BOHEMIA INTERACTIVE PRESENTS

Unity Tips & Tricks

Xeniya Vondrášková, Filip Vondrášek

YLANDS

Who are we?

We work at Bohemia Interactive as programmers

Our current project is Ylands, a sandbox game and a platform for creating your own games.

Currently in early access, leaving EA in Q1 2019

What we are going to talk about

Unintuitive Unity APIs

Unity Garbage Collector

LINQ, foreach

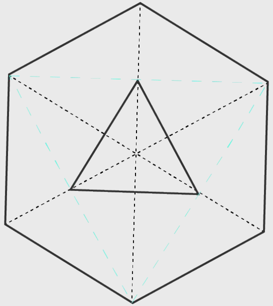
Structure memory layout

Unity assembly reload

Data-oriented approach

Rendering millions objects

General Unity tips



Unintuitive Unity APIs

Camera API: Camera.main

The first enabled camera tagged as "*MainCamera*"

Camera.main is not a direct reference.

Every time you call **Camera.main** it uses **FindGameObjectsWithTag** internally and returns the result. The result is not cached.

It happens multiple times per frame

Solution?
Cache it manually and track changes of the main camera.

Particle system API

Almost all methods of the Particle System are **recursive calls.**

- iteration cycle through each child of the PS, calling **GetComponent<ParticleSystem>()** on each of them and potential calling of the original method on each child separately

Most-used API are affected by this behaviour: **Start(), Stop(), Pause(), Clear(), Simulate()** and even **IsAlive()**

Could be a problem for the deep particles hierarchy, which is common for complex effects

Particle system API

Potential solution:

- All of these methods come with **withChildren** parameter, that is true by default
- Cache your Particle Systems and manually iterate over them

WaitForSeconds()

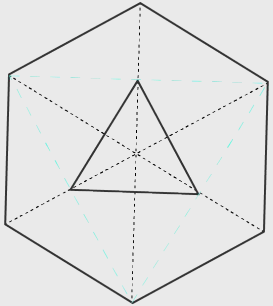
WaitForSeconds() sounds fairly self-explanatory

Beware of scaled time: given time is divided by **Time.timeScale.**

- In cases when you are tweaking the time scale value (for example, for slow motion effects) or pausing the game using Time.timescale = 0.

It means for Time.timeScale = 0.5f your WaitForSeconds(1f) will actually wait 2 seconds. TimeScale = 0 causes WaitForSeconds coroutines not to run.

Alternative:
Use WaitForSecondsRealtime() instead, that uses unscaled time



Unity Garbage Collector

Unity Garbage Collector

Part of today’s speech will be about memory allocations.

Unity uses Boehm GC algorithm:

1. Periodically sweeps through **all objects** stored on the heap.
2. Marks unreferenced objects for deletion.

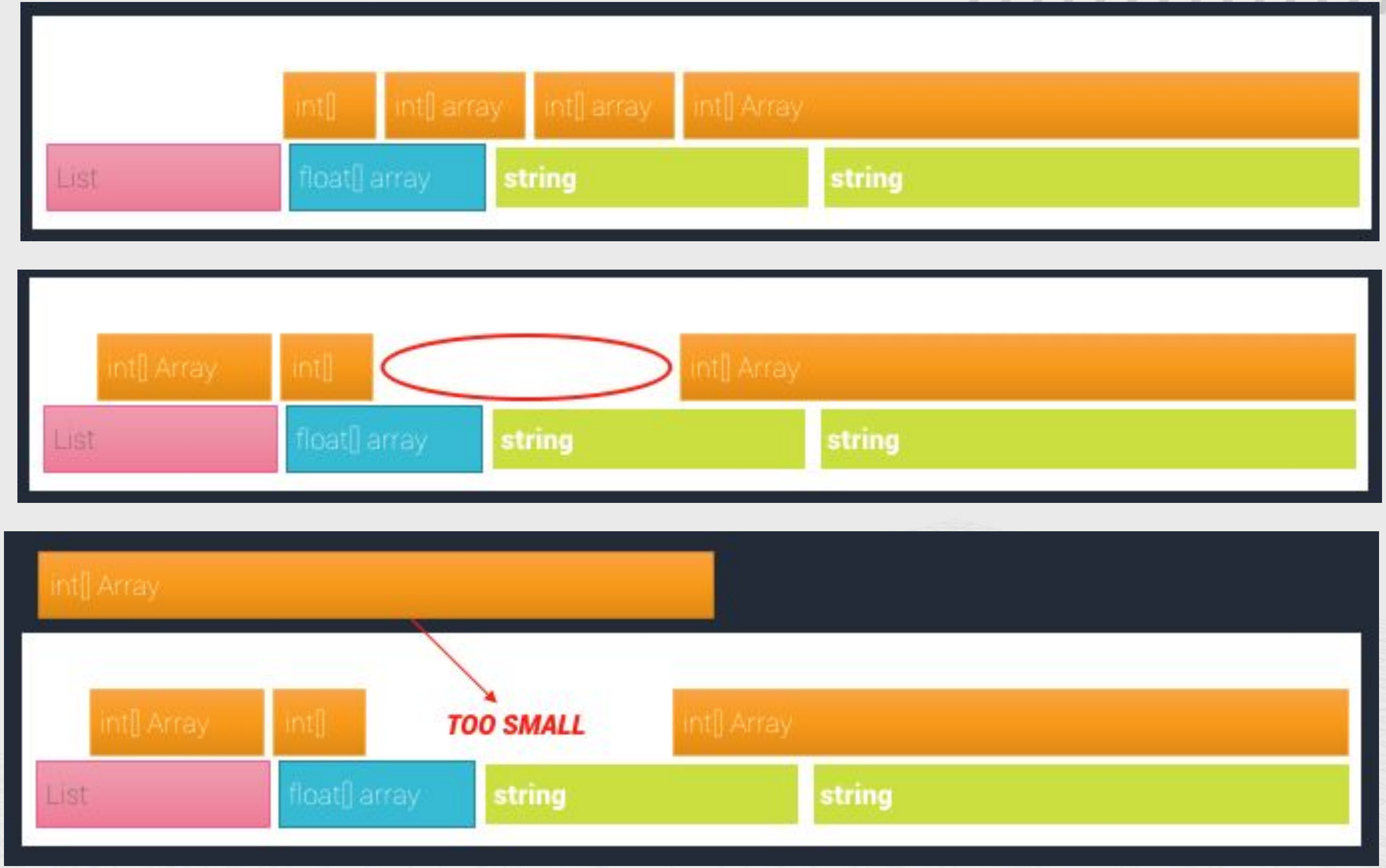
Unity Garbage Collector

Doesn’t defragment the heap (non-compact).

The time needed for a GC pass is **directly dependent on the size** of the heap. (non-generational)

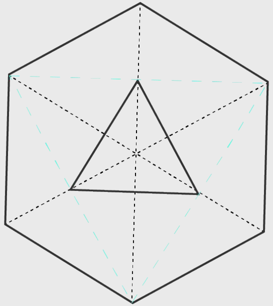
From less than 1 ms to hundreds of ms

Stops the world during a sweep: GC spikes



(Images are taken from the Unity Manual)

Source of the pictures on previous slide: https://docs.unity3d.com/Manual/BestPracticeUnderstandingPerformanceInUnity4-1.html



LINQ

Not enough space on the heap?

Run the GC (if it hasn’t run recently)

If we freed a **gap large enough** to store the new allocation, use it

Otherwise, **expand the heap** (usually double the size on most platforms)

The expanded space is **not often shrunk** if it’s empty

The address space is **never returned** to the OS

Keep your allocations at minimum

Coroutines: yield return new …

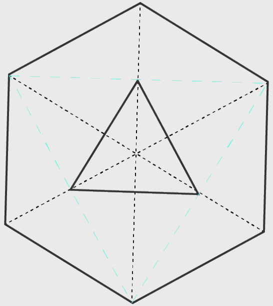
yield return new WaitForSeconds(…) allocates every time you use it

Let's write a helper class!

