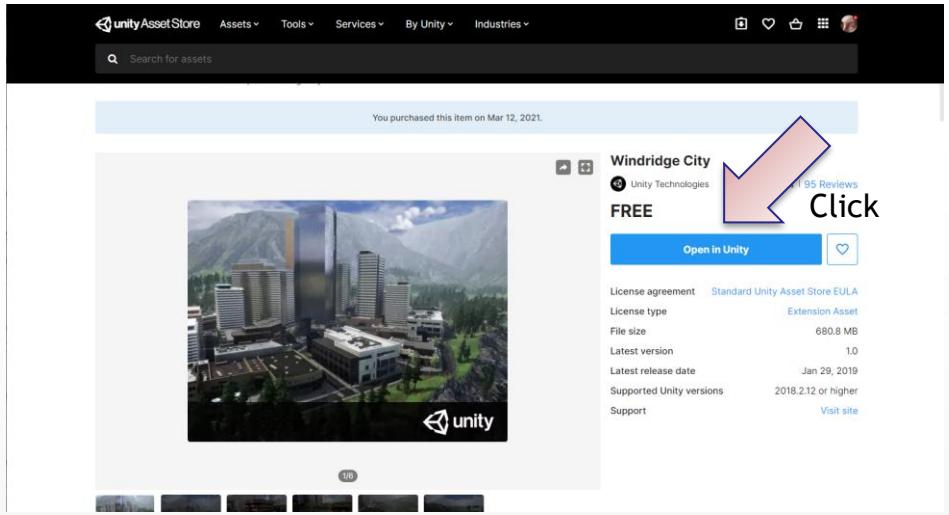
Authentication is REQUIRED
You MUST create a Unity.com account

1 … 2… 3…

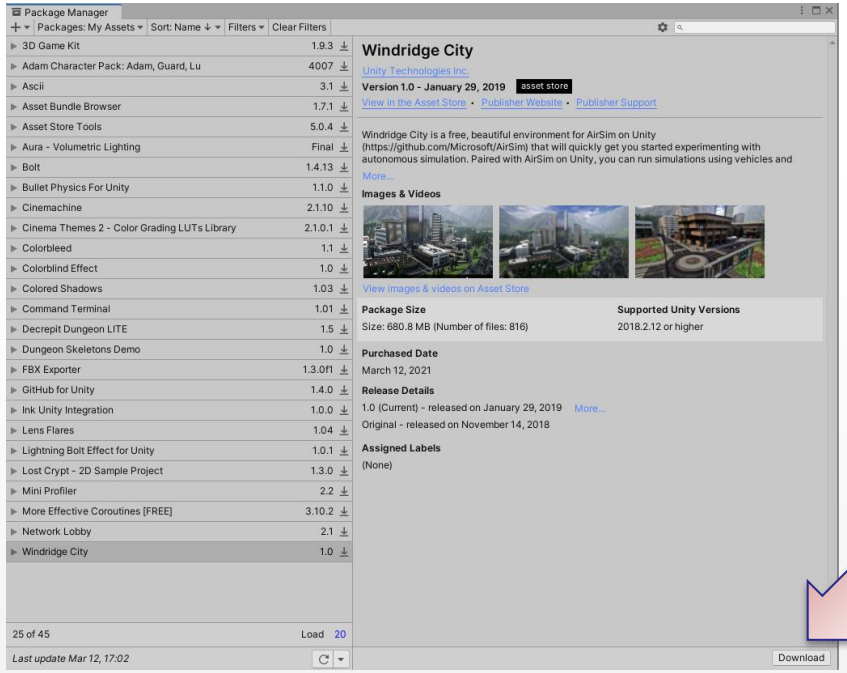


2. Add

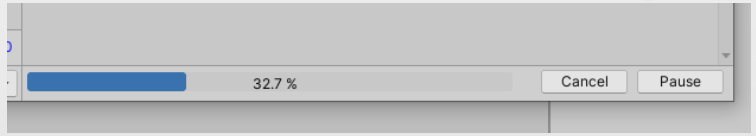
Getting Assets From the Store



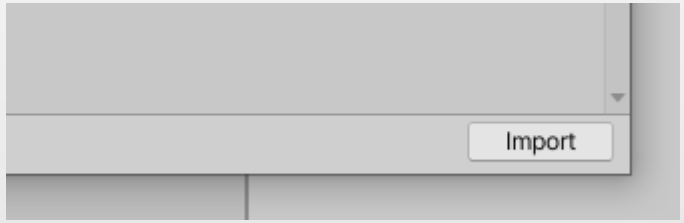
3. Open in Unity



4. Download

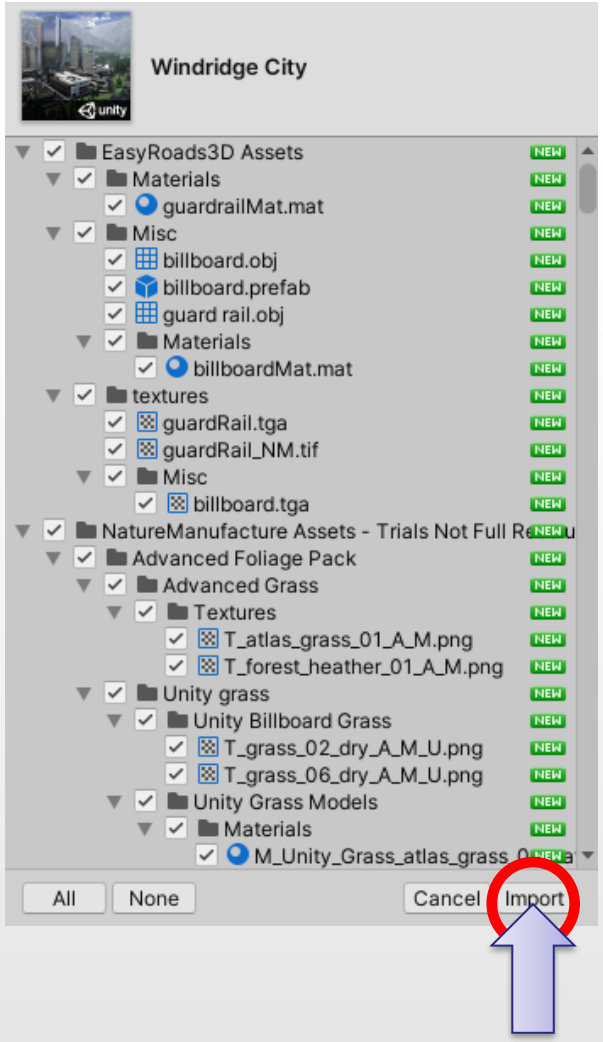


5. Wait

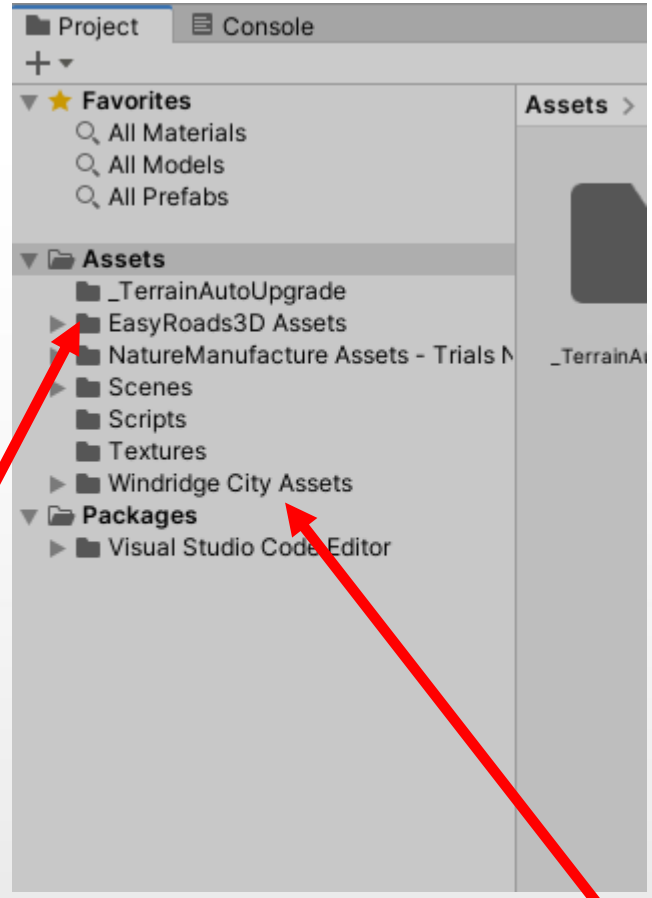


6. Ready to import

Importing Assets



May take a LONG time!
Be patient

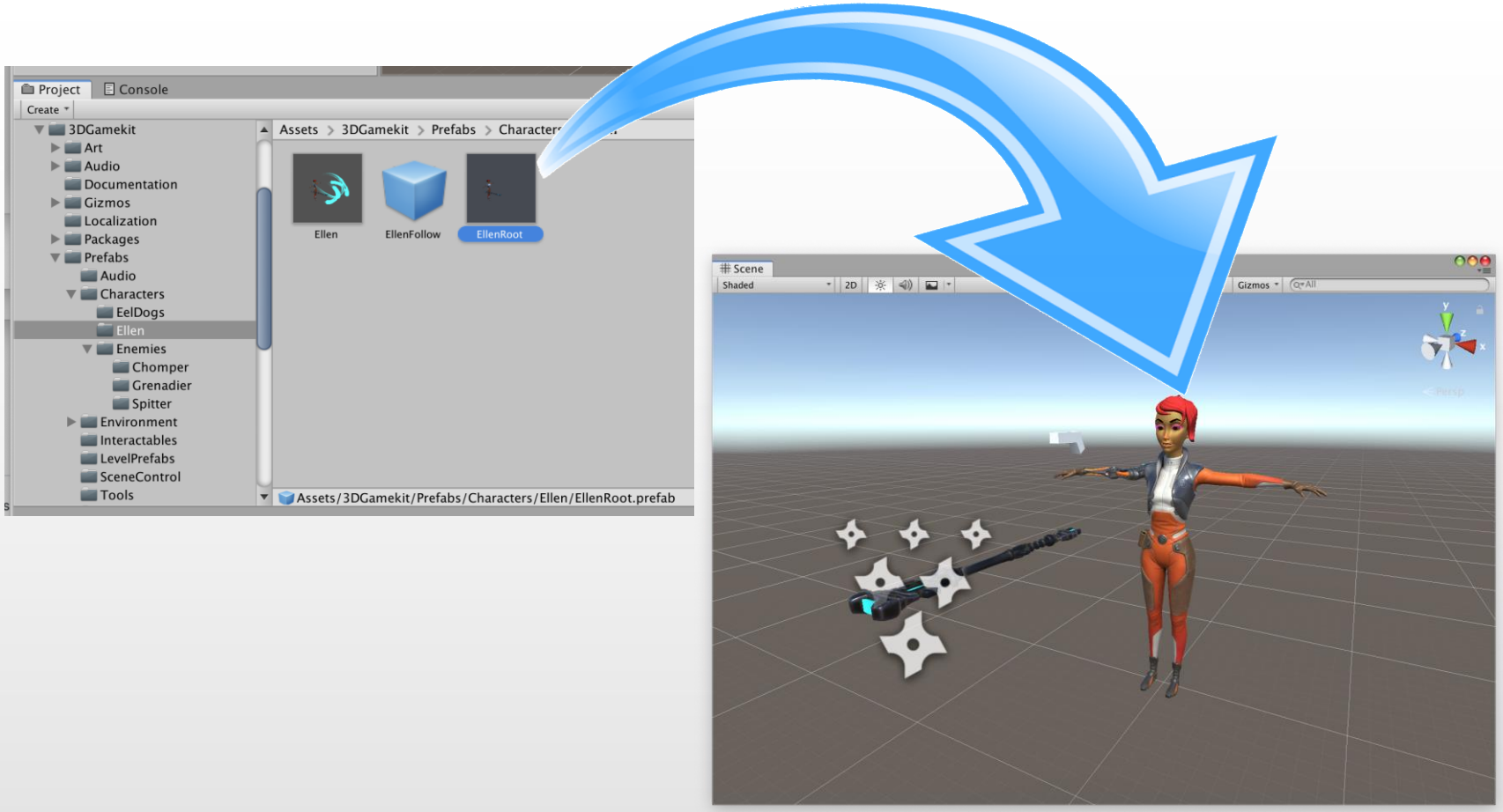


Auto-installed
Dependences

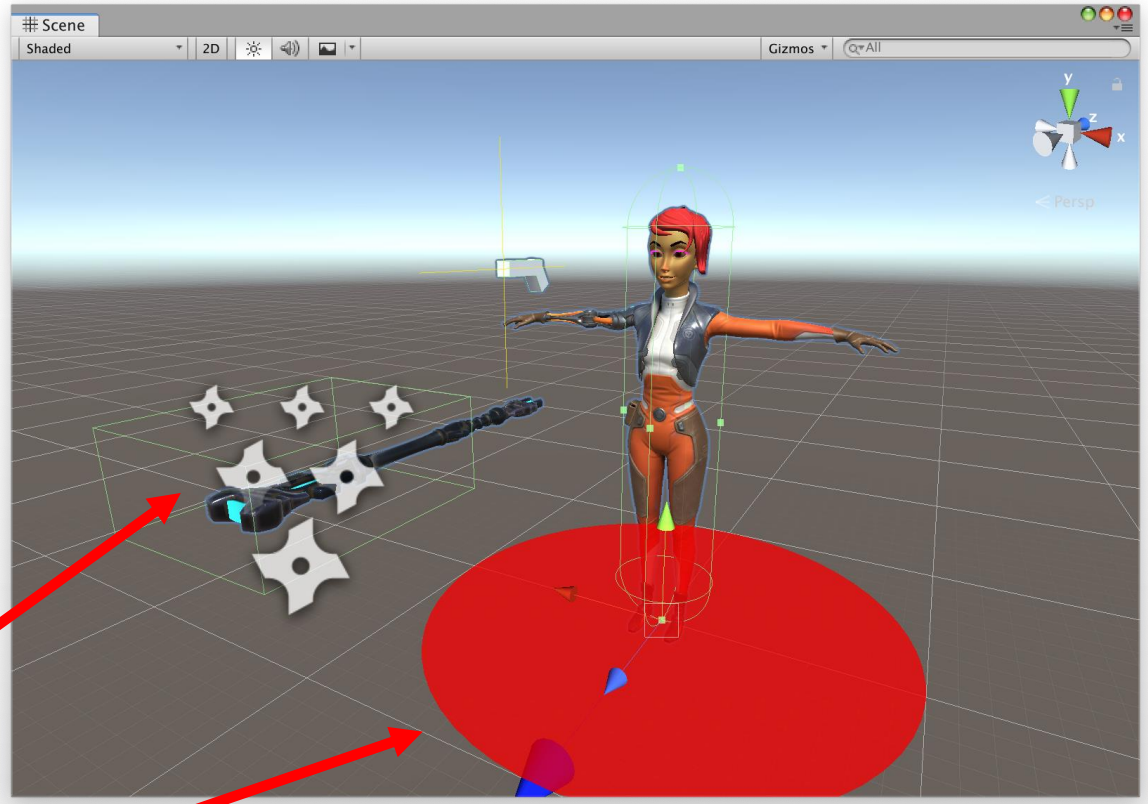
Out of ”Packages”!

