CT3536

Games Programming

Dr. Sam Redfern

sam.redfern@nuigalway.ie
https://discord.gg/nqD5JN95WT
http://www.psychicsoftware.com

CT3536

Games Programming

BSc Computer Science & IT (3rd year)
BA Digital Arts & Technology (3rd year)
Structured PhDs (?)
Visiting Students. (?)

CT3536
Games Programming

Goals..

- 2nd year (CT255)
  - directly support your Java programming
  - focused on 2D games development projects

- 3rd year
  - bring together the games/media skills and programming covered since 1st year
  - develop full games using a modern Games Engine
  - practicing software development and learn some useful things in C#

THIS MODULE REQUIRES A REASONABLE GENERAL KNOWLEDGE OF PROGRAMAMING

CT3536
Games Programming

Lectures:

- Wednesdays 10am-11.50am IT125G

Labs:

- Mondays 1pm-3pm IT106
- Starting next week

Bring your own laptop (required)

Discord Server

The following Discord server has been set up. We can discuss the course and you can ask questions there anytime during the semester. This is also how I'll distribute materials/solutions etc.

https://discord.gg/nqD5JN95WT

Unity3D Game Engine (unity3d.com)

A closed-source games engine

- Well designed and elegant to use "2nd Generation" game engine
- Excellent GUI/HUD editing and animation system
- Powerful & very popular
- Hugely successful Asset Store
- Core language: C# (very similar to Java)
- Deploys to: iOS, Android, Web (HTML5/WebGL), Windows, Mac OSX, Linux, Switch, and more

Why take a closed source route?

- to work on a complete game at a higher level, now that you have the skills, rather than to drill down to finer detail on an open-source engine
- I think you'll find it useful (and fun?), and by the end of the course you'll be able to make complete, decent quality games/multimedia apps.

Unity in the Lab

Sign up for an account on Unity3D.com, and download+install Unity – as soon as possible, and definitely before your first lab session

You will require your own laptops: our lab rooms have desks and docking stations but no desktop PCs.

We have just 12 weeks..

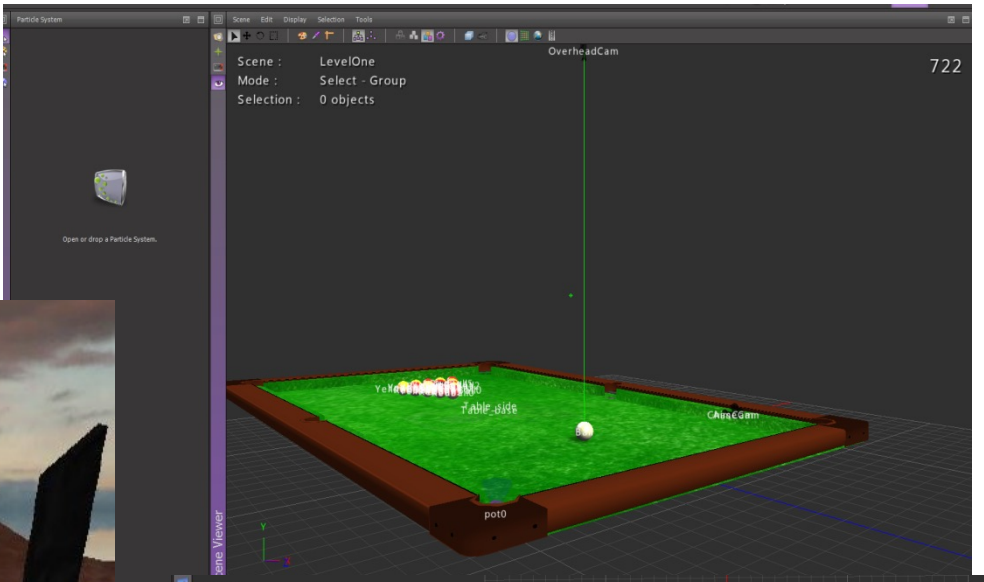
- Overview of Game Engines
- Overview of the Unity IDE and API
- Key Game Entities
- Flow of Control
- Data
- C#
- Translation, rotation, scale of game objects
- The physics engine
- Triggers and colliders
- The camera object
- GUIs
- Raycasting
- Runtime object management, efficiency, garbage collection
- Particle emitters
- Materials, Shaders and Lighting
- State machines
- 2D games
- Audio
- Game Design