Coroutines: yield return new …

```
public static class YieldUtil {
      private static var _waits = new Dictionary<float, WaitForSeconds>();

      public static IEnumerator WaitForSeconds(float seconds) {
            WaitForSeconds rv;
            if (!_waits.TryGetValue(seconds, out rv)) {
                  rv = new WaitForSeconds(seconds);
                  _waits.Add(seconds, rv);
            }
            return rv;
      }
}
```



LINQ

LINQ

Easy to write, easy to read

Generates a lot of garbage!

- Even Microsoft advises on their official web not to use LINQ in Unity because of heavy allocations
- https://docs.microsoft.com/cs-cz/windows/mixed-reality/performance-recommendations-for-unity

Some platforms (iOS) don’t work very well at all with LINQ

- Produces AOT/Jitter errors

Bad performance

LINQ: Initialization

Let’s have a simple class containing only Vector3 Position called Player and a List of 100.000 instances of that class:

LINQ: Initialization

```
public class Player
{
      public Vector3 Position;
}

public void Init()
{
      _players = new List<Player>();
      for (int i = 0; i < 100000; ++i)
      {
            _players.Add(new Player(new Vector3(i, i, i)));
      }
}
```

LINQ: Selection

Consider these two snippets that pick the players whose x coordinate is an even number:

LINQ: Selection

Consider these two snippets that pick the players whose x coordinate is an even number:

```
List<Player> evenPlayers = _players.Where(x => x.Position.x % 2 == 0).ToList();
```

LINQ: Selection

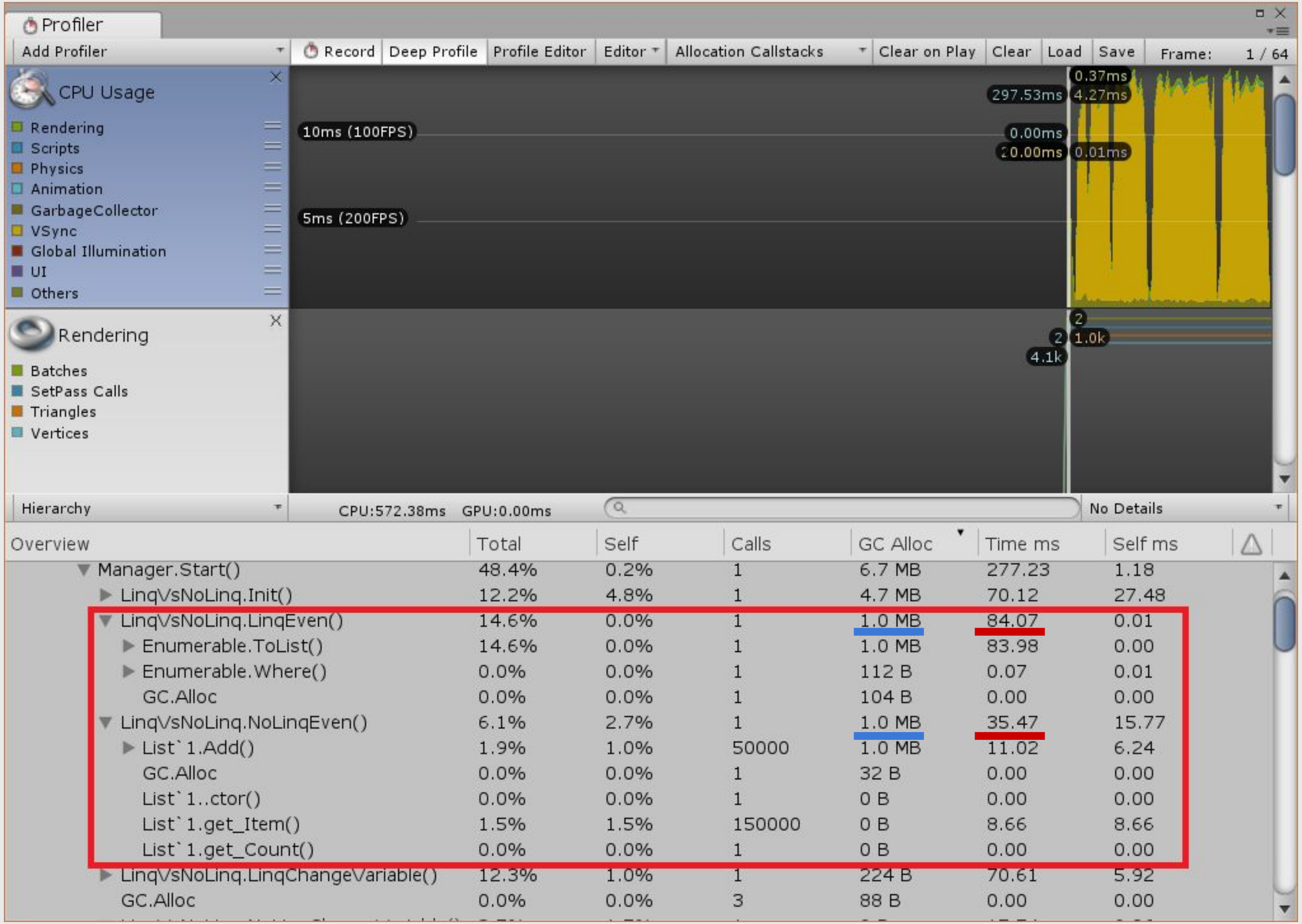
Consider these two snippets that pick the players whose x coordinate is an even number:

```
List<Player> evenPlayers = _players.Where(x => x.Position.x % 2 == 0).ToList();
```

and :

```
List<Player> evenPlayers = new List<Player>();

int count = _players.Count;
for (int i = 0; i < count; ++i)
{
      if (_players[i].Position.x % 2 == 0)
      {
            evenPlayers.Add(_players[i]);
      }
}
```



Overview	Total	Self	Calls	GC Alloc	Time ms	Self ms
▼ Manager.Start()	48.4%	0.2%	1	6.7 MB	277.23	1.18
▶ LinqVsNoLinq.Init()	12.2%	4.8%	1	4.7 MB	70.12	27.48
▼ LinqVsNoLinq.LinqEven()	14.6%	0.0%	1	1.0 MB	84.07	0.01
▶ Enumerable.ToList()	14.6%	0.0%	1	1.0 MB	83.98	0.00
▶ Enumerable.Where()	0.0%	0.0%	1	112 B	0.07	0.01
GC.Alloc	0.0%	0.0%	1	104 B	0.00	0.00
▼ LinqVsNoLinq.NoLinqEven()	6.1%	2.7%	1	1.0 MB	35.47	15.77
▶ List`1.Add()	1.9%	1.0%	50000	1.0 MB	11.02	6.24
GC.Alloc	0.0%	0.0%	1	32 B	0.00	0.00
List`1..ctor()	0.0%	0.0%	1	0 B	0.00	0.00
List`1.get_Item()	1.5%	1.5%	150000	0 B	8.66	8.66
List`1.get_Count()	0.0%	0.0%	1	0 B	0.00	0.00
▶ LinqVsNoLinq.LinqChangeVariable()	12.3%	1.0%	1	224 B	70.61	5.92
GC.Alloc	0.0%	0.0%	3	88 B	0.00	0.00

LINQ: Manipulation

Let’s have a code that changes the y coordinate to 0 for Players with an even x coordinate:

LINQ: Manipulation

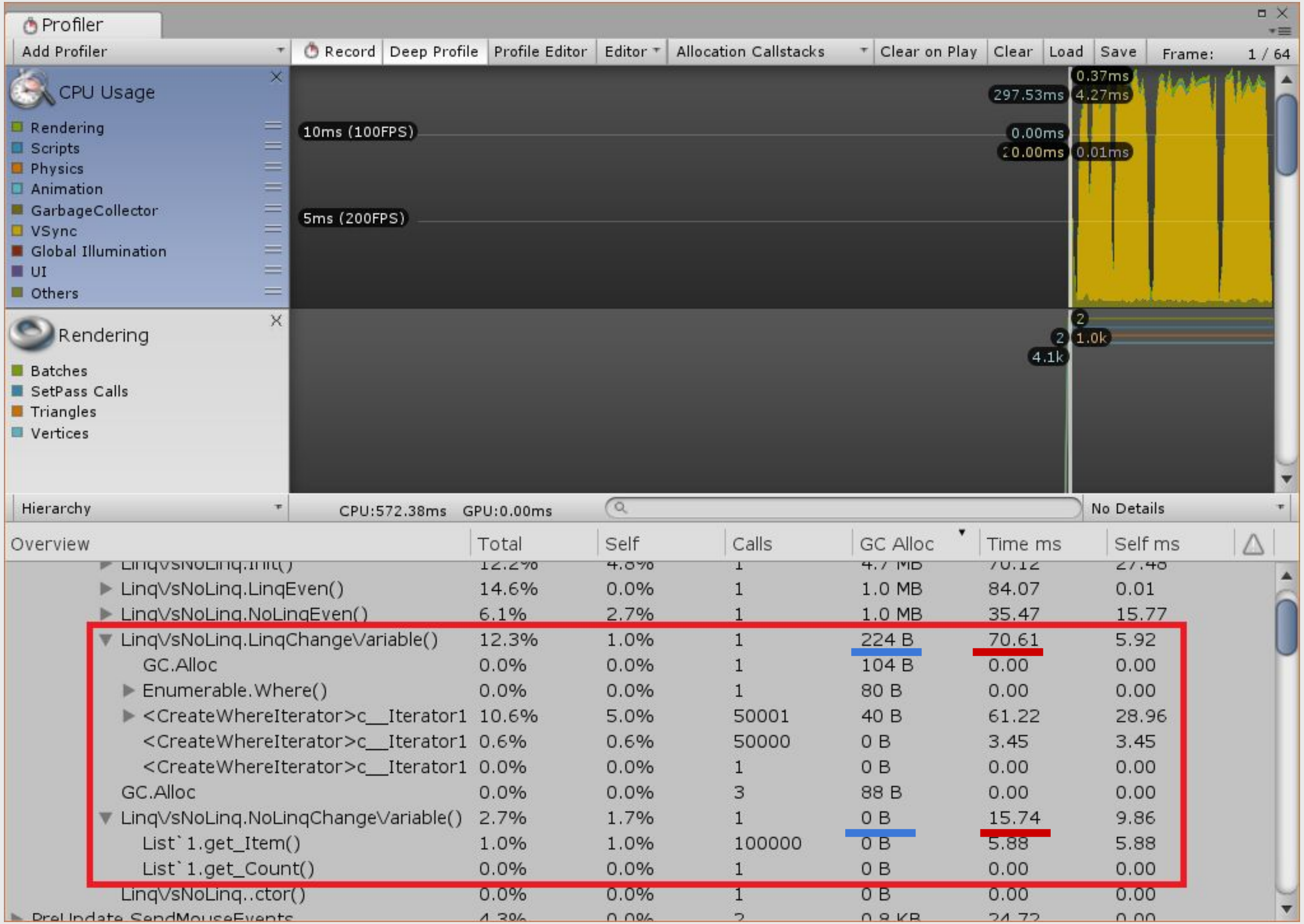
Let’s have a code that changes the y coordinate to 0 for Players with an even x coordinate:

```
IEnumerable<Player> evenPlayers = _players.Where(x => x.Position.x % 2 == 0);
for (var e = evenPlayers.GetEnumerator(); e.MoveNext();)
{
      e.Current.Position.y = 0f;
}
```

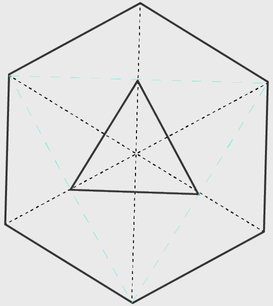
LINQ: Manipulation

and

```
int count = _players.Count;
for (int i = 0; i < count; ++i)
{
      Player player = _players[i];
      if (player.Position.x % 2 == 0)
      {
            player.Position.y = 0f;
      }
}
```



Overview	Total	Self	Calls	GC Alloc	Time ms	Self ms
LinqVsNoLinq.Init()	12.2%	4.8%	1	4.7 MB	70.12	27.48
LinqVsNoLinq.LinqEven()	14.6%	0.0%	1	1.0 MB	84.07	0.01
LinqVsNoLinq.NoLinqEven()	6.1%	2.7%	1	1.0 MB	35.47	15.77
LinqVsNoLinq.LinqChangeVariable()	12.3%	1.0%	1	224 B	70.61	5.92
GC.Alloc	0.0%	0.0%	1	104 B	0.00	0.00
Enumerable.Where()	0.0%	0.0%	1	80 B	0.00	0.00
<CreateWhereIterator>c__Iterator1	10.6%	5.0%	50001	40 B	61.22	28.96
<CreateWhereIterator>c__Iterator1	0.6%	0.6%	50000	0 B	3.45	3.45
<CreateWhereIterator>c__Iterator1	0.0%	0.0%	1	0 B	0.00	0.00
GC.Alloc	0.0%	0.0%	3	88 B	0.00	0.00
LinqVsNoLinq.NoLinqChangeVariable()	2.7%	1.7%	1	0 B	15.74	9.86
List`1.get_Item()	1.0%	1.0%	100000	0 B	5.88	5.88
List`1.get_Count()	0.0%	0.0%	1	0 B	0.00	0.00
LinqVsNoLinq..ctor()	0.0%	0.0%	1	0 B	0.00	0.00
PreUpdate.SendMouseEvents	4.3%	0.0%	2	0.8 KB	24.72	0.00



Foreach vs for

Foreach vs for

Foreach in most cases **no longer generates garbage** as it used to before Unity 5.6

Don’t iterate over **IList<T>, IEnumerable<T>**, that still has to do Boxing, which allocates (enumerator allocation)

The problem with foreach is no longer garbage, but the **performance**

Let’s have the same Player class as in the LINQ part and a List<Player> of 100.000 objects