Putting Assets in the Scene

Just Drag and drop from the asset database to the scene



What ...

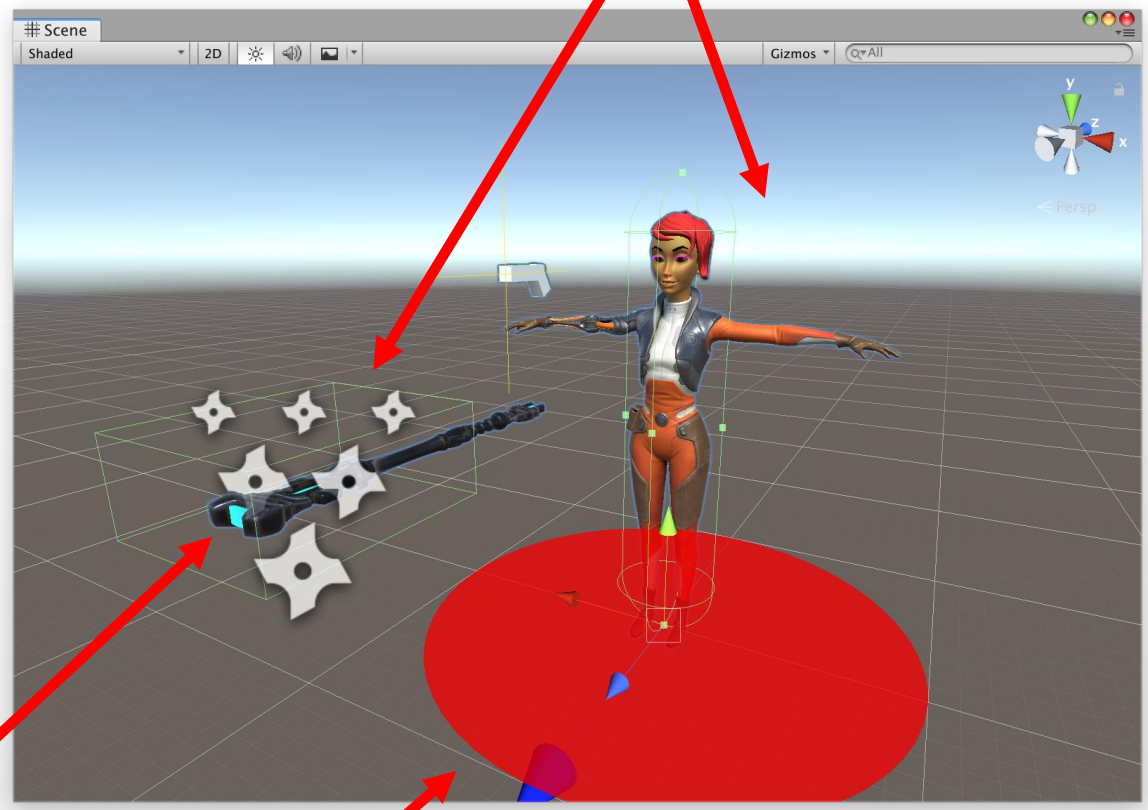


The hell are those?

What ...

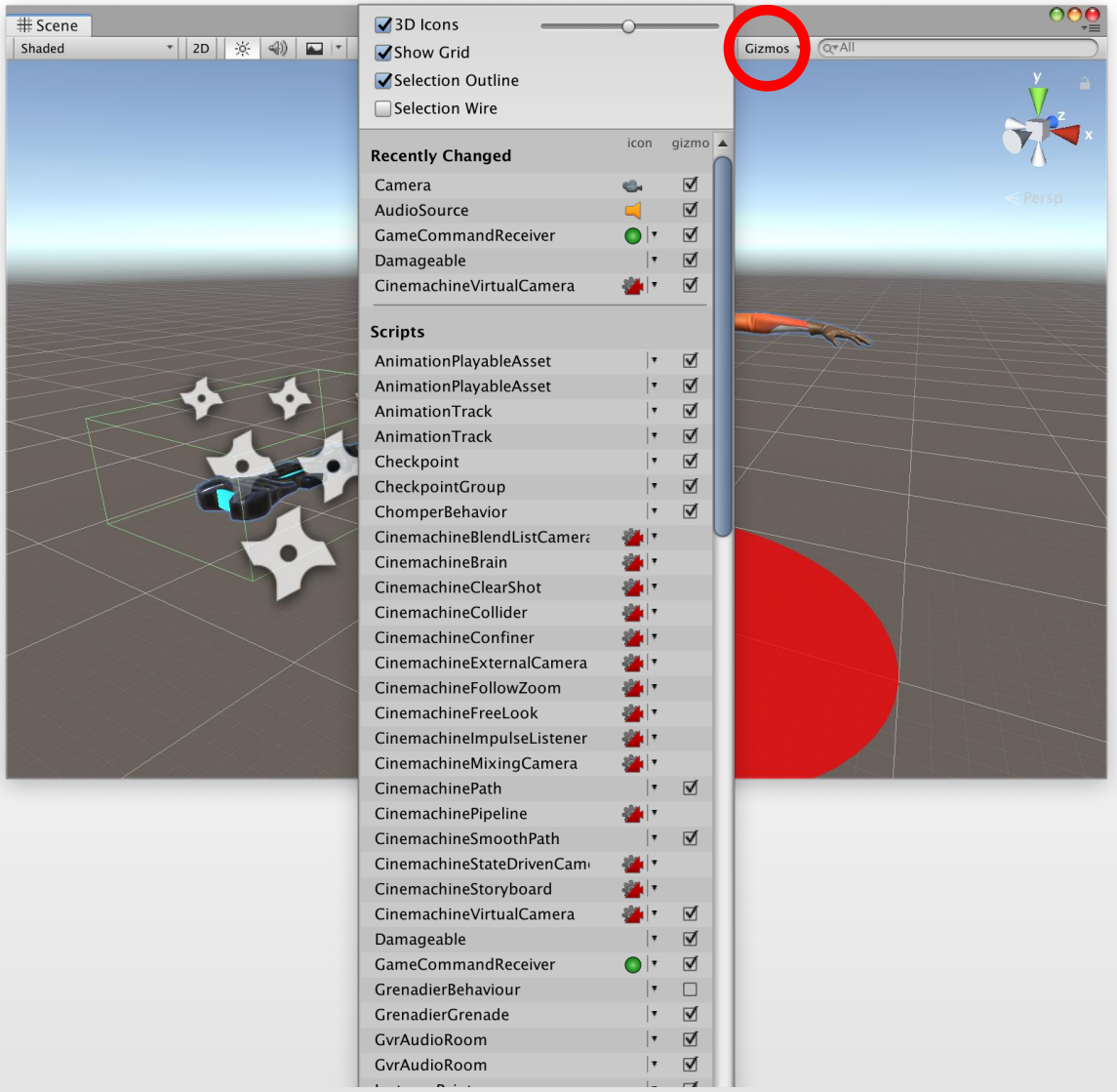
- Environmental function such as lights, the camera, or particle generators have 3D icons in the editor to track them down (you will not see them in the game window)

- When you select something, highlight (such as collision boxes) and visual debugging information (gizmos) will be activated



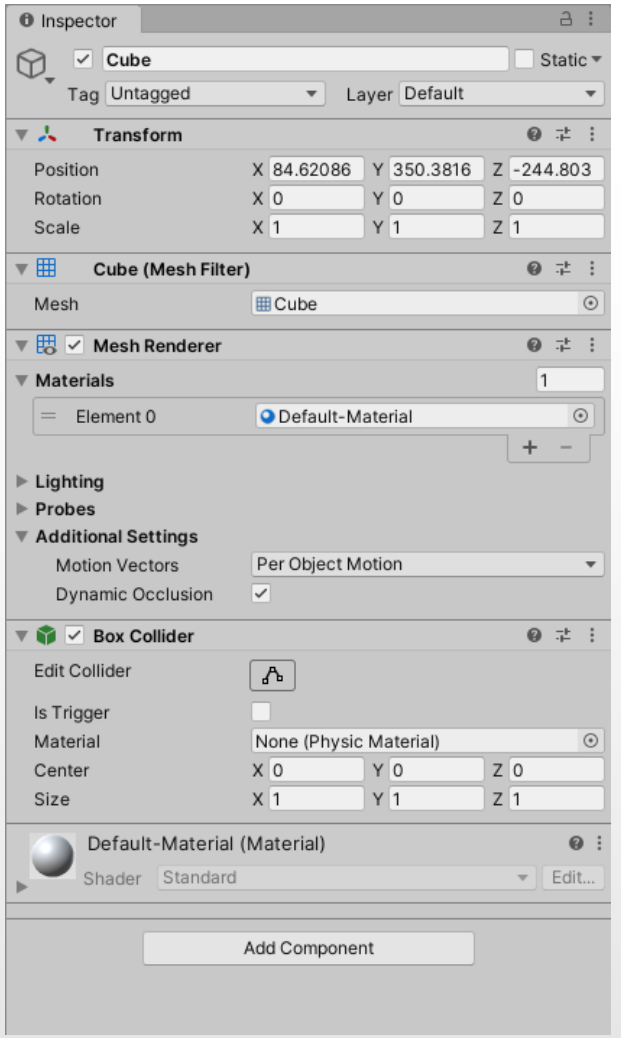
What ...

- Both environmental icons and gizmos can be deactivated using the ”Gizmos” menu in the scene panel



The Inspector Panel

- The inspector will give you all possible details about an object in the scene (from now on “gameobject”).
- Each gameobject **can have** a variable number of elements (components) inside
- **All** gameobjects **have** a **transform** component
  - And you cannot strip it off

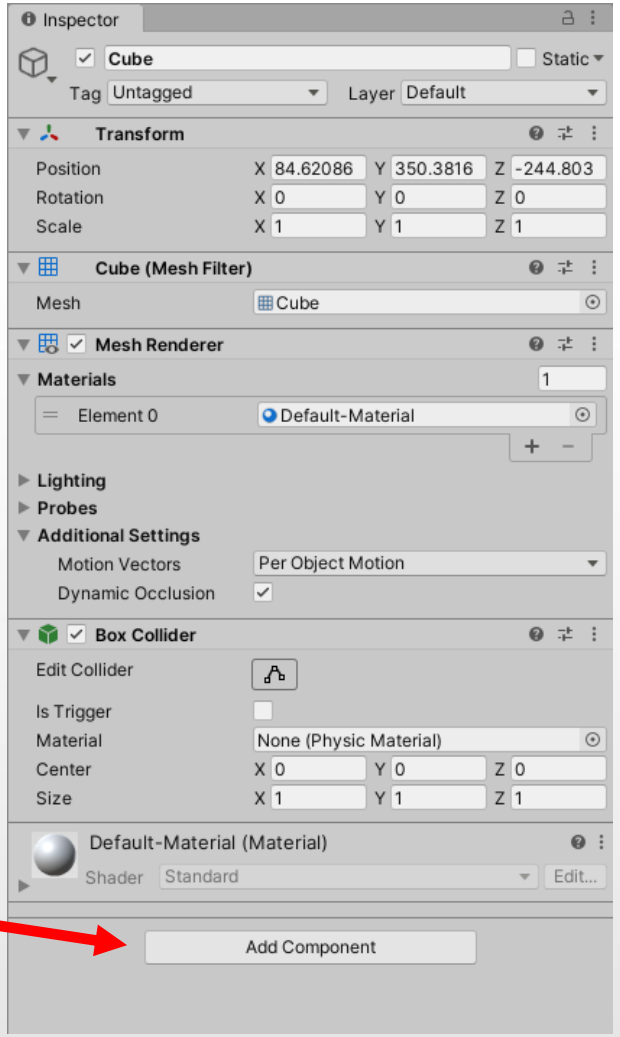


Yes! That is the damn 4x4 matrix you (did not) care about in the computer graphics class!

The Inspector Panel

- Components are add-ons that can be linked to an object in order to to add functionalities

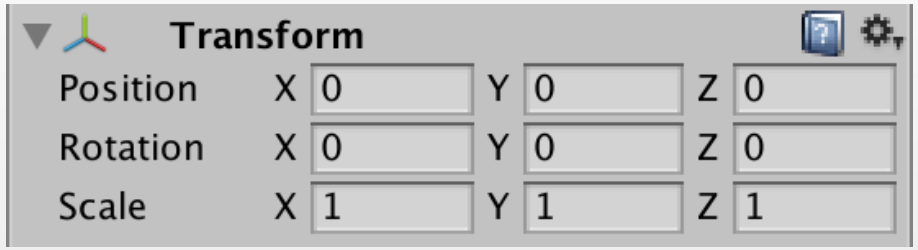
- Components can
  - Give a geometric position, rotation, and scale factor
  - Give a shape to the object
  - Specify ”how to draw” the object
  - Declare and area (with shape) for collision detection



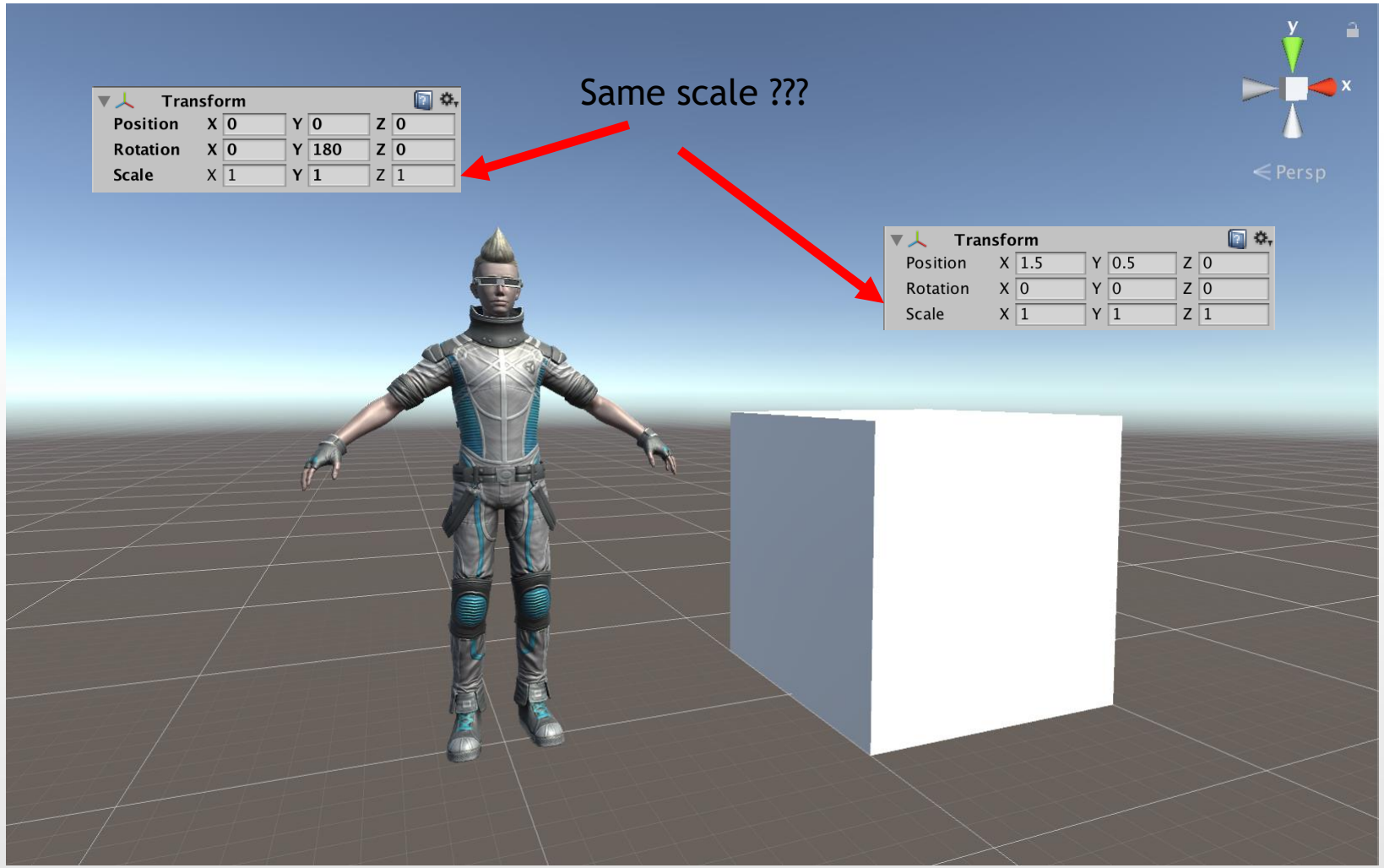
Click here and see the full list!