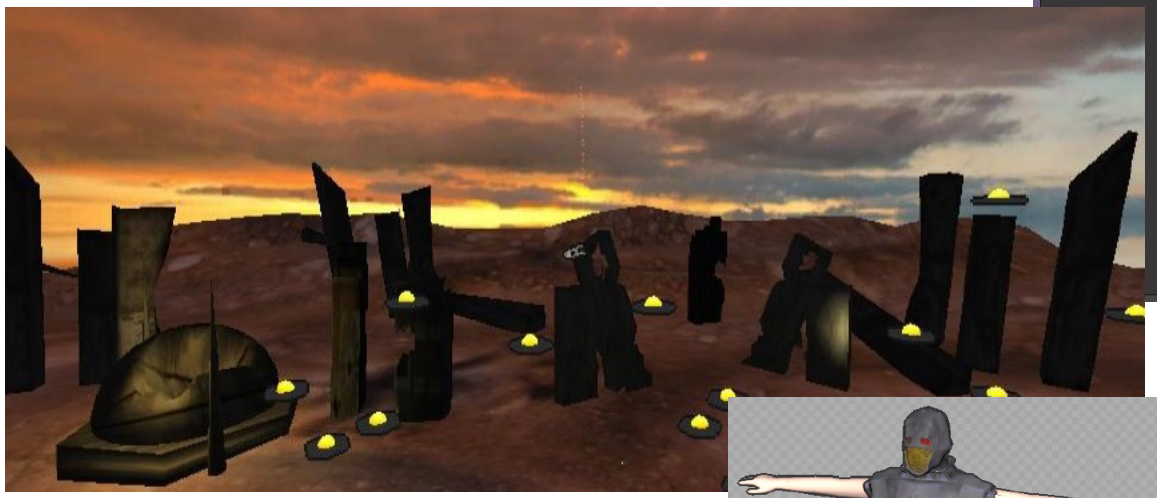
Game projects

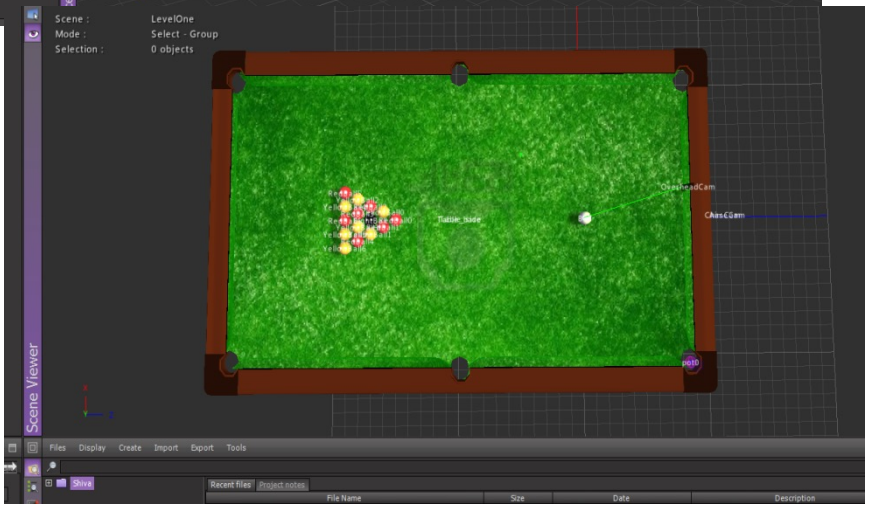
- 1-3 person projects
- To begin on week 5 (approx.)
- Will be worth 30% of the course marks (with 10% going to the six graded lab assignments, and 60% to the final exam)
- Demo on final week
- Submit as a document containing images, discussion, code
- Marks are awarded for:
  - Overall complexity
  - Code architecture and neatness
  - Game design/elegance and UX
  - Discussion
  - Graphics (if self-created)
  - Audio (if self-created)
  - (Group size is also taken into account)

- Start to consider your ideas by week 5!
  - Please discuss with me (and the lab tutors) as we can advise on scope and difficulty
  - Simple 3D games are no harder than 2D games
  - ‘Snake’ games are not allowed!