Let’s reset the Position variable in each object

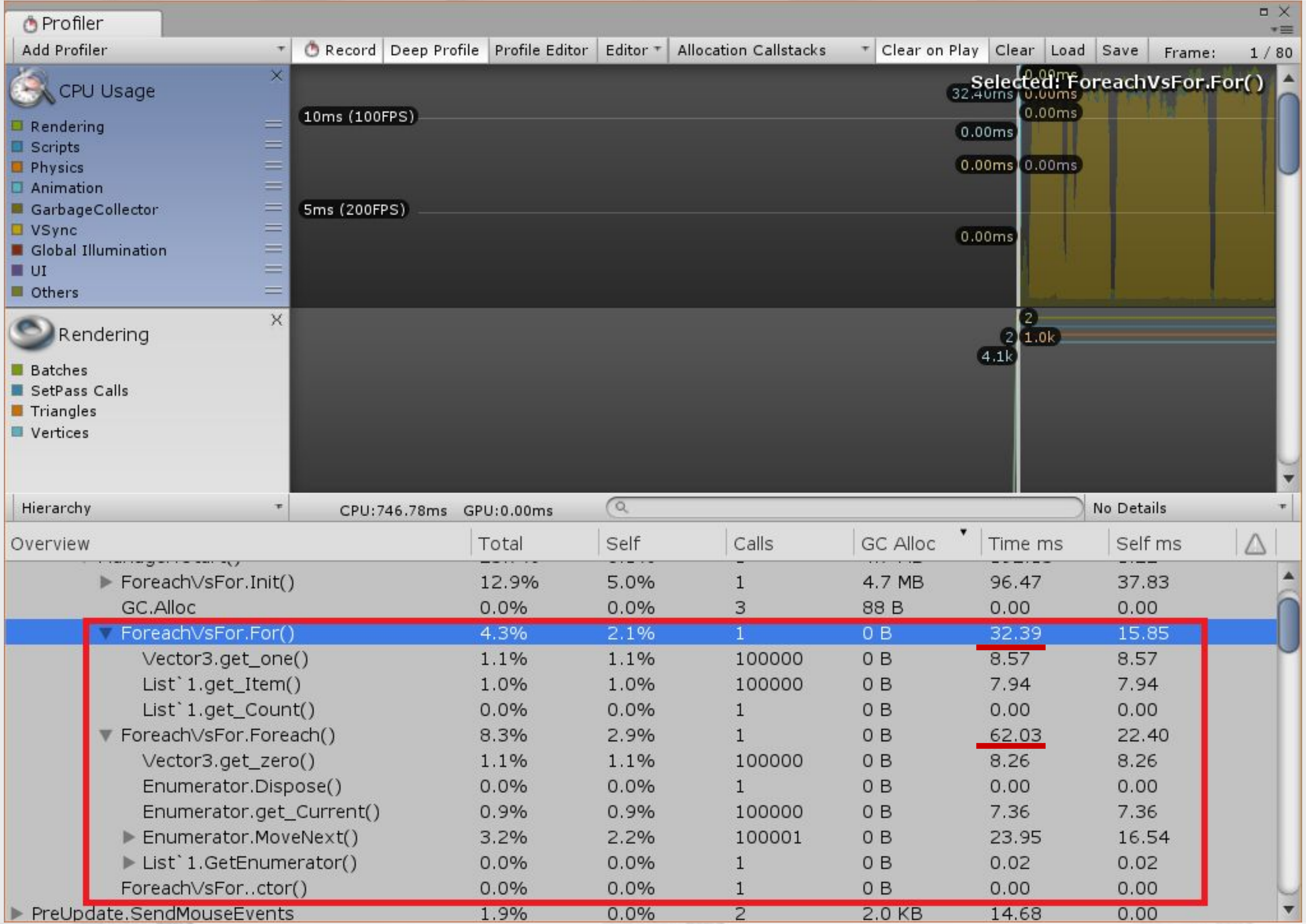
Foreach vs for

```
foreach (Player player in _players)
{
	player.Position = Vector3.zero;
}
```

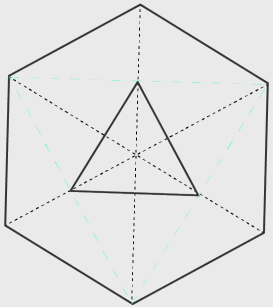
Foreach vs for

```
foreach (Player player in _players)
{
      player.Position = Vector3.zero;
}

int count = _players.Count;
for (int i = 0; i < count; ++i)
{
      _players[i].Position = Vector3.zero;
}
```



Overview	Total	Self	Calls	GC Alloc	Time ms	Self ms
▶ ForeachVsFor.Init()	12.9%	5.0%	1	4.7 MB	96.47	37.83
GC.Alloc	0.0%	0.0%	3	88 B	0.00	0.00
▼ ForeachVsFor.For()	4.3%	2.1%	1	0 B	32.39	15.85
Vector3.get_one()	1.1%	1.1%	100000	0 B	8.57	8.57
List`1.get_Item()	1.0%	1.0%	100000	0 B	7.94	7.94
List`1.get_Count()	0.0%	0.0%	1	0 B	0.00	0.00
▼ ForeachVsFor.Foreach()	8.3%	2.9%	1	0 B	62.03	22.40
Vector3.get_zero()	1.1%	1.1%	100000	0 B	8.26	8.26
Enumerator.Dispose()	0.0%	0.0%	1	0 B	0.00	0.00
Enumerator.get_Current()	0.9%	0.9%	100000	0 B	7.36	7.36
▶ Enumerator.MoveNext()	3.2%	2.2%	100001	0 B	23.95	16.54
▶ List`1.GetEnumerator()	0.0%	0.0%	1	0 B	0.02	0.02
ForeachVsFor..ctor()	0.0%	0.0%	1	0 B	0.00	0.00
▶ PreUpdate.SendMouseEvents	1.9%	0.0%	2	2.0 KB	14.68	0.00



Structure memory layout

Structure memory layout

```
public struct NPCData
{
      public Vector3 Position;
      public byte IsPositionCurrent;
      public Quaternion BodyOrientation;
      public byte IsOrientationCurrent;
      public int Health;
      public byte HasEverTakenDamage;
      public int Damage;
      public byte HasEverShot;
      public Quaternion HeadOrientation;
}
```

Structure memory layout

```
unsafe
{
	Debug.Log(sizeof(NPCData));
}
```

```
68
```

Structure memory layout

```
public struct NPCData
{
        public Quaternion BodyOrientation;
        public Quaternion HeadOrientation;
        public Vector3 Position;
        public int Health;
        public int Damage;
        public byte IsPositionCurrent;
        public byte IsOrientationCurrent;
        public byte HasEverTakeDamage;
        public byte HasEverShot;
}
```

Structure memory layout

```
unsafe
{
        Debug.Log(sizeof(NPCData));
}
```

```
56
```

Structure memory layout

12 bytes less - why?

The answer is **memory layout**

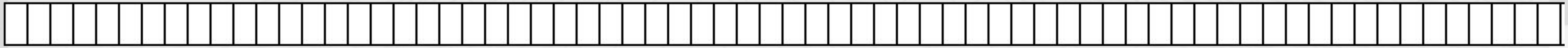
C# goes **from top to bottom** and puts the elements of structs into memory in that order.

Every data type has a **natural alignment**, which must be respected in order to permit the CPU to read and write memory effectively.

If it’s not aligned, instead of a simple read/write, the CPU has to read more blocks of memory, mask and shift them and then OR them together.

Structure memory layout

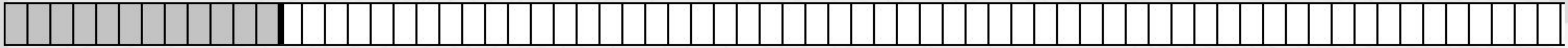
```
public struct WrongLayout
{
	public Vector3 Position;
	public byte IsPositionCurrent;
	public Quaternion BodyOrientation;
	public byte IsOrientationCurrent;
	public int Health;
	public byte HasEverTakenDamage;
	public int Damage;
	public byte HasEverShot;
	public Quaternion HeadOrientation;
}
```



Structure memory layout

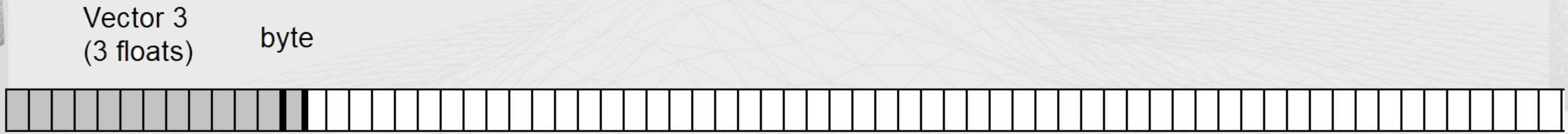
```
public struct WrongLayout
{
      → public Vector3 Position;
      public byte IsPositionCurrent;
      public Quaternion BodyOrientation;
      public byte IsOrientationCurrent;
      public int Health;
      public byte HasEverTakenDamage;
      public int Damage;
      public byte HasEverShot;
      public Quaternion HeadOrientation;
}
```

Vector 3
(3 floats)