Beware of the Geometry

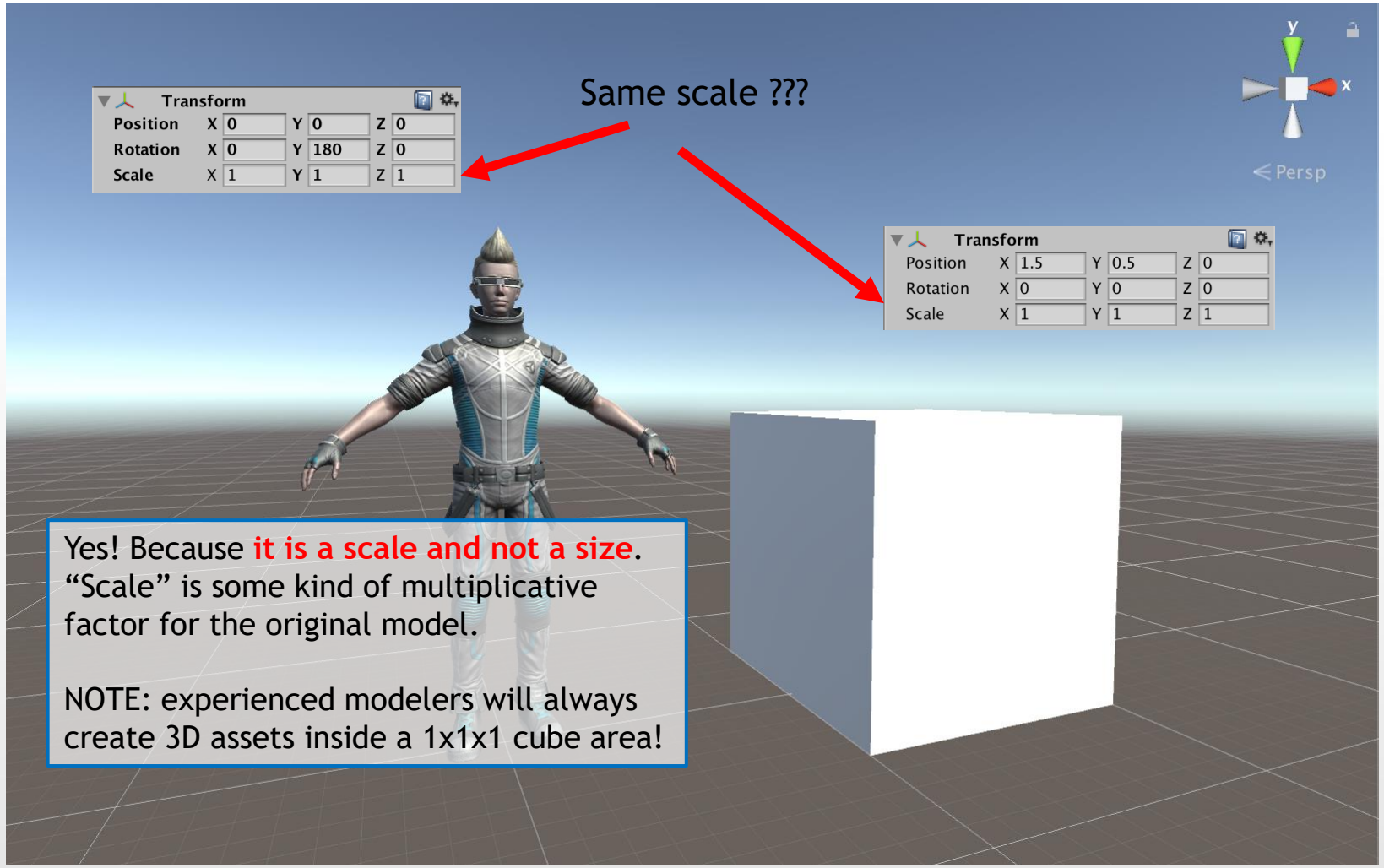
- The transform component provides 3D vectors for:
  - Displacement
  - Rotation
  - Scale
- for **everything** linked to the gameobject
- NO! Not just its shape!



The Horror Scale



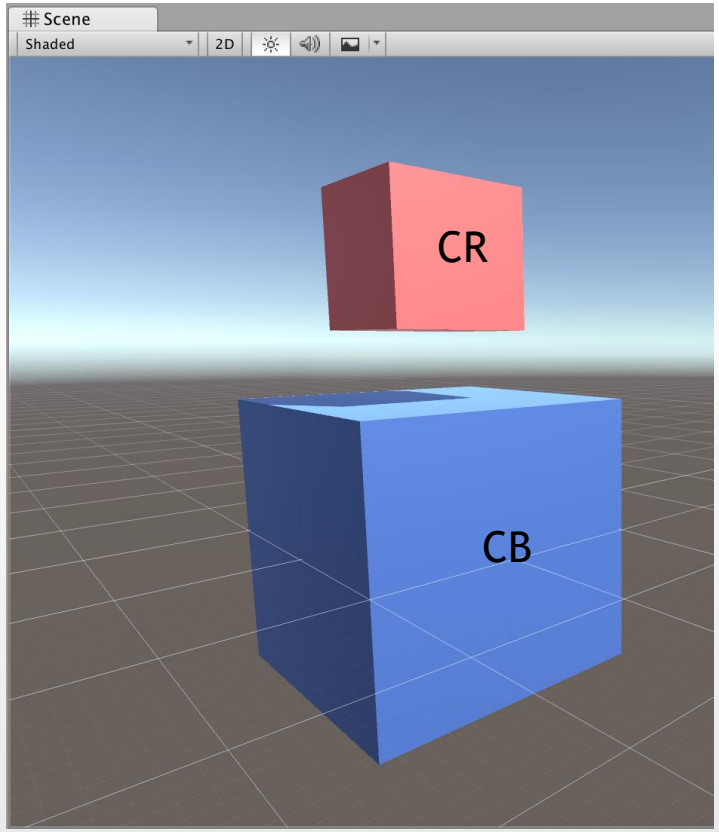
The Horror Scale

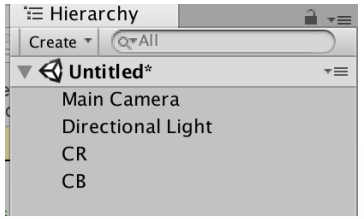


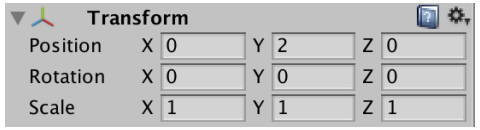
Combining Objects

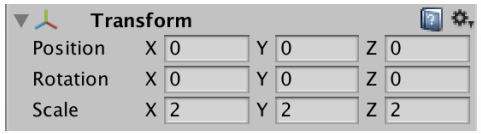
- Combining (parenting) objects will create a hierarchy tree
  - As simple as that
- From a geometric point of view, **their transforms will be applied in sequence from leaf to root**
- When parenting (and unlinking) objects the child will not change shape, position, and size; but its transform will be adjusted accordingly

Parented Objects



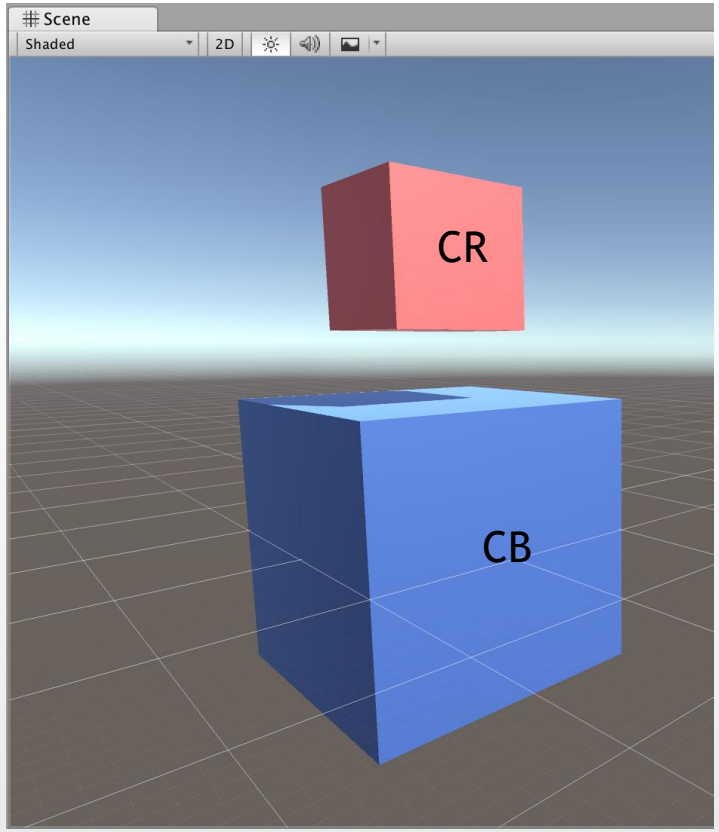


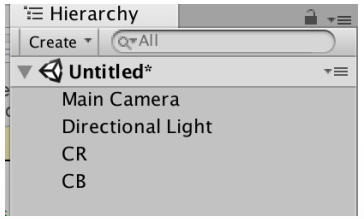


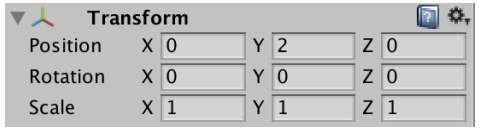


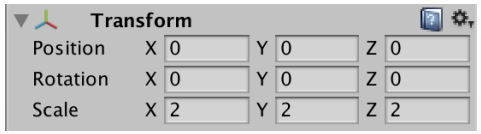
Before parenting

Parented Objects

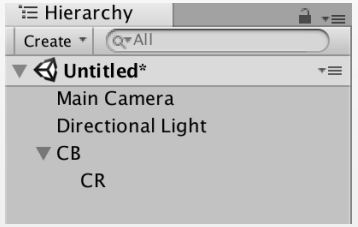


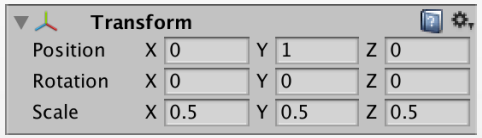


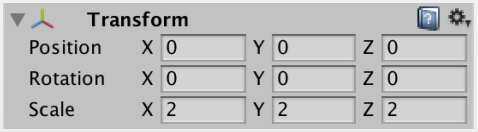




Before parenting

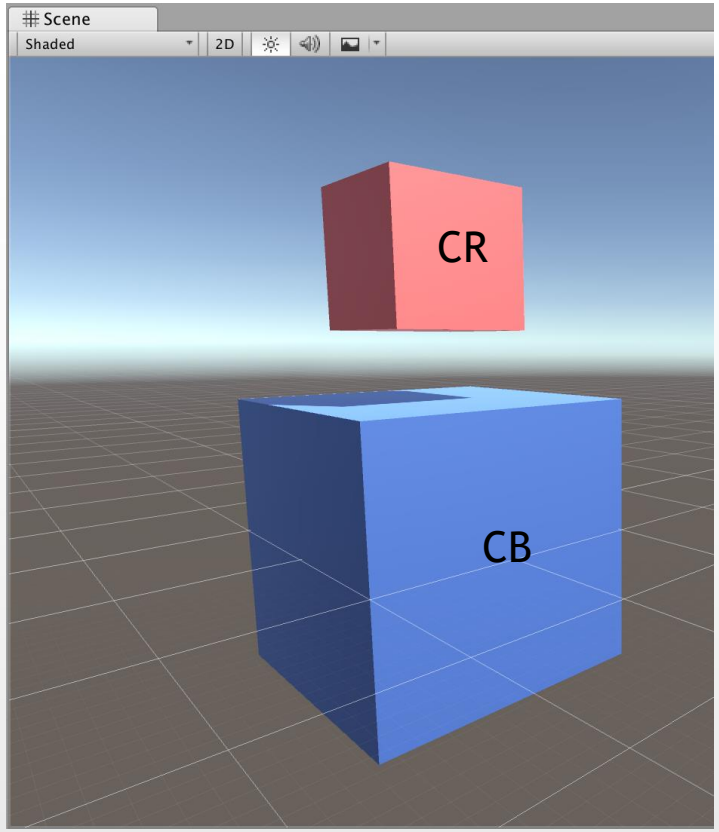


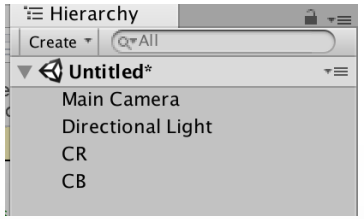


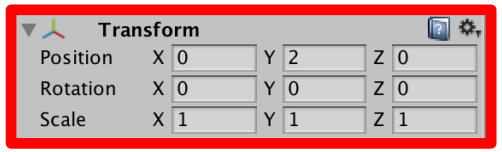


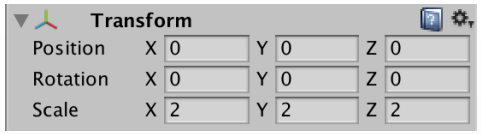
After parenting

Parented Objects

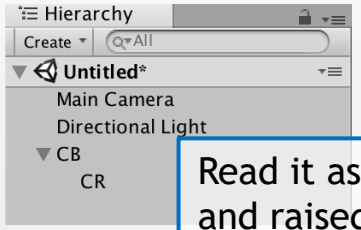


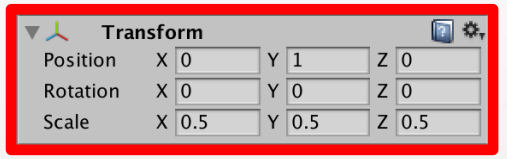




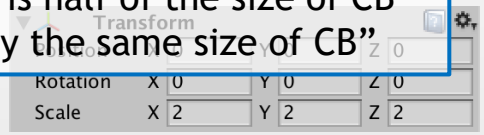


Before parenting





Read it as: “CR is half of the size of CB and raised up by the same size of CB”