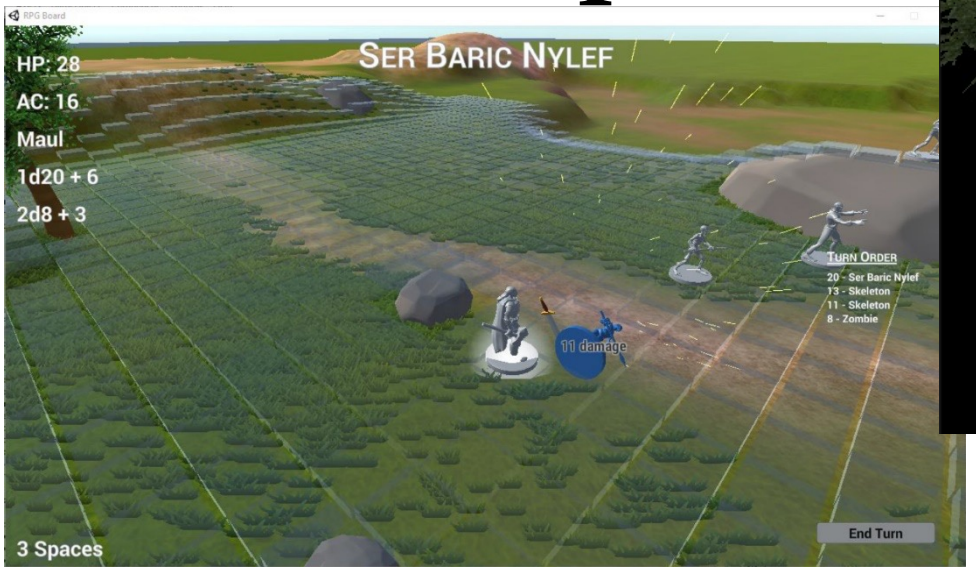
Some projects from the past

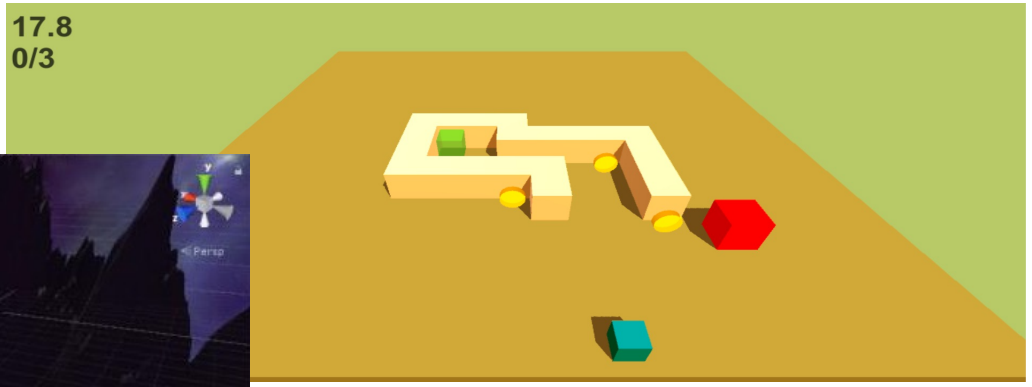


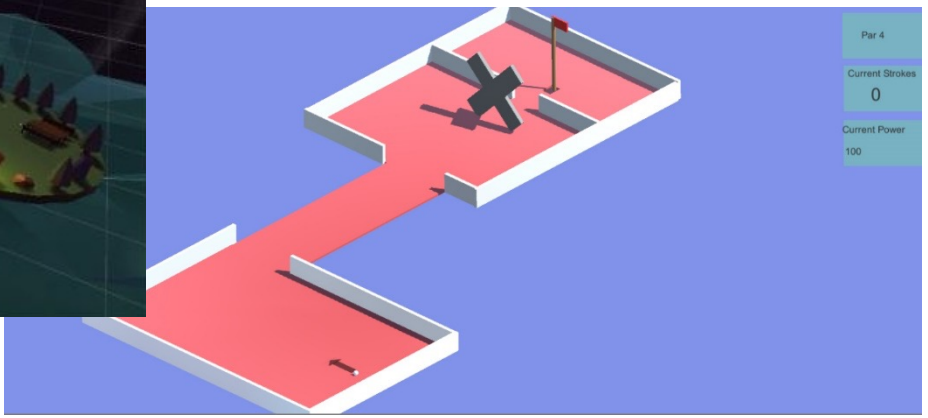




Some projects from the past

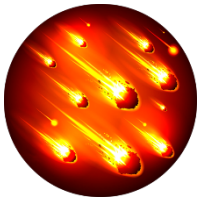


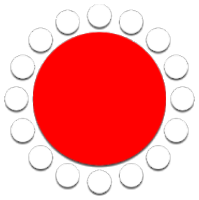




Some of my own games

- www.psychicsoftware.com
- 1980s: BBC Micro (BBC Basic, Assembler)
- 1990s: Commodore Amiga (AMOS)
- 2000s: PC/Mac (Torque Game Engine)
- 2010-2014: PC/Mac/Smartphones (Shiva3D game engine)
- 2015-now PC/Mac/Smartphones/Web (Pixijs render engine/Nodewebkit/Cordova)
- 2017-now PC/Mac/XBox/PS/Switch (Unity3D game engine)





Game Engines

- Game Engines provide a powerful set of integrated sub-systems geared towards making games (and other high-performance *realtime* media)
  - Graphics Rendering (3D, 2D, terrain)
  - Physics
  - Networking
  - Special Effects
  - 3D Audio
  - User Input