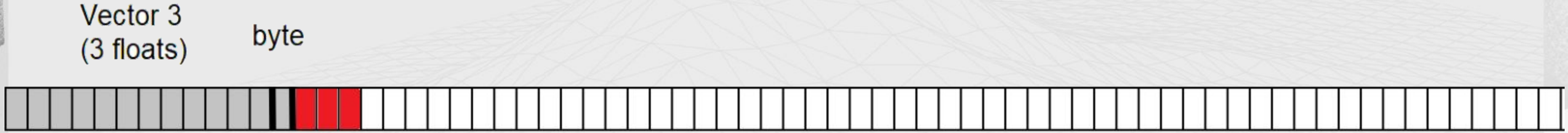
Structure memory layout

```
public struct WrongLayout
{
	public Vector3 Position;
	→ public byte IsPositionCurrent;
	public Quaternion BodyOrientation;
	public byte IsOrientationCurrent;
	public int Health;
	public byte HasEverTakenDamage;
	public int Damage;
	public byte HasEverShot;
	public Quaternion HeadOrientation;
}
```



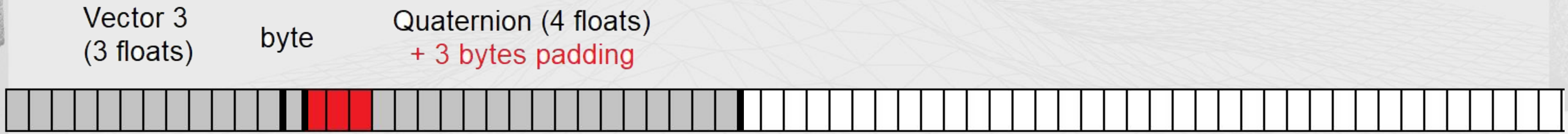
Structure memory layout

```
public struct WrongLayout
{
       public Vector3 Position;
       public byte IsPositionCurrent;
       → public Quaternion BodyOrientation;
       public byte IsOrientationCurrent;
       public int Health;
       public byte HasEverTakenDamage;
       public int Damage;
       public byte HasEverShot;
       public Quaternion HeadOrientation;
}
```



Structure memory layout

```
public struct WrongLayout
{
      public Vector3 Position;
      public byte IsPositionCurrent;
      → public Quaternion BodyOrientation;
      public byte IsOrientationCurrent;
      public int Health;
      public byte HasEverTakenDamage;
      public int Damage;
      public byte HasEverShot;
      public Quaternion HeadOrientation;
}
```



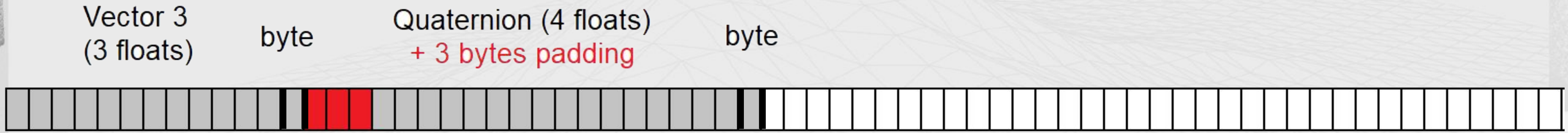
Vector 3
(3 floats)

byte

Quaternion (4 floats)
+ 3 bytes padding

Structure memory layout

```
public struct WrongLayout
{
        public Vector3 Position;
        public byte IsPositionCurrent;
        public Quaternion BodyOrientation;
        → public byte IsOrientationCurrent;
        public int Health;
        public byte HasEverTakenDamage;
        public int Damage;
        public byte HasEverShot;
        public Quaternion HeadOrientation;
}
```



Vector 3
(3 floats)

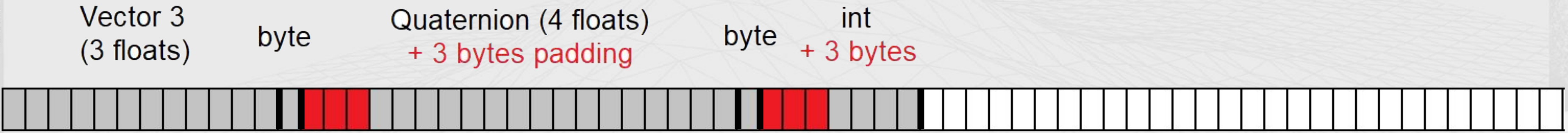
byte

Quaternion (4 floats)
+ 3 bytes padding

byte

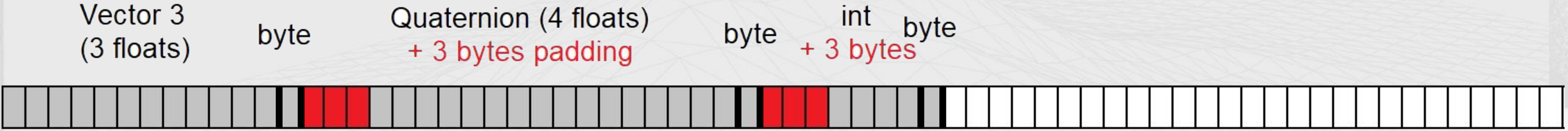
Structure memory layout

```
public struct WrongLayout
{
        public Vector3 Position;
        public byte IsPositionCurrent;
        public Quaternion BodyOrientation;
        public byte IsOrientationCurrent;
        → public int Health;
        public byte HasEverTakenDamage;
        public int Damage;
        public byte HasEverShot;
        public Quaternion HeadOrientation;
}
```



Structure memory layout

```
public struct WrongLayout
{
        public Vector3 Position;
        public byte IsPositionCurrent;
        public Quaternion BodyOrientation;
        public byte IsOrientationCurrent;
        public int Health;
        → public byte HasEverTakenDamage;
        public int Damage;
        public byte HasEverShot;
        public Quaternion HeadOrientation;
}
```



Vector 3
(3 floats)

byte

Quaternion (4 floats)
+ 3 bytes padding

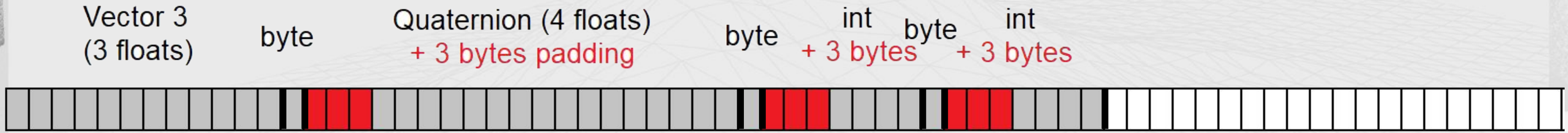
byte

int
+ 3 bytes

byte

Structure memory layout

```
public struct WrongLayout
{
        public Vector3 Position;
        public byte IsPositionCurrent;
        public Quaternion BodyOrientation;
        public byte IsOrientationCurrent;
        public int Health;
        public byte HasEverTakenDamage;
        → public int Damage;
        public byte HasEverShot;
        public Quaternion HeadOrientation;
}
```



Vector 3
(3 floats)

byte

Quaternion (4 floats)
+ 3 bytes padding

byte

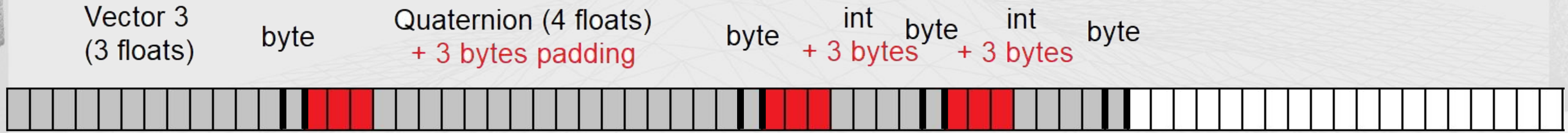
int
+ 3 bytes

byte

int
+ 3 bytes

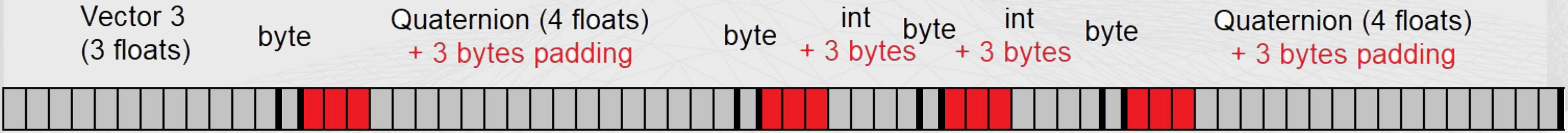
Structure memory layout

```
public struct WrongLayout
{
        public Vector3 Position;
        public byte IsPositionCurrent;
        public Quaternion BodyOrientation;
        public byte IsOrientationCurrent;
        public int Health;
        public byte HasEverTakenDamage;
        public int Damage;
        → public byte HasEverShot;
        public Quaternion HeadOrientation;
}
```