After parenting

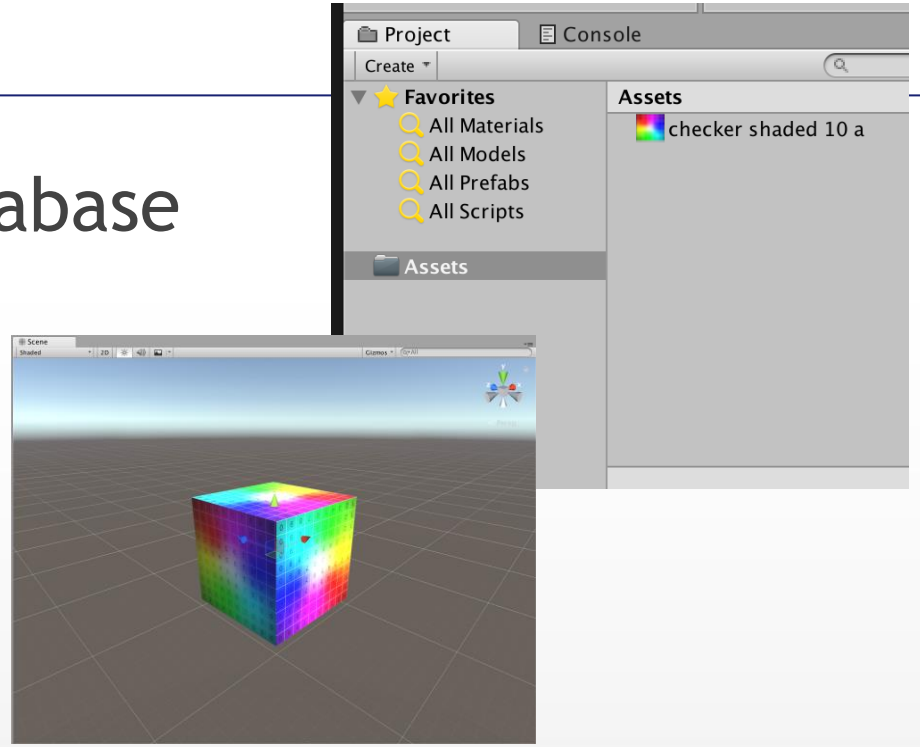
Transforms in Objects Hierarchy

- Root object
  - Positions: position in world coordinates
  - Rotation: orientation w.r.t. world axis
  - Scale: pure scale factor
- Child object
  - Position: displacement using parent as unit
    - <0, 1, 0> is “displace ‘above’ by the size of my parent”
  - Rotation: orientation w.r.t. parent’s axis
    - And this is going to redefine the concept of “above” for position
  - Scale: scale with respect to parent size
    - <1, 1, 1> is “the same size as my parent/container”

This is VERY important to know when scripting your objects!

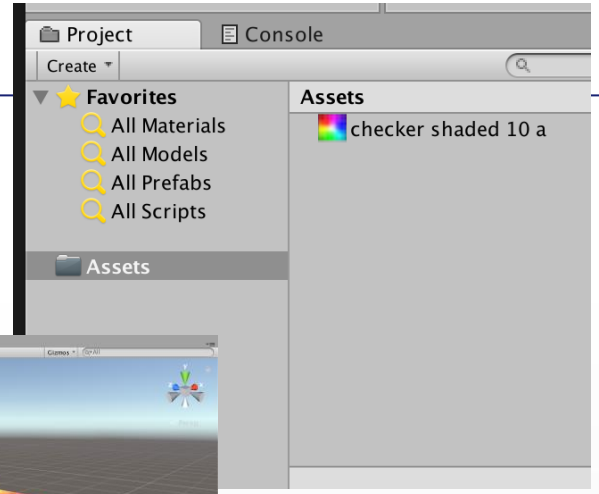
Adding Colors and Texture

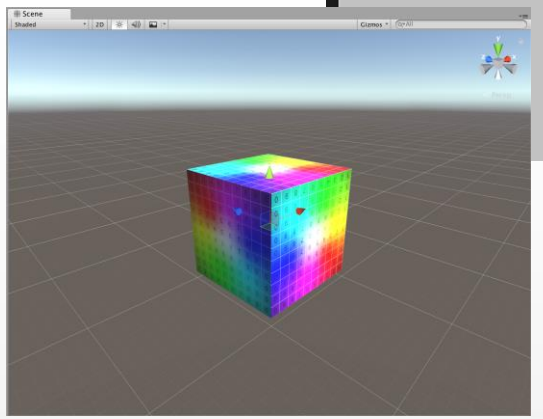
- Put a texture in your asset database
- Drag the texture on an object
- … and that's it!



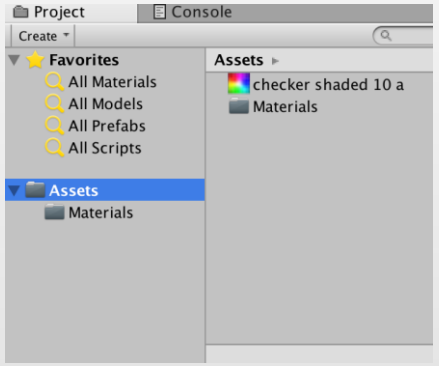
Adding Colors and Texture

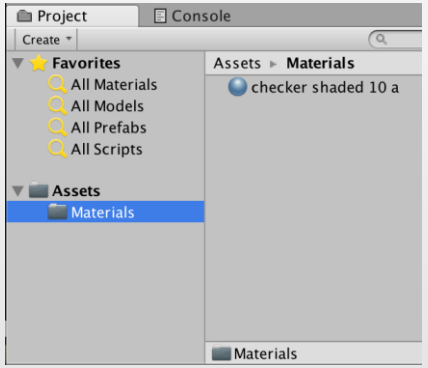
- Put a texture in your asset database
- Drag the texture on an object
- … and that's it!





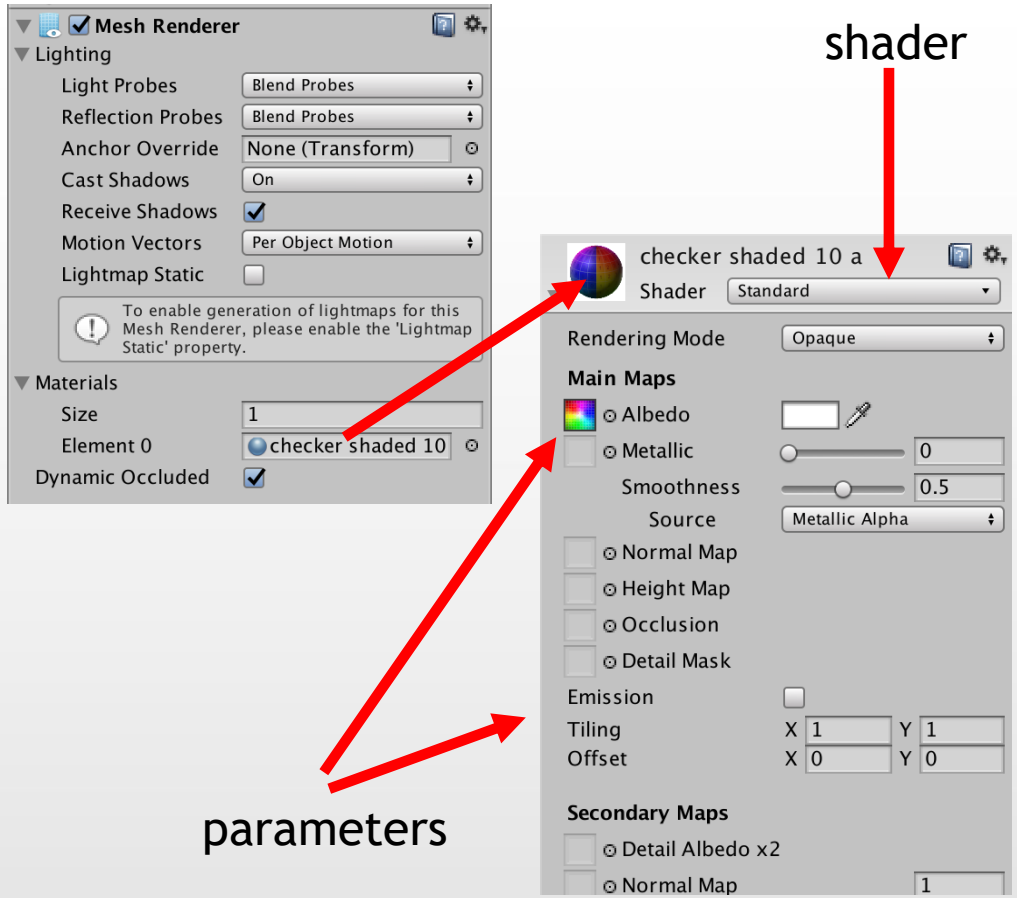
- WAIT !!!
  What the hell is that?





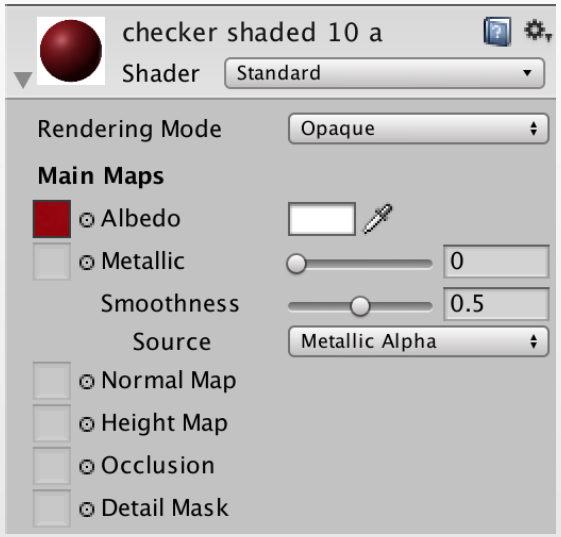
Unity Perspective for Colors and Textures

- A gameobject which requires rendering **must** have a mesh renderer component
- The mesh rendered component holds a reference to a material asset
- A material is a container for parameters that a shader will use to draw the gameobject
- A shader is a piece of code for the GPU
- The texture we applied is merely a parameter (albedo) for the shader

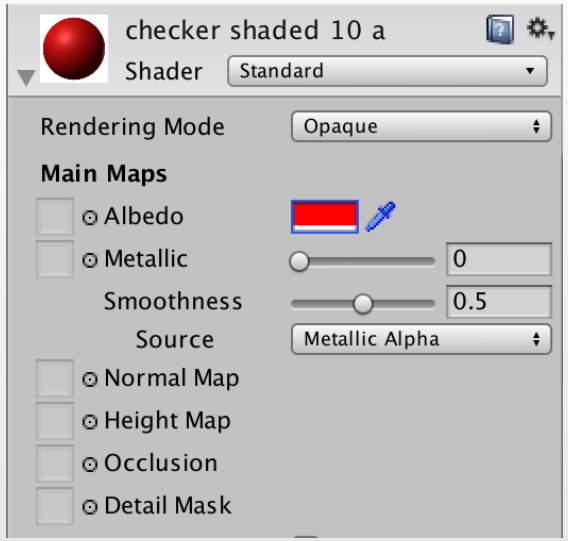


A Color is NOT a Texture

- A texture is like a gift wrap around the gameobject
  - Albedo box parameter
    - No albedo means “white wrap”
- A color is a tint applied to the wrap
  - Color box parameter

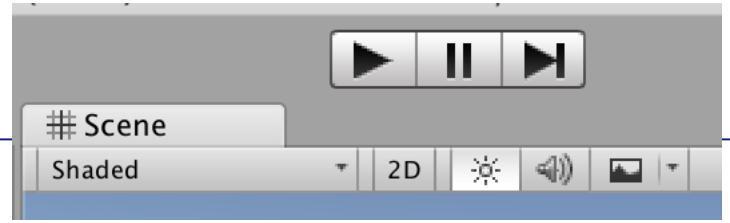


A red textured gameobject

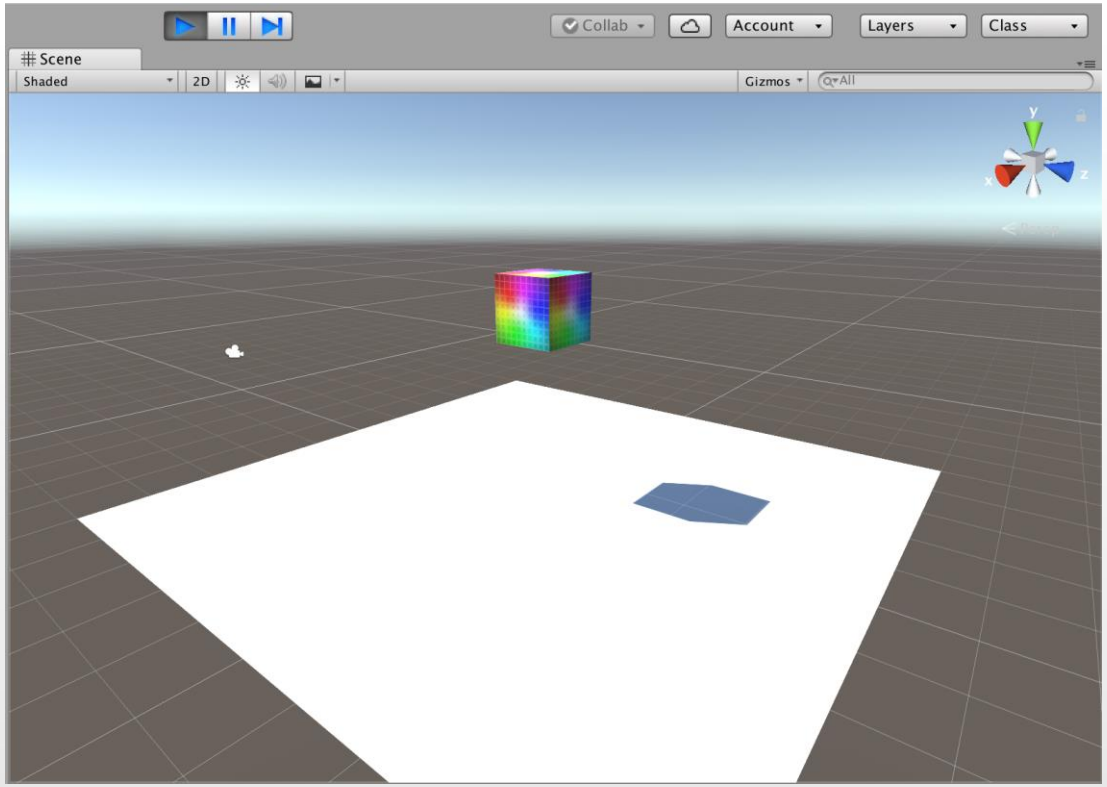


A red tinted gameobject

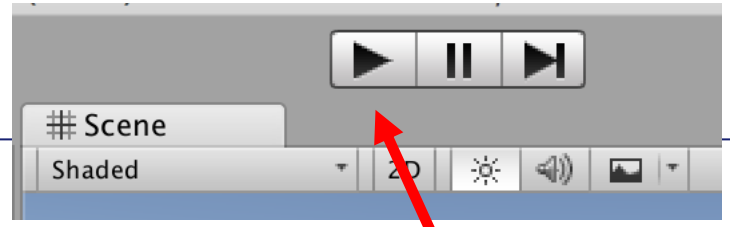
Ready, Set, GO!