The Game Loop

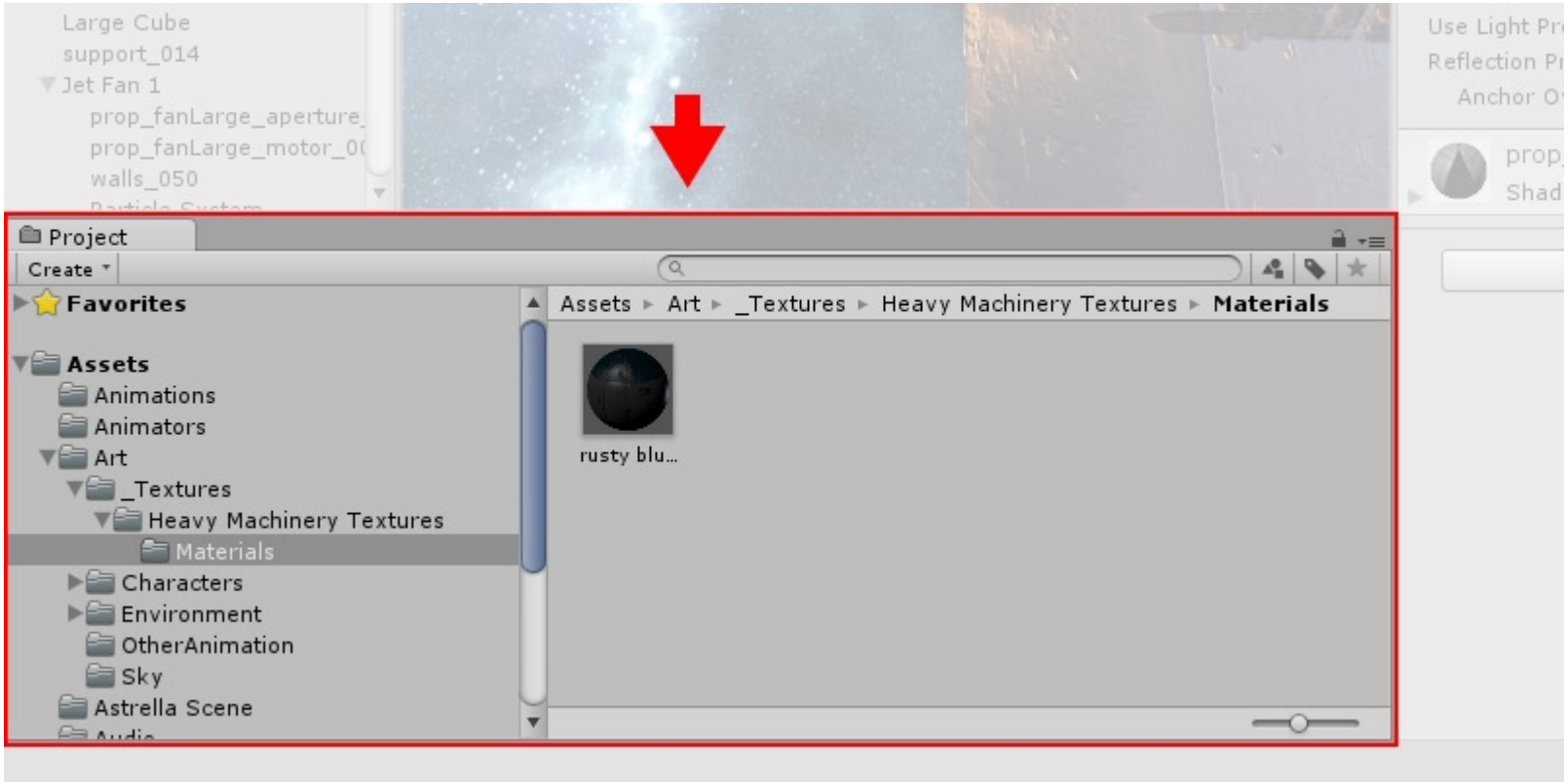
- At their core, games operate a *game loop* (although game engines somewhat hide this from you, it's still useful to know):
  - Operating at 60 fps (or more)
  - Deal with inputs
    - Received asynchronously via events (e.g. keyboard, or network data), or polled for right now
  - Process game objects
    - User-controlled objects
    - Enemy AI
    - Other scheduled behaviours (e.g. sprite frame updates)
  - Move game objects
    - Move the physics simulation (if any) forwards
    - Move objects by physics simulation or direct control
  - Redraw
  - Wait (maybe) or run at maximum obtainable speed (maybe)
    - but don't block the main thread