Structure memory layout

```
public struct WrongLayout
{
      public Vector3 Position;
      public byte IsPositionCurrent;
      public Quaternion BodyOrientation;
      public byte IsOrientationCurrent;
      public int Health;
      public byte HasEverTakenDamage;
      public int Damage;
      public byte HasEverShot;
      → public Quaternion HeadOrientation;
}
```



Vector 3 (3 floats)

byte

Quaternion (4 floats) + 3 bytes padding

byte

int + 3 bytes

byte

int + 3 bytes

byte

Quaternion (4 floats) + 3 bytes padding

[StructLayout] attribute

[StructLayout(LayoutKind.Sequential/Explicit/Auto)]

Sequential: Default (from top to bottom)

Explicit: Define the layout yourself

- You can even implement unions with this!

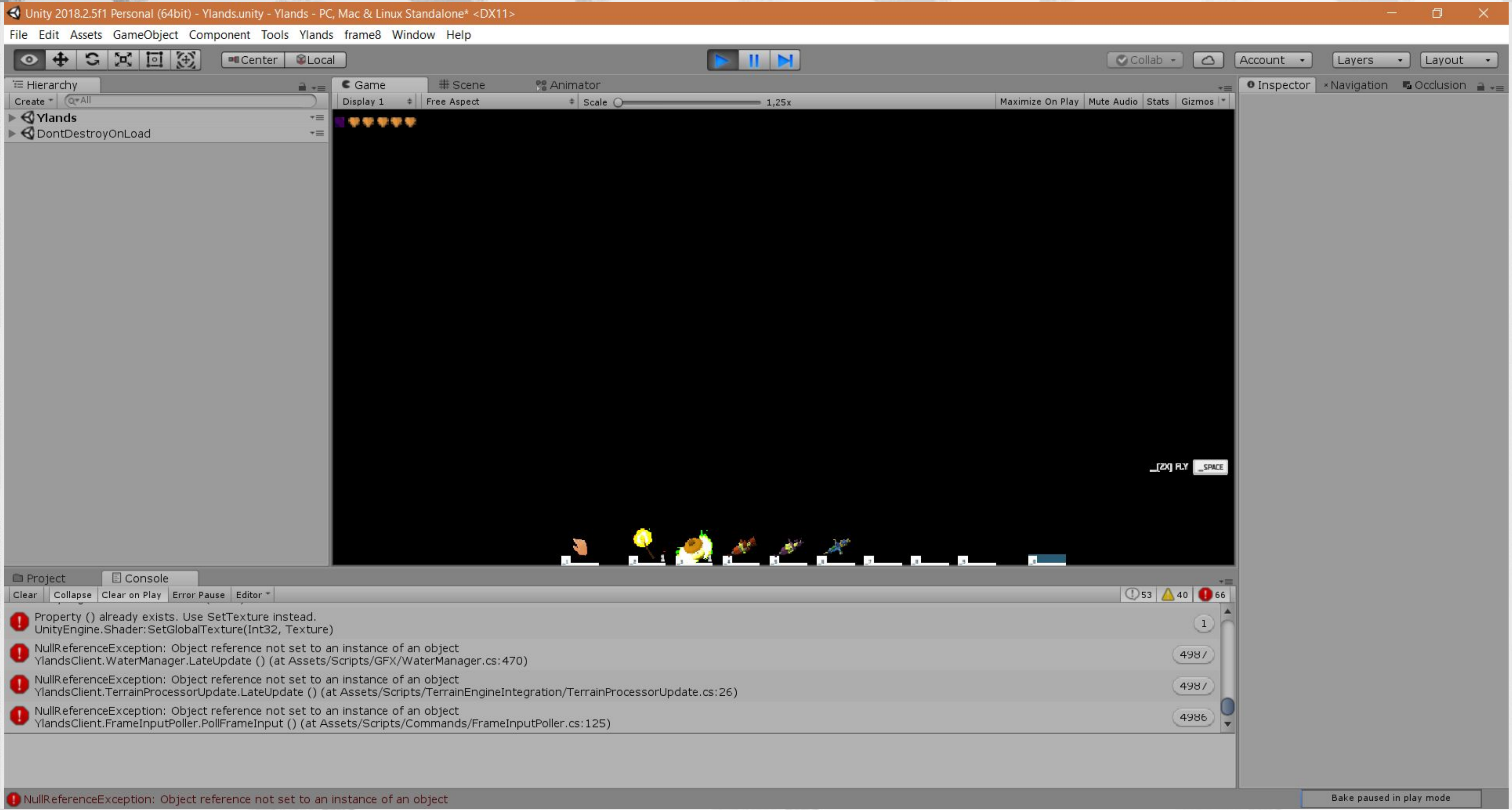
Auto: Let the compiler decide

- You can no longer expose that struct to native code

Unity assembly reload

Unity assembly reload

All of you have probably seen something like this...


Unity 2018.2.5f1 Personal (64bit) - Ylands.unity - Ylands - PC, Mac & Linux Standalone* <DX11>
File Edit Assets GameObject Component Tools Ylands frame8 Window Help
Center Local
Collab Account Layers Layout
Hierarchy
Create
Ylands
DontDestroyOnLoad
Game Scene Animator
Display 1 Free Aspect Scale 1,25x
Maximize On Play Mute Audio Stats Gizmos
Inspector Navigation Occlusion
_[ZX] FLY _SPACE
Project Console
Clear Collapse Clear on Play Error Pause Editor
53 40 66
Bake paused in play mode

```
Property () already exists. Use SetTexture instead.
UnityEngine.Shader:SetGlobalTexture(Int32, Texture)                                          1
NullReferenceException: Object reference not set to an instance of an object
YlandsClient.WaterManager.LateUpdate () (at Assets/Scripts/GFX/WaterManager.cs:470)          4987
NullReferenceException: Object reference not set to an instance of an object
YlandsClient.TerrainProcessorUpdate.LateUpdate () (at Assets/Scripts/TerrainEngineIntegration/TerrainProcessorUpdate.cs:26)   4987
NullReferenceException: Object reference not set to an instance of an object
YlandsClient.FrameInputPoller.PollFrameInput () (at Assets/Scripts/Commands/FrameInputPoller.cs:125)   4986

NullReferenceException: Object reference not set to an instance of an object
```

Unity assembly reload

Unity (Mono) compiles a new DLL containing your game.

It serializes data from the running game to the HDD.

It replaces the old DLL with the new DLL.

It deserializes data from the HDD back to RAM.

Only these are serialized

public, or [SerializeField] attribute

not static

not const

not readonly

a fieldtype that can be serialized

- this is a very limited subset

[Serializable] attribute used with structs and custom classes tells Unity that you want that serialized, too

However...