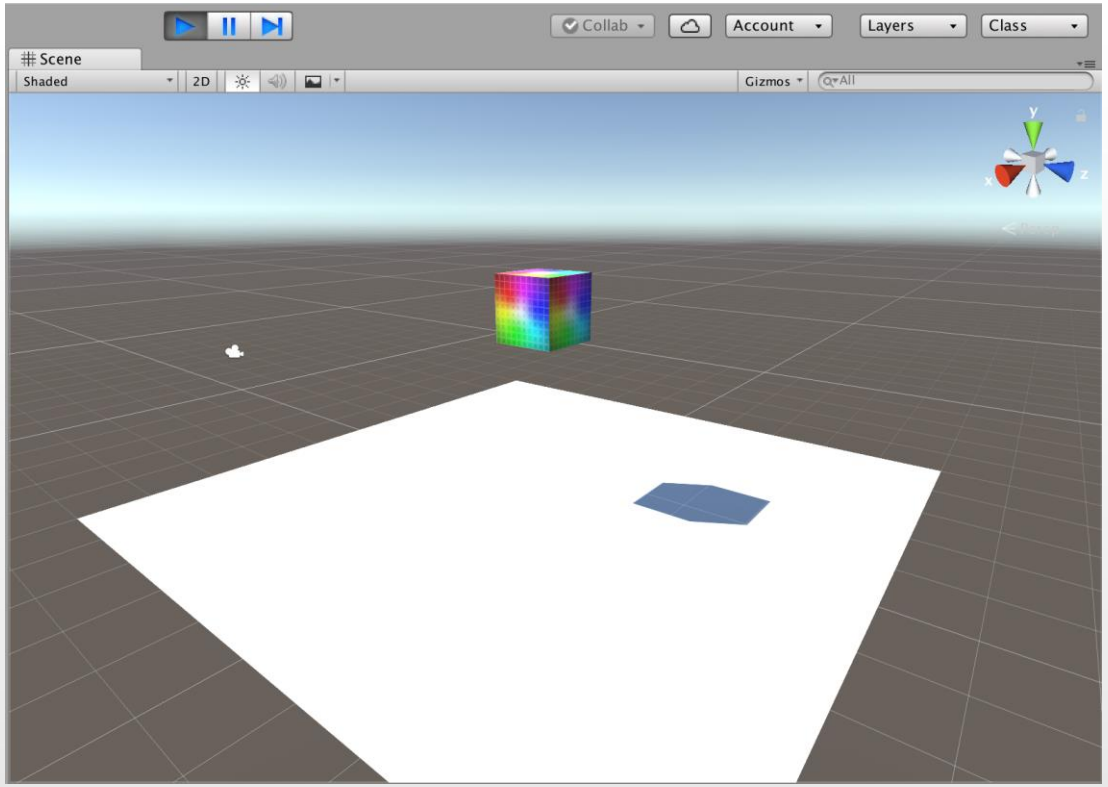
Once you are happy with the scene,
you can start your game and see what happens



Ready, Set, GO!

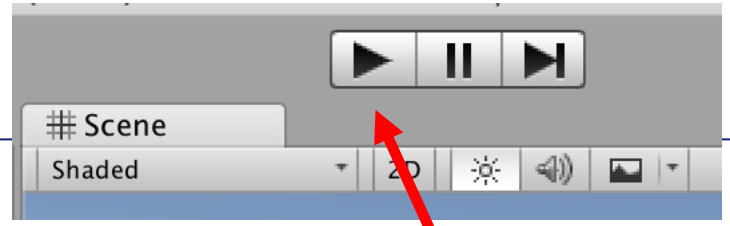


Just push the button on top!

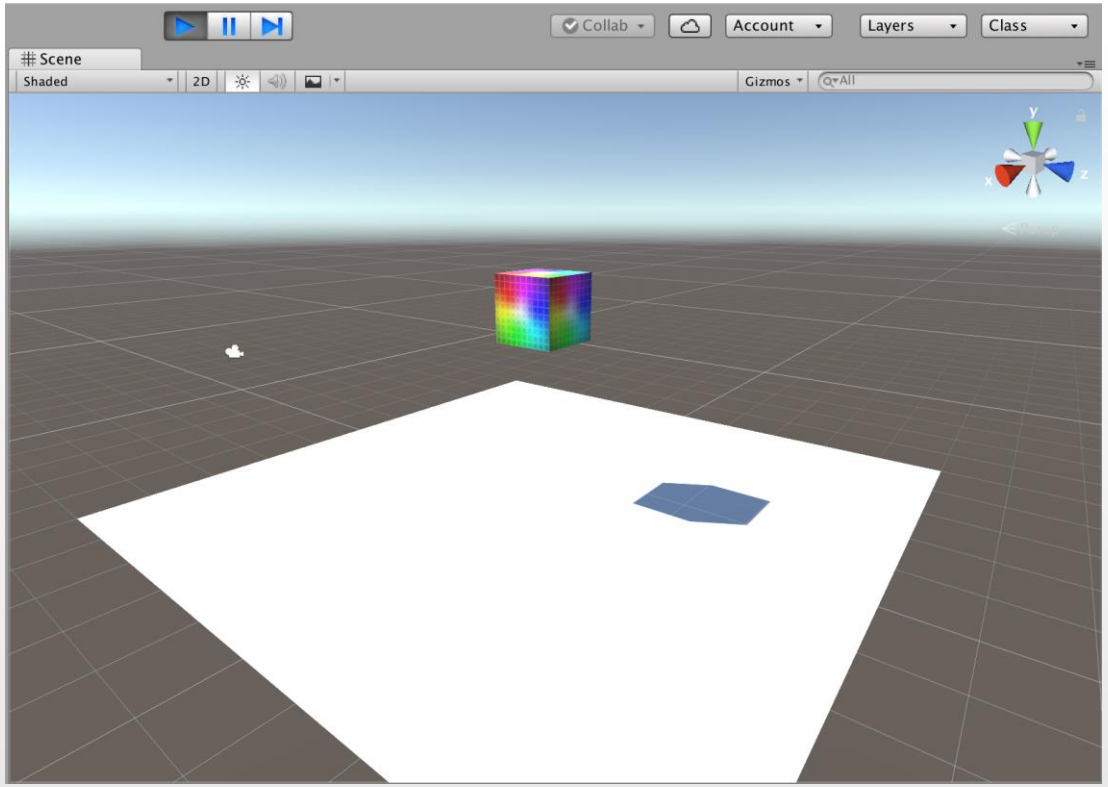


GO!

Ready, Set, GO!



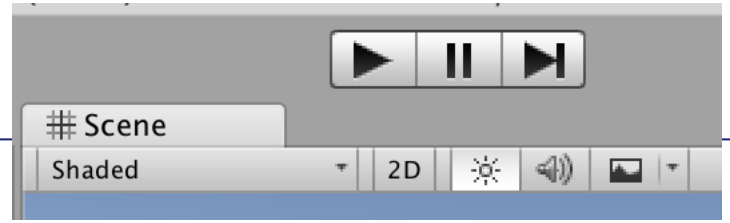
but ...



GO!

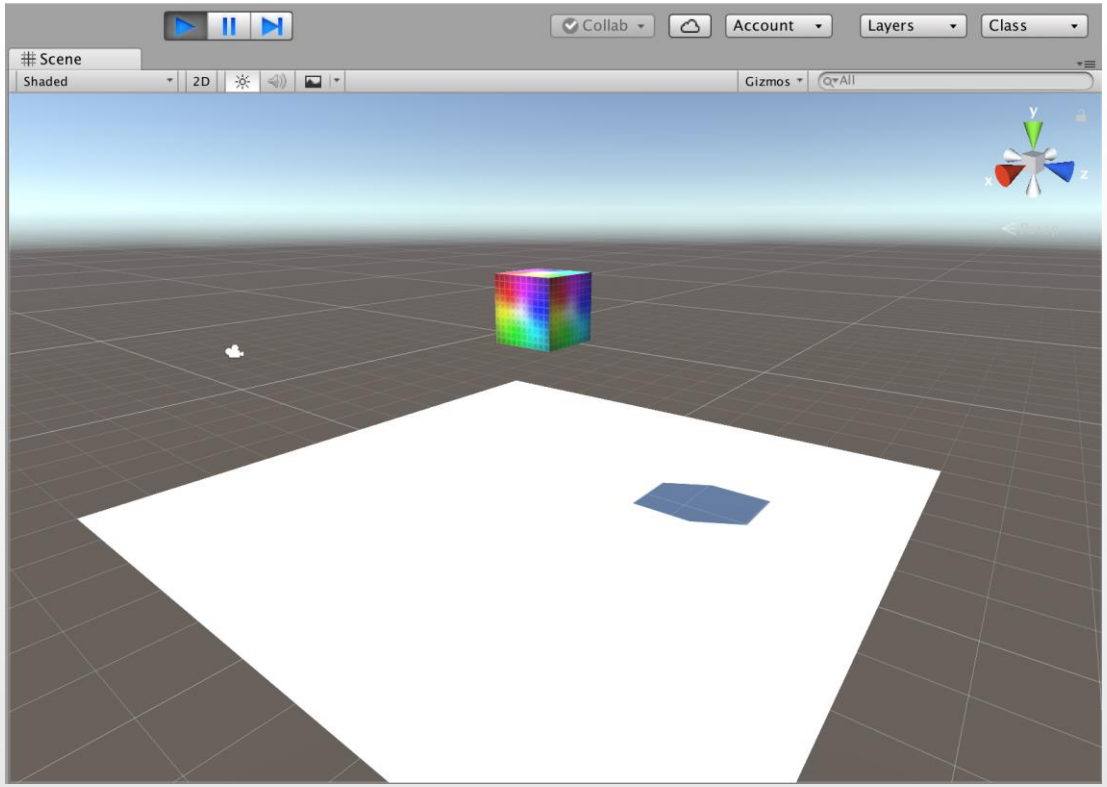
Nothing happens!

Ready, Set, GO!



The system is stuck because there are no evolution rules.

The runtime has no directions about how to transform the environment over time



Nothing happens!