The Unity IDE

- https://docs.unity3d.com/Manual/UsingTheEditor.html

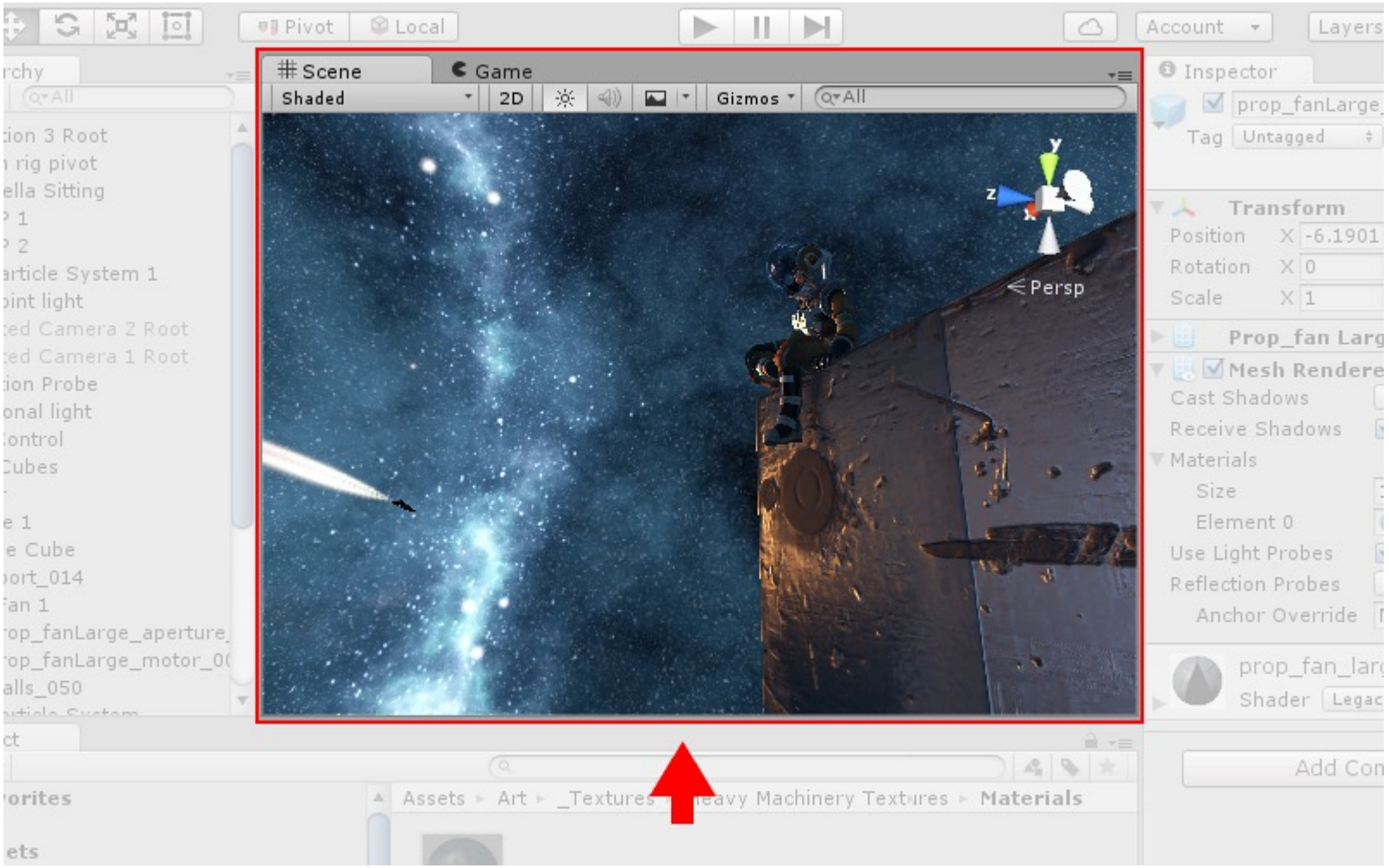
A. Toolbar
B Hierarchy Winow
C Game View
D Scene View
E Overlays
F Inspector Window
G Project Window
H Status Bar

The Project Window (Assets)



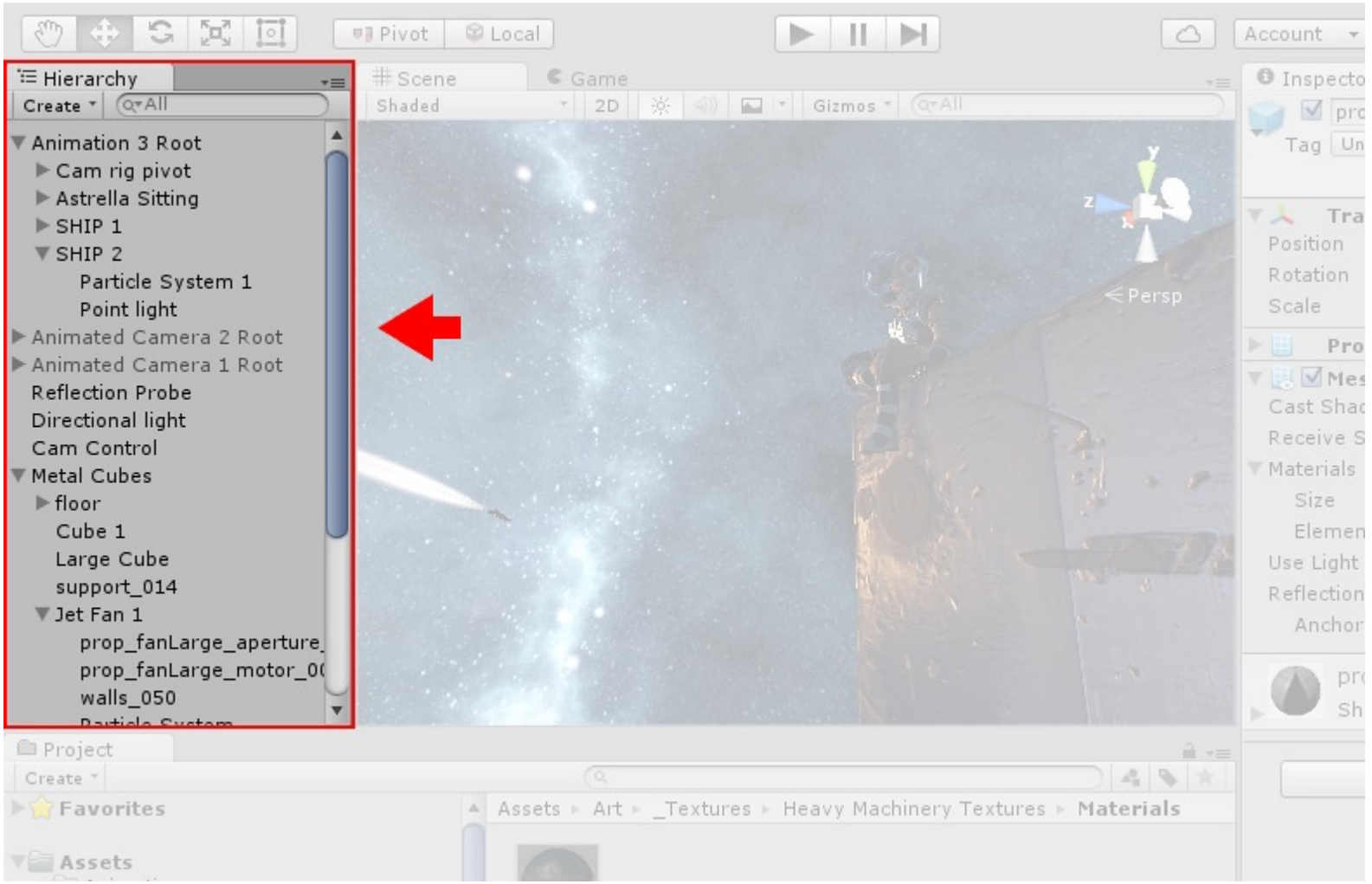
- The Project Window displays your library of assets that are available to use in your project.
- When you import assets into your project, they appear here
- More details: https://docs.unity3d.com/Manual/ProjectView.html