Serialization **doesn’t work very well** with custom classes

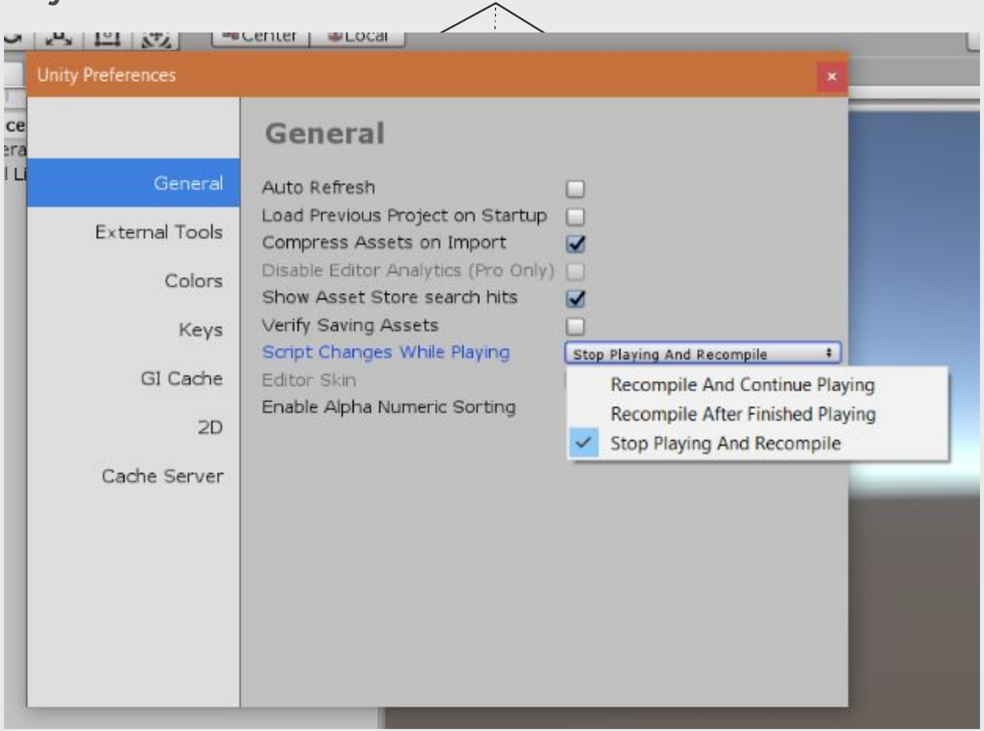
And it works even worse with custom classes that use **polymorphism**:

```
public Animal[] animals;
animals[0] = new Dog();
animals[1] = new Cat();
animals[2] = new Lizard();
```

After deserialization, we **lose the specific children** and only have **3 Animal** objects

A lazy, yet effective workaround

Since Unity 2018:



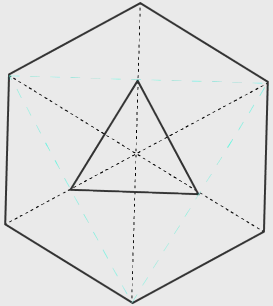
General

Auto Refresh □
Load Previous Project on Startup □
Compress Assets on Import ☑
Disable Editor Analytics (Pro Only) □
Show Asset Store search hits ☑
Verify Saving Assets □
Script Changes While Playing: Stop Playing And Recompile
Editor Skin
Enable Alpha Numeric Sorting

Recompile And Continue Playing
Recompile After Finished Playing
✓ Stop Playing And Recompile

A lazy, yet effective workaround

```
#if UNITY_EDITOR
[InitializeOnLoad]
public static class StopEditorOnRecompile
{
      static StopEditorOnRecompile()
      {
            if (EditorApplication.isPlaying)
            {
                  EditorApplication.isPlaying = false;
            }
      }
}
#endif
```



Data-oriented approach

Data-Oriented approach

Was introduced in Unity 2018.1 as an **experimental package**.

Contains three main aspects, which are supposed to greatly improve the performance of the result product.

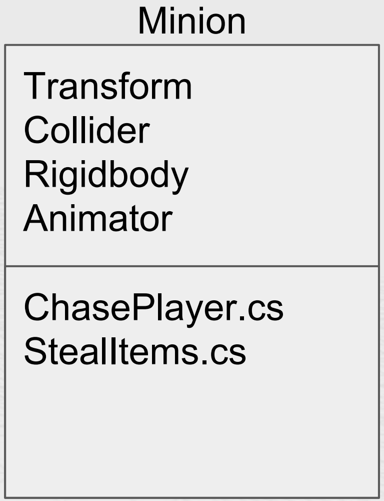
- **Entity-Component-System (ECS)**: will take care of the memory layout
- **C# Job System** : multi-threading
- **Burst compiler** : highly optimized machine code.

Entity Component System

ECS is a new architecture suggestion. Relies on the idea that instead of the OOP concept developers will start using a new Data oriented design.

Traditional approach:

- GameObjects + Components + MonoBehaviour



Minion

Transform
Collider
Rigidbody
Animator

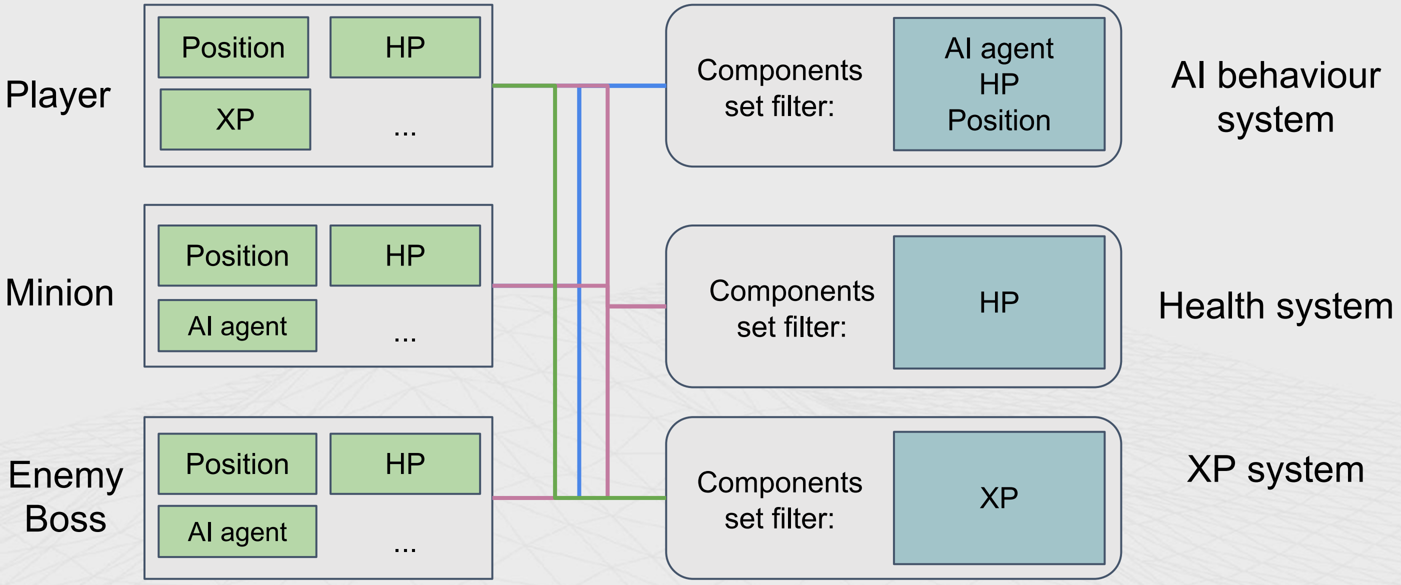
ChasePlayer.cs
StealItems.cs

Entity Component System

In ECS: divide the whole structure of the game into

- **Entities**: “ID”
- **Components**: structs that contain only the instance data for an Entity. Cannot contain methods.
- **Systems**: functionality/logic containers. Responsible for updating all Entities with a matching set of components.

Entity Component System



Previous slide inspired by:
https://software.intel.com/en-us/articles/get-started-with-the-unity-entity-component-system-ecs-c-sharp-job-system-and-burst-compiler

C# Job System

Makes it possible to take advantage of multi-core processors: manages multithreaded code by creating jobs instead of threads.