Back to the Architecture

Asset management

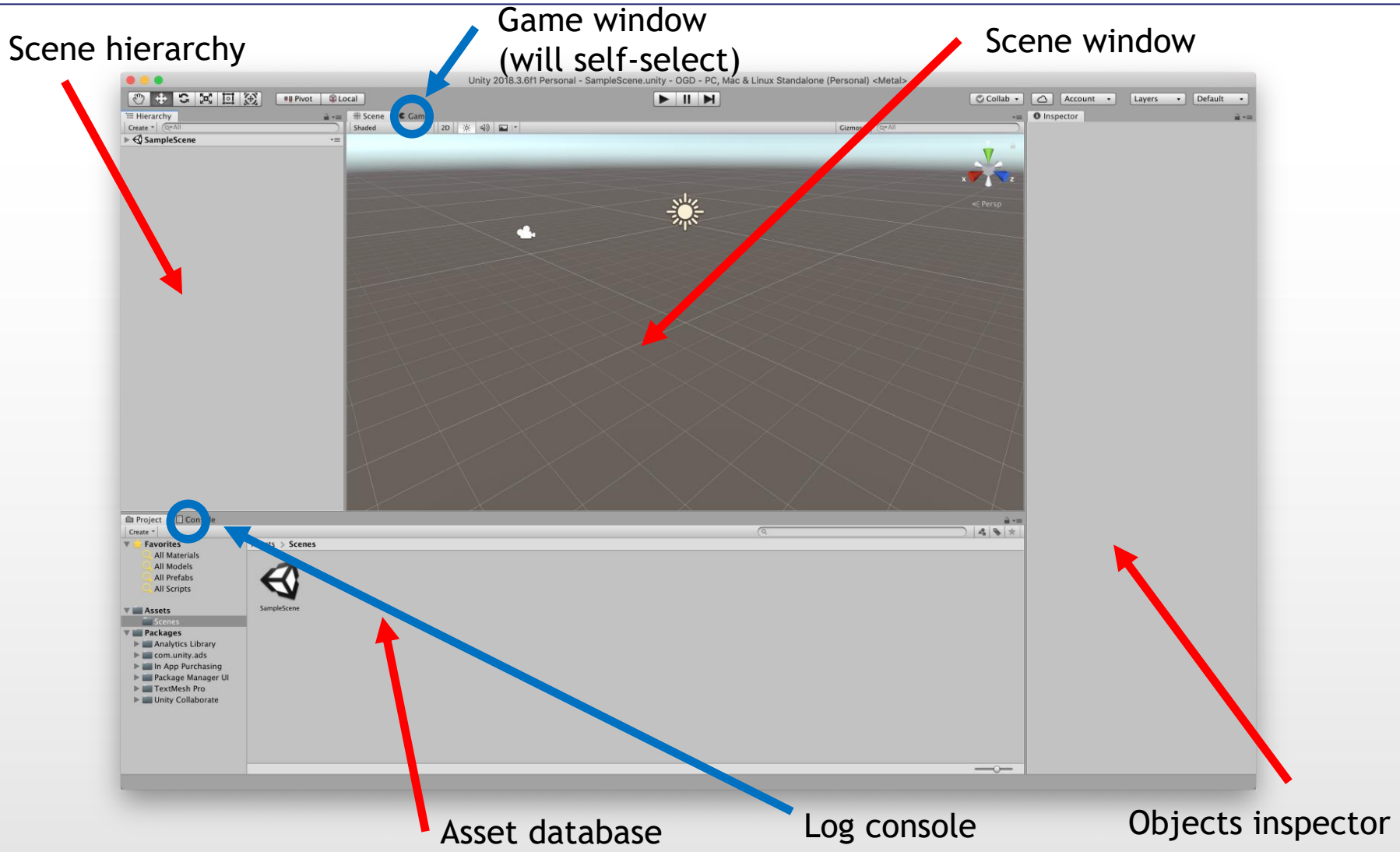
Scene editing

Source code

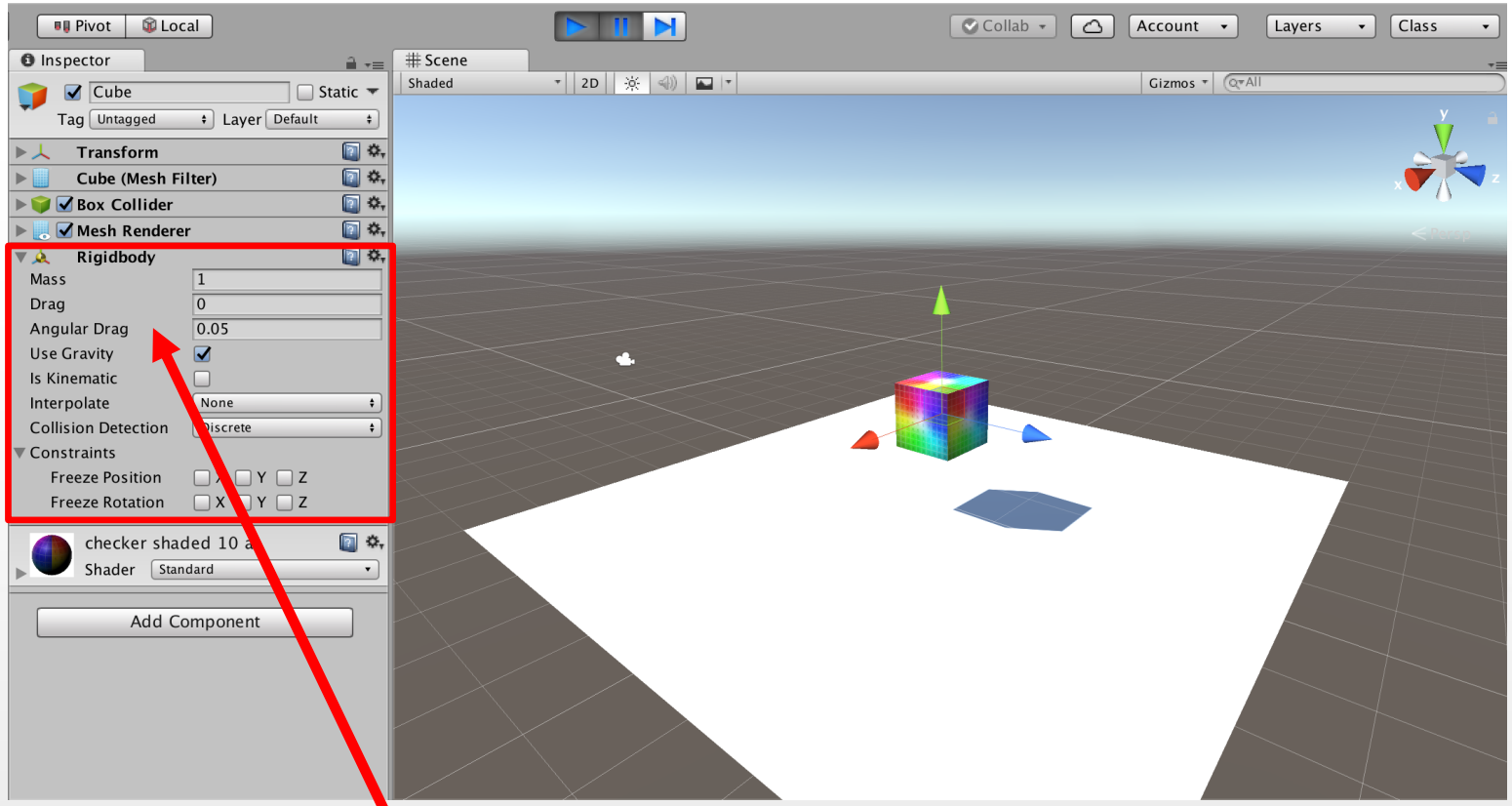
Components can be seen as evolution Rules!

They can be found built-in in the runtime (for standard evolution) or defined by the user via code

Unity Interface Anatomy (Update)



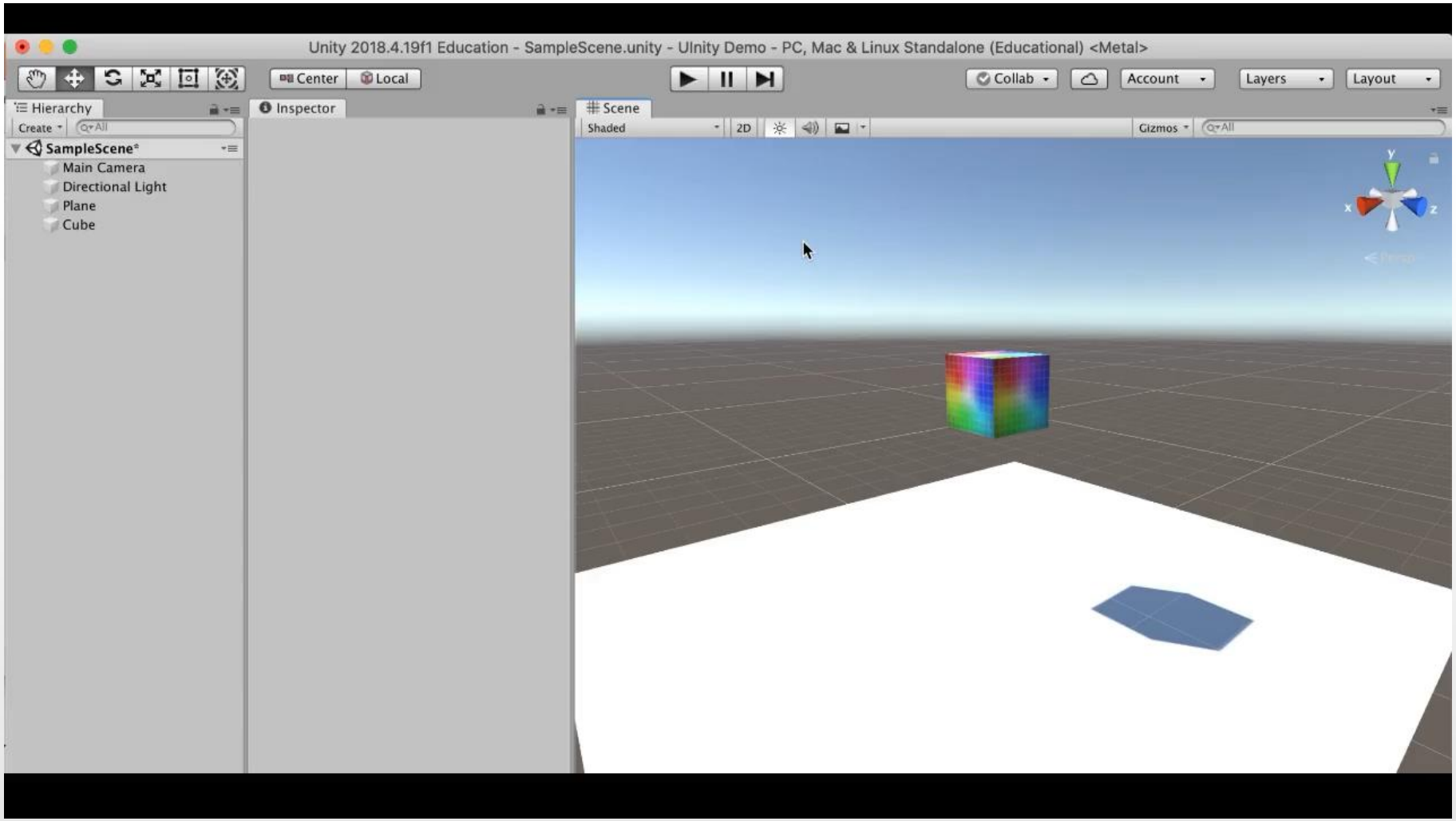
Make it Evolve (with a Physics Component)



Here is the rule!

RigidBody is a component that can define an evolution through the implementation of physics laws. In this example, falling

And the Exciting Result is …



Defining Your Own ~~Components~~ Evolution

UK Rulez!

- A component is a class extending *MonoBehaviour*
- This class, once compiled, becomes an asset that can be added to any gameobject
  - A panel will be shown in the inspector through introspection
  - Public fields will become the component UI elements
  - Methods will be invoked when needed
- No. There is no “main” function here!
- You can do any operation and access any data, component, and gameobject as you do with the unity GUI
  - ... and much more!

An Empty Script

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

public class Rotation : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

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

    }
}
```

- Start() will be called when object is first placed in the scene
  (even if you cannot see it!)
- Update() will be called at every frame

Not the same “frames” as on the screen!

A frame, in this case is the time required by Unity to update all gameobects in the scene. That is to say, calling all the update() functions and restart the cycle

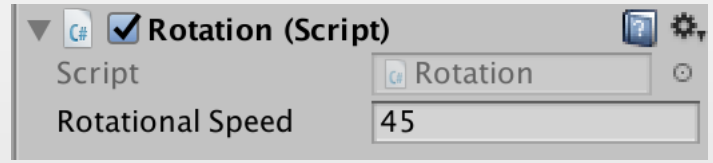
Simple Rotational Script

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

public class Rotation : MonoBehaviour {

    public float rotationalSpeed = 45f;

    void Start () {
        ; // nothing to do
    }

    void Update () {
        transform.Rotate (transform.up, rotationalSpeed * Time.deltaTime);
    }
}
```

Make rotation time-dependent and frame-independent

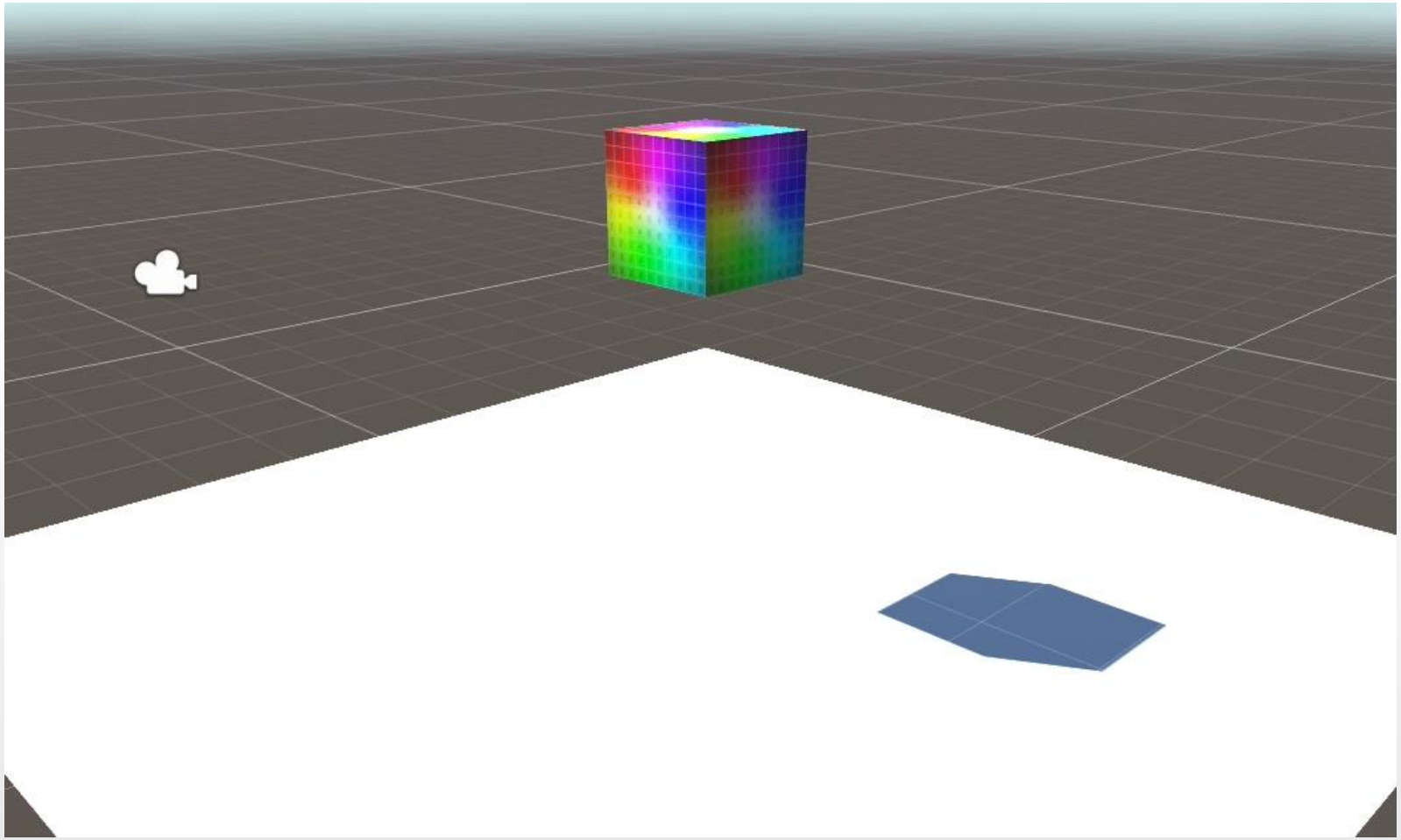
Time.deltaTime is the time elapsed since this Update() was last called

This is going to change if we interact with the interface



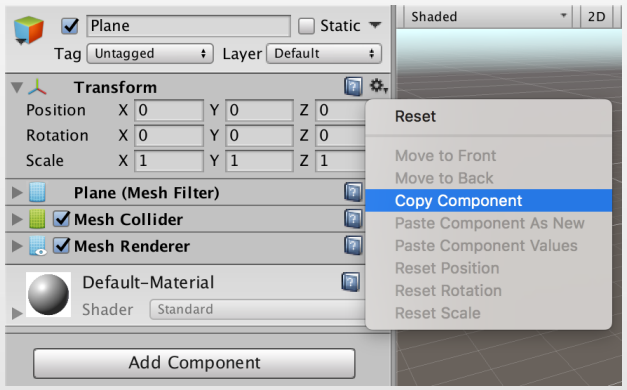
Once added to the gameobject we will see this in the inspector