The Scene View



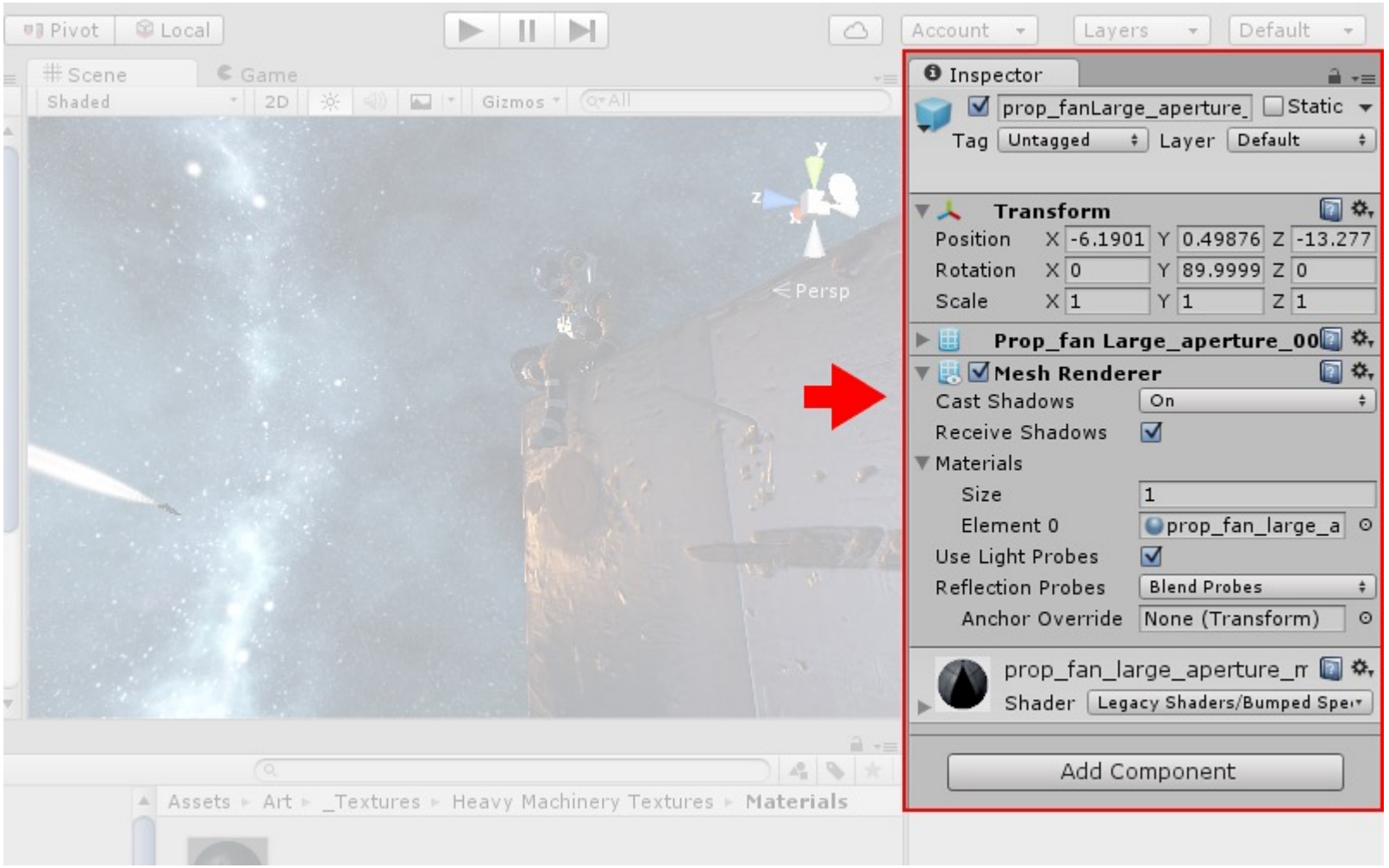
- **The Scene View** allows you to visually navigate and edit your scene (without running the game).
- The scene view can show a 3D or 2D perspective

The Hierarchy (Scene Graph) window



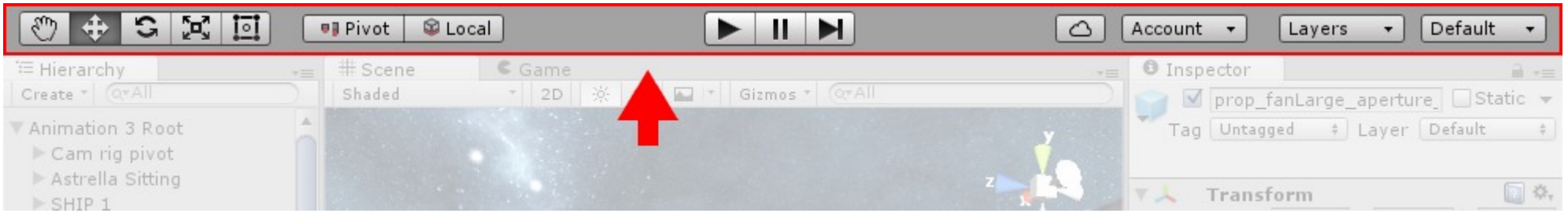
- **The Hierarchy Window** is a hierarchical (nested) text representation of every "**game object**" in the scene

The Inspector window



- **The Inspector Window** allows you to view and edit all the properties of the currently selected game object.
- More detail: https://docs.unity3d.com/Manual/UsingTheInspector.html

The Toolbar



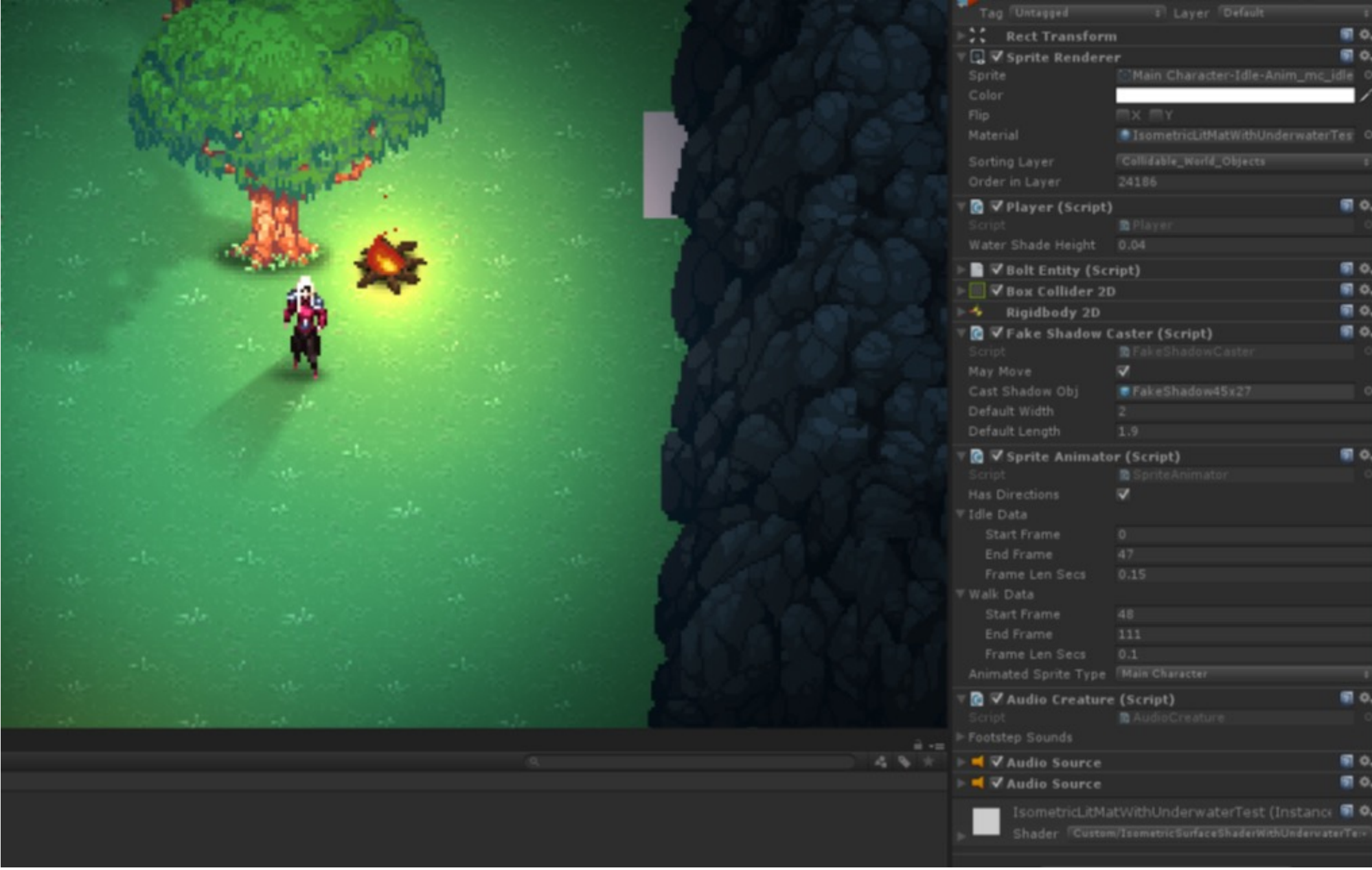
- **The Toolbar** provides access to the most essential working features.
- On the left it contains the basic tools for manipulating the scene view and the objects within it.
- In the centre are the play and pause controls.
- The buttons to the right give you access to your Unity Cloud Services and your Unity Account, followed by a **layer** visibility menu, and finally the editor layout menu (which provides some alternate layouts for the editor windows, and allows you to save your own custom layouts).