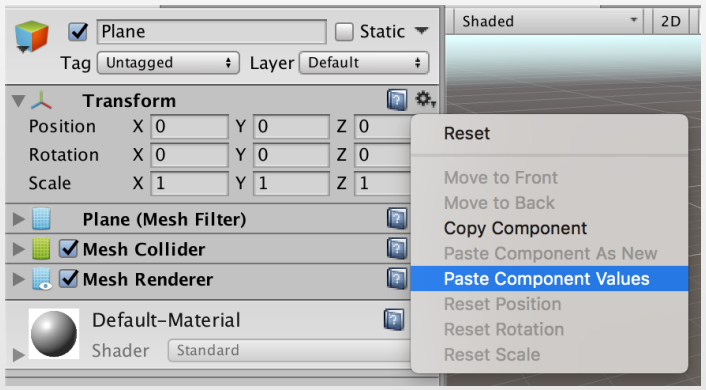
Application on a Cube



About Parameters

- They can be changed at runtime, yes!
  - Find the best values for your application while it is running!
- But when the game stops, everything is rolled back to the original values
  - Including your parameters
- To keep the values, copy a component during play and paste its values after you stop the game





Reaction to External Events

- Many methods, if implemented, will be called based on events:

1. Inside the simulation
   - Collisions
   - Mouse click and drag
   - Position change (as the result of a push)
   - Creation, destruction, hide, and show

2. In the editor GUI
   - Pause/restart
   - Panel draw
   - Parameter validation

3. On the network
   - Player connection or disconnection

- See the complete list here: https://docs.unity3d.com/ScriptReference/MonoBehaviour.html

A Bump Up on Collision

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

public class BumpUp : MonoBehaviour {

    public float push = 400;

    void OnCollisionEnter(Collision c) {
        Rigidbody rb = GetComponent<Rigidbody> ();
        rb.AddForce (transform.up * push);
    }
}
```

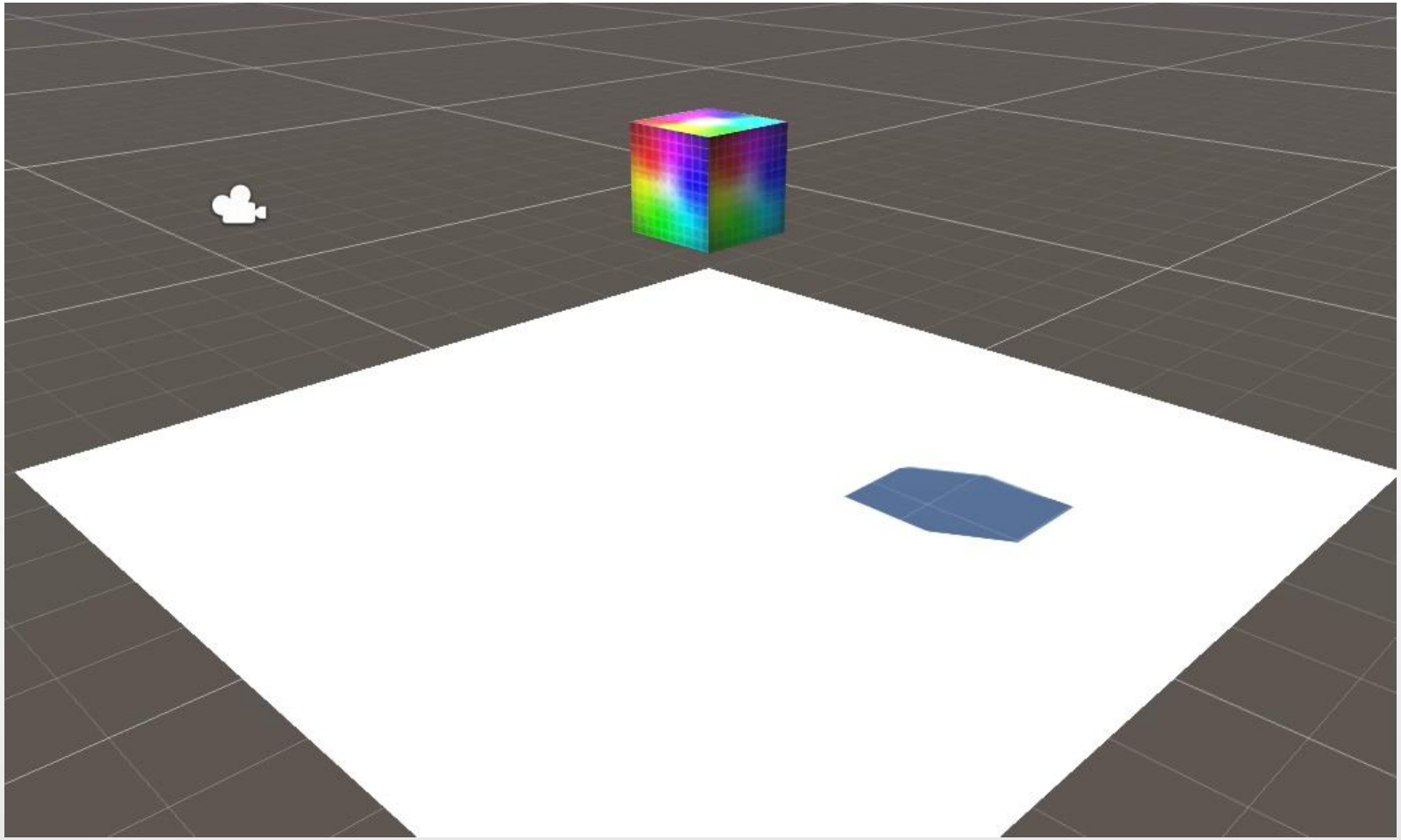
Get another component of type RigidBody from the same gameobject as this component

No worries, there will be only one

Will be called by the runtime when the gameobject collides with another gameobject

Use the RigidBody instance to push the gameobject “up” with respect to the orientation of the containing object

No One is Perfect

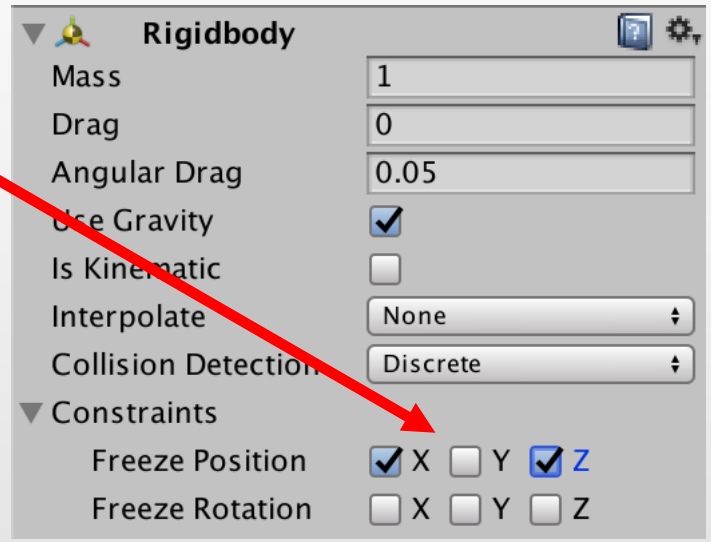


Physics has Limitations

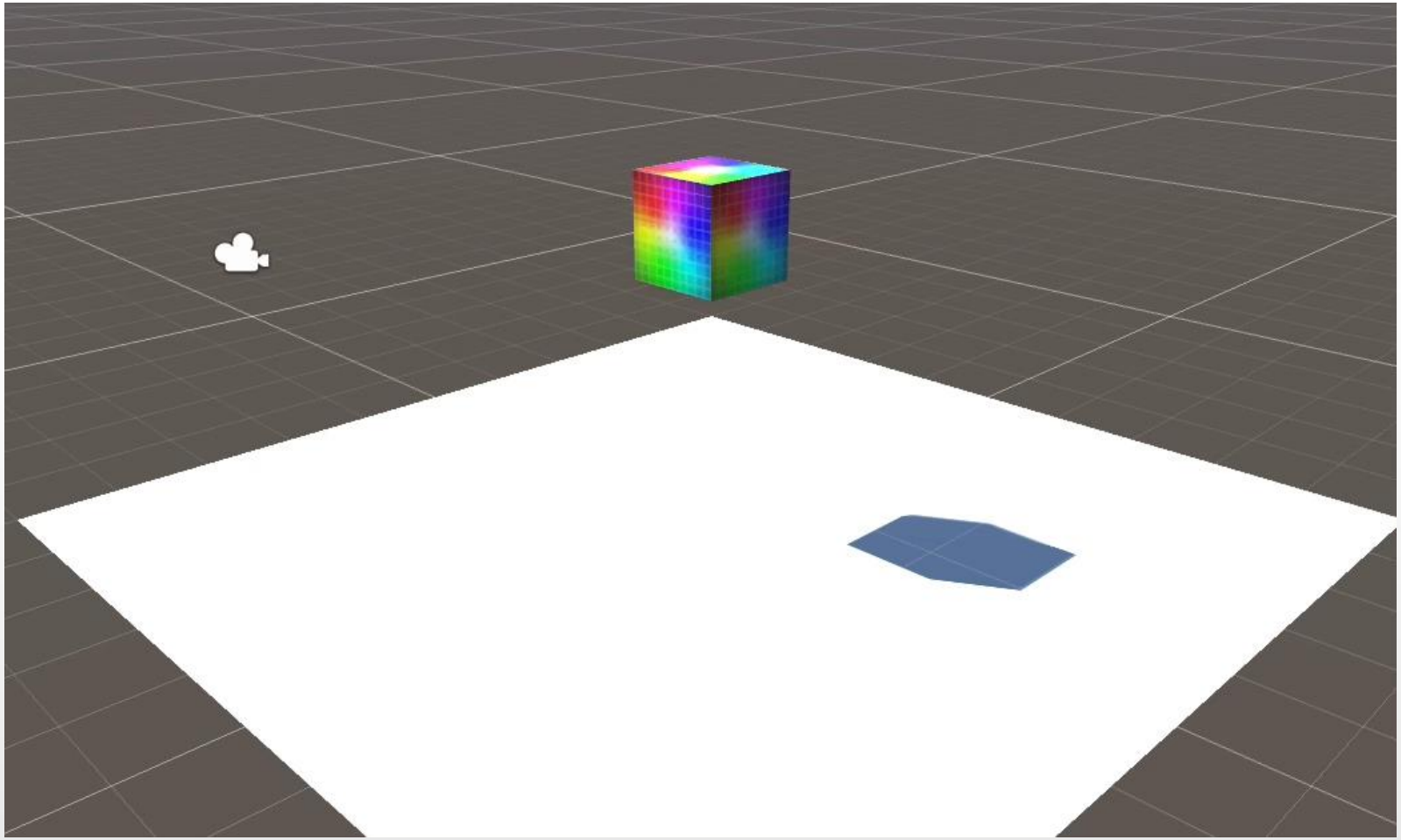
- Very fast/small objects collision will not be detected
- Calculations are not 100% precise for sake of performances

- But we have workarounds
  - Such as freezing the object position from inside the component

- … and also other physics engines such as Bullet Physics For Unity (which is free on the store)



But …



The Fixed Version

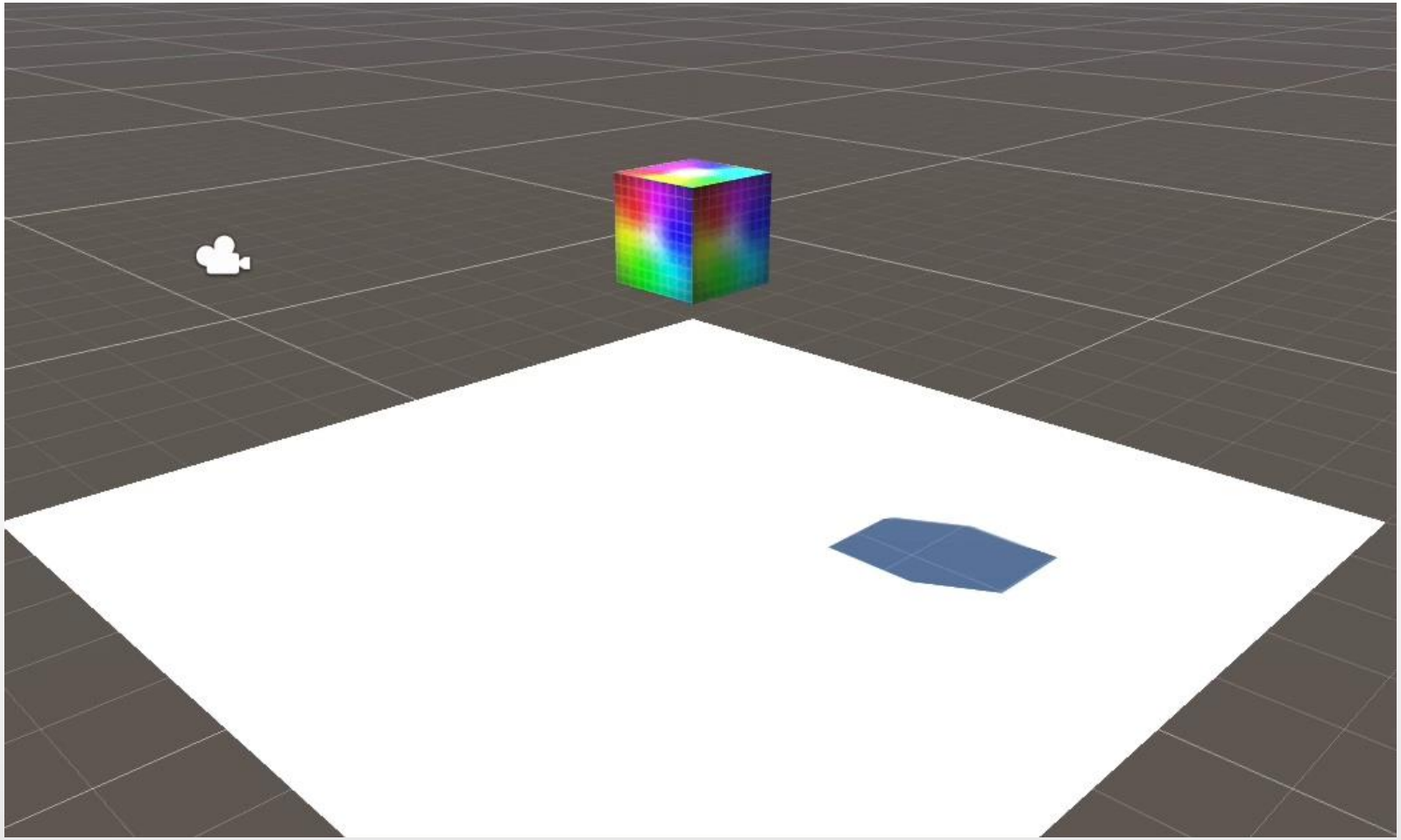
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BumpUp : MonoBehaviour {

    public float push = 400f;

    void OnCollisionEnter(Collision c) {
        Rigidbody rb = GetComponent<Rigidbody> ();
        rb.AddForce (c.gameObject.transform.up * push);
    }
}
```

Pushing “up” is not the right solution because if the object spins around you will be pushing sideway ... or down