The Unity API: overview

- Luckily everything is not just drag-n-drop. We can write lots of juicy C# code to make the game work
- Much of our code will involve manipulation of the Unity API classes from the "**UnityEngine**" and "**UnityEngine.UI**" namespaces
- Of course, we can also do anything else supported by the core C#/.NET library (file handling, networking, collection classes, etc.)

Component Based Architecture

- 2nd generation game engines such as Unity3D use CBA which suits game logic very well
- Different to classic OOP as it's based on the *Composition* rather than *Inheritance* principle
- Every *entity* consists of one or more *components* which add additional behaviour or functionality.
- The behaviour of an entity can even be changed at runtime by adding or removing components.
- Eliminates the ambiguity problems of deep and wide inheritance class-hierarchies that are difficult to understand, maintain and extend
- **Each component is, essentially, a separate software object –** but all **attached to some higher level GameObject** (which typically equates to an actual onscreen character, enemy, bullet, effect, vehicle etc.)



- A player character and a tree are not examples of the same thing, so how can a class hierarchy be a sensible way to architect them? (even though we want code re-use)
- They share characteristics with each other, so these are modelled as components in a CBA
- - e.g. they both cast shadows

Composition over Inheritance

- Using composition rather than inheritance can be somewhat a mantra, and not correct in all cases
- Code reuse is often (incorrectly) considered to be the main principle of inheritance
- In fact, designing hierarchical taxonomies of classes is the most appropriate reason to use inheritance
- Composition is better when you would otherwise get tangled up in the murky world of multiple- inheritance
- Composition is excellent for code reuse
- https://www.thoughtworks.com/insights/blog/composition-vs-inheritance-how-choose

Some References

- Interface/IDE basics:
  - https://docs.unity3d.com/Manual/LearningtheInterface.html
  - https://docs.unity3d.com/Manual/UnityOverview.html
- Unity Manual:
  - https://docs.unity3d.com/Manual/index.html
- Unity API reference:
  - https://docs.unity3d.com/Documentation/ScriptReference/index.html
- Tutorials:
  - https://learn.unity.com/

Introducing Demon Pit

- A game I've recently finished (release Oct-Dec 2019)
- Provides lots of good (mostly code) examples (of appropriate complexity) which I will illustrate in this module
- A first person shooter (FPS) arena game.. Not the world's most original game, but relatively simple and appropriate for study in this module
- 10 types of enemy
  - Flyers
  - Walkers (pathfinding)
  - Melee attacks
  - Gun attacks
  - Animations
- 7 types of player gun
  - Raycast
  - Projectile
  - Particle
- Constantly changing arena (walls, floors, lava)
- 'Lasso' mechanism
- https://store.steampowered.com/app/548240/Demon_Pit/
  https://www.youtube.com/watch?v=Lu01dXKGLL8

Lab Session 1

- Import a model of Mars (and a texture .jpg for it)
- Create a simple demo which rotates Mars using the built in Physics engine
- (We could also take direct control of the Mars object and rotate it ourselves, a little bit on every frame.. but here we're letting the Physics engine do the work)
- No need to submit anything from this 1st lab session (not graded)

- I'll demonstrate this now..
- … you probably want to write down the C# code I'm writing