Therefore, the cube get stuck

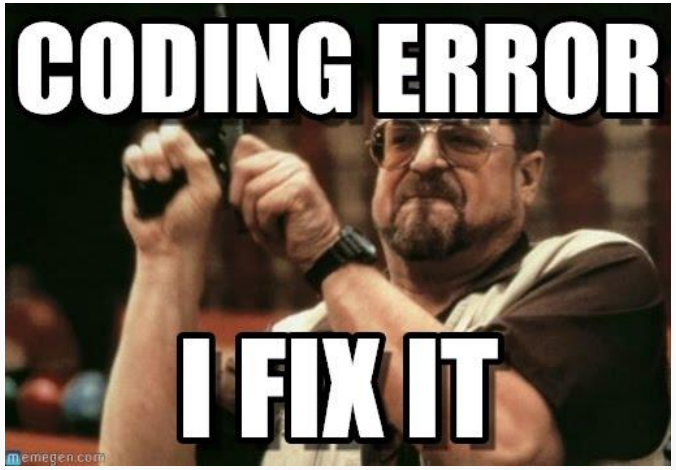
Much better use the “up” of the plane we are colliding with

And Finally …



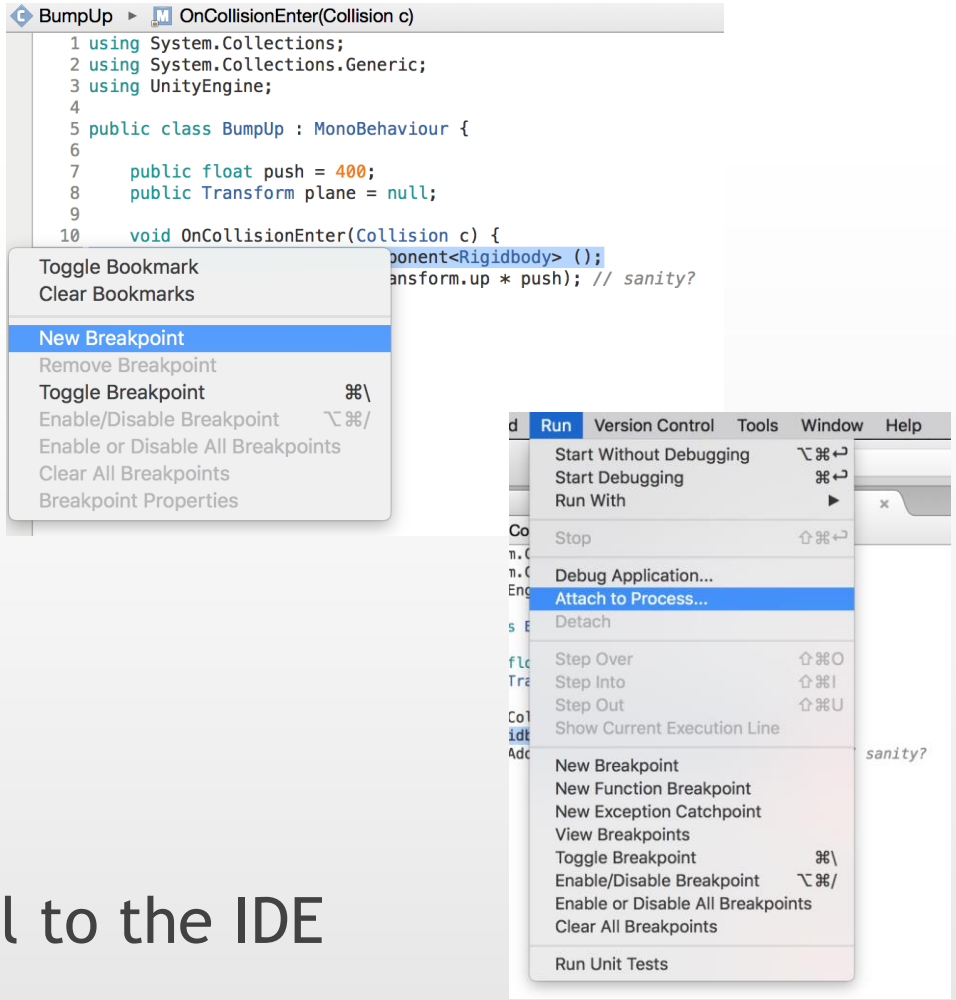
No Software is Bug Free

- But, unfortunately, the console output of Unity is also very crappy
  - Delayed flush
  - Lot of clutter
  - Difficult to read
    - There is the stacktrace in every message
  - **Uses a lot of CPU**



But We Can Debug!

- Add a breakpoint with your IDE
- Attach debugging to the unity process
- Start the game in Unity and proceed as usual
- Unity will pause and give control to the IDE upon reaching the breakpoint



The Single Thread Menace

- An important contract holds:
  Start() and Update() are not interruptible!

- This is to ensure consistency of shared resources
  - Did you care about that in the operating system class?

- If you get stuck in an Update(), any Update(), Unity will freeze
  - You will be required to kill and restart the whole environment !

Performing Side Tasks

- You can spawn a secondary thread (it is C# after all)
  … but the secondary thread will NOT be able to access any scene data (to guarantee consistency again)
  - Yes, this sucks!

- You can write a stateful *Update()*
  - But then you must spawn a gameobject for every task

- … or you can delegate the task to a coroutine

Coroutine

- It is just like a method which can “get suspended” and restart later without bothering any *Update()*

- Typical usage are tasks to be performed only once, have a delayed effect, or need a specific execution rate
  - Steering
  - Jumping
  - Counting
  - Fading

Fade on Click

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

public class BumpNFade : MonoBehaviour {

    public float push = 400f;
    public float fadeTime = 3f;

    void OnCollisionEnter(Collision c) {
        Rigidbody rb = GetComponent<Rigidbody> ();
        rb.AddForce (c.gameObject.transform.up * push);
    }

    void OnMouseDown() {
        StartCoroutine (Fade ());
    }

    // Remember to set te rendering mode of the shader to "fade"

    IEnumerator Fade() {
        Material m = GetComponent<Renderer> ().sharedMaterial;
        Color c;
        for (float f = 1f; f >= 0; f -= 0.1f) {
            c = m.color; c.a = f; m.color = c;
            yield return new WaitForSeconds(fadeTime / 10f);
        }
        c = m.color; c.a = 1f; m.color = c; // danger here!
        Destroy (gameObject);
    }
}
```

Will be called by the runtime when we click on the object

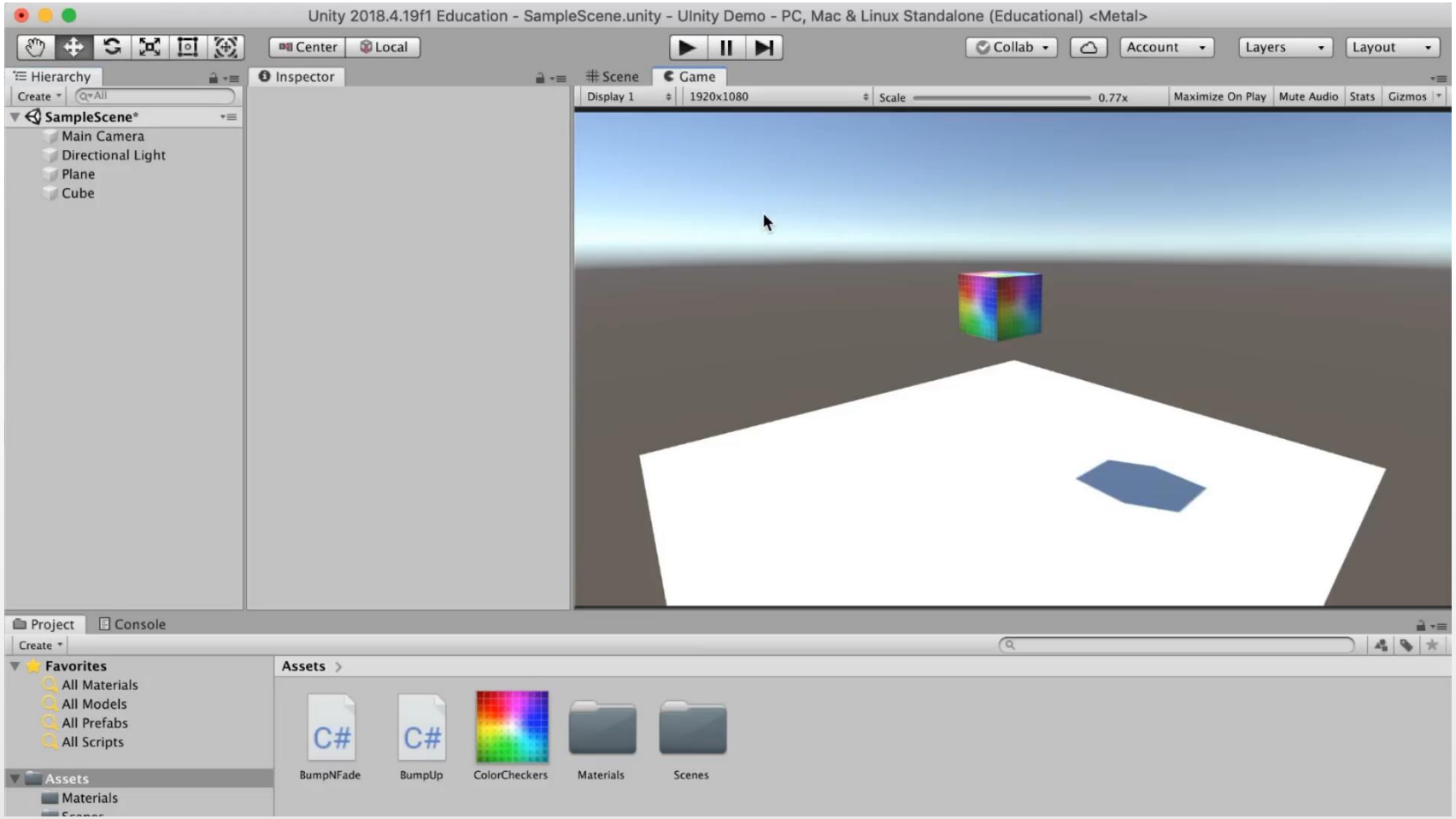
Hidden traps here!

Set transparency to appropriate value

Suspend and get rescheduled in 0.3 seconds (3/10)

When the last statement is reached, the coroutine is over and all resources are freed

And Let's Fade



Creating Blueprints

- You have this beautiful gameobject with all components set and tuned … but you need TWO of them
- And what if you need many, and at runtime …
  like a bullet spawning from a gun?
- Drag it to the asset database and make it a prefab
  - Gameobject will turn blue in the scene hierarchy
  - From now on, it will be like an asset (including its parameters)
- Spawn can be performed via *Instantiate()*

A Simple Spawner

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

public class Spawn : MonoBehaviour {

    public Transform toSpawn = null;
    public float rate = 1f;

    void Start () {
        StartCoroutine (Spawner ());
    }

    IEnumerator Spawner() {
        while (true) {
            if (toSpawn != null)
                Instantiate (toSpawn, transform.position + Random.insideUnitSphere, Quaternion.identity);
            yield return new WaitForSeconds(1f / rate);
        }
    }

}
```

In the interface we will be required to drag and drop the blueprint into a slot of the component

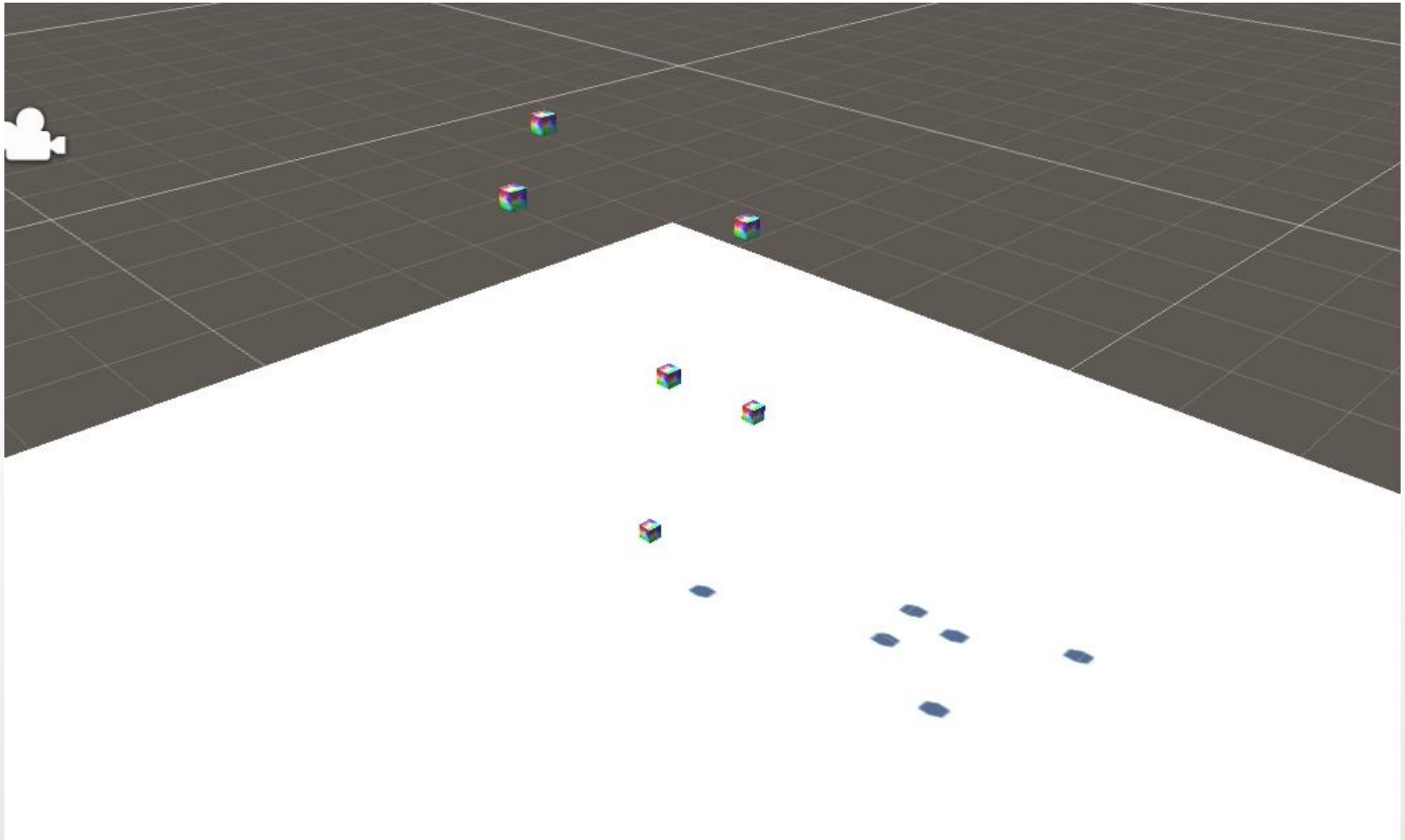
What to spawn

Where to spawn it

Orientation (we do not really care)

Guarantee for a fixed rate!

Spawning



Changing the spawning rate on the fly

Happy Hacking

Dario Maggiorini (dario@di.unimi.it)