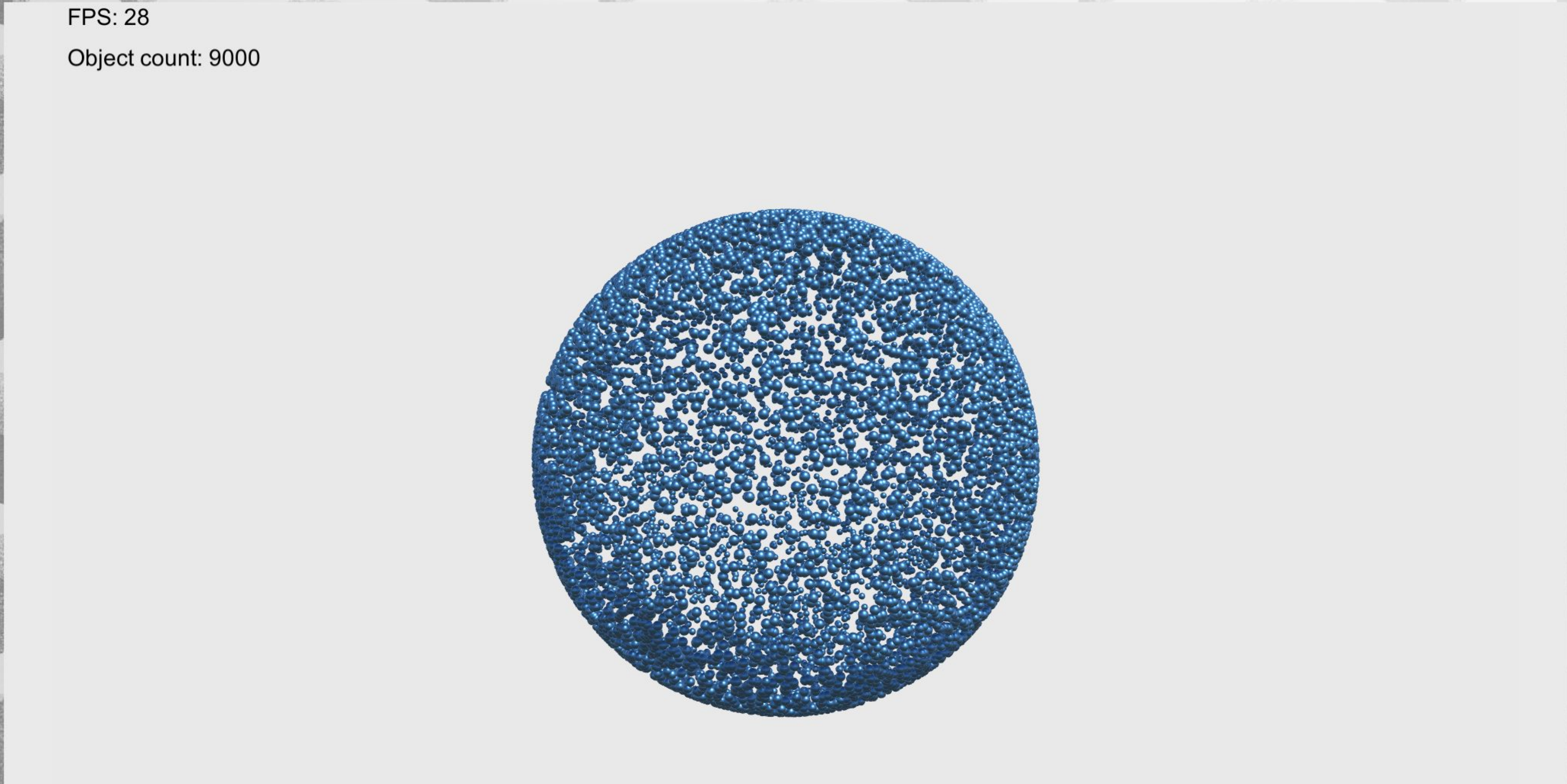
Job system relies on a set of working threads (one worker thread per logical CPU core). It puts jobs into a job queue where a working thread will execute them later.

Burst compiler

New compiler technology on producing highly optimized code.

Based on the LLVM technology.

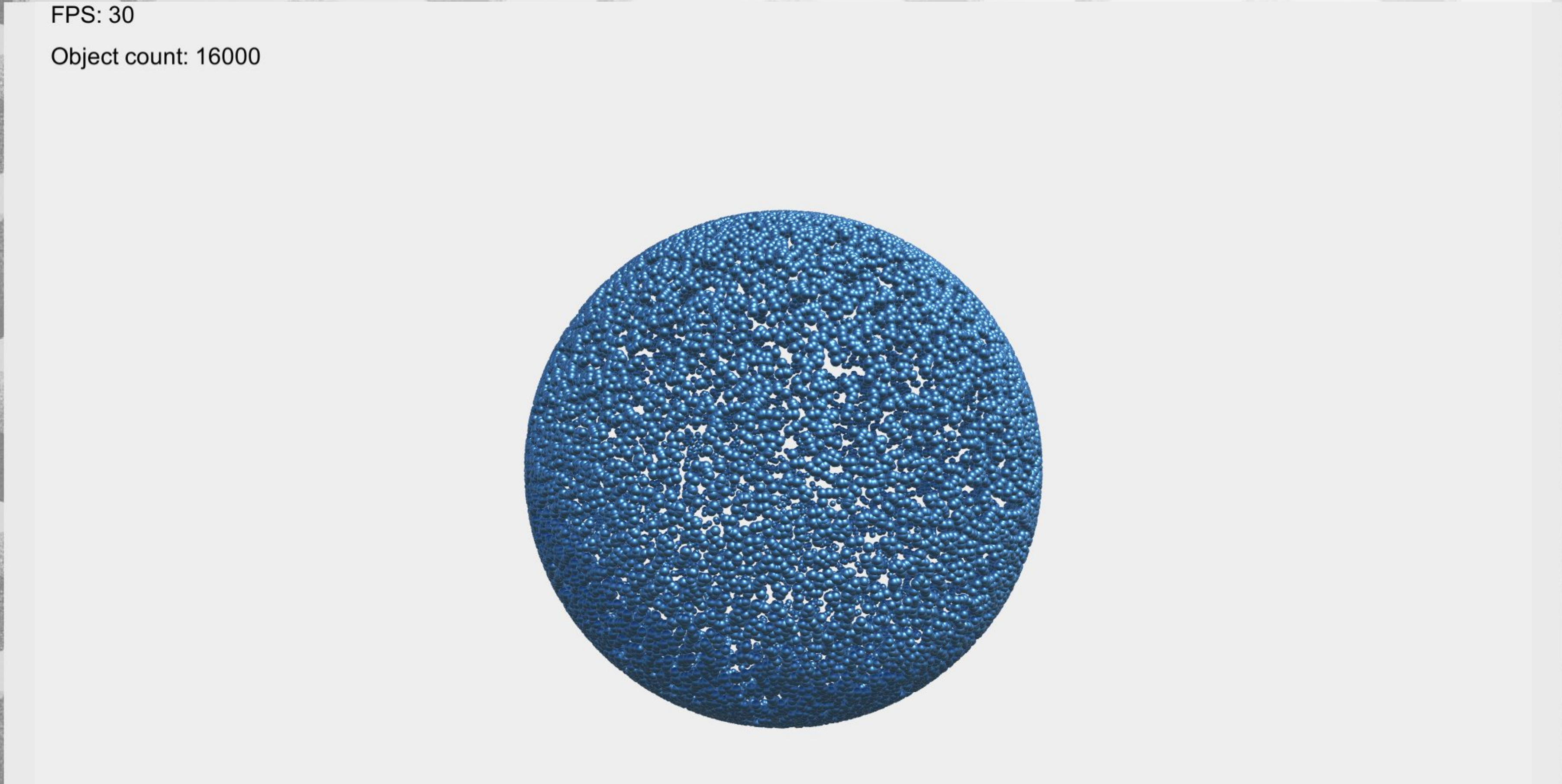
Burst compiler is relying on knowing, that all data has been set up the correct way with the new Entity-Component-System and Job System.



FPS: 28

Object count: 9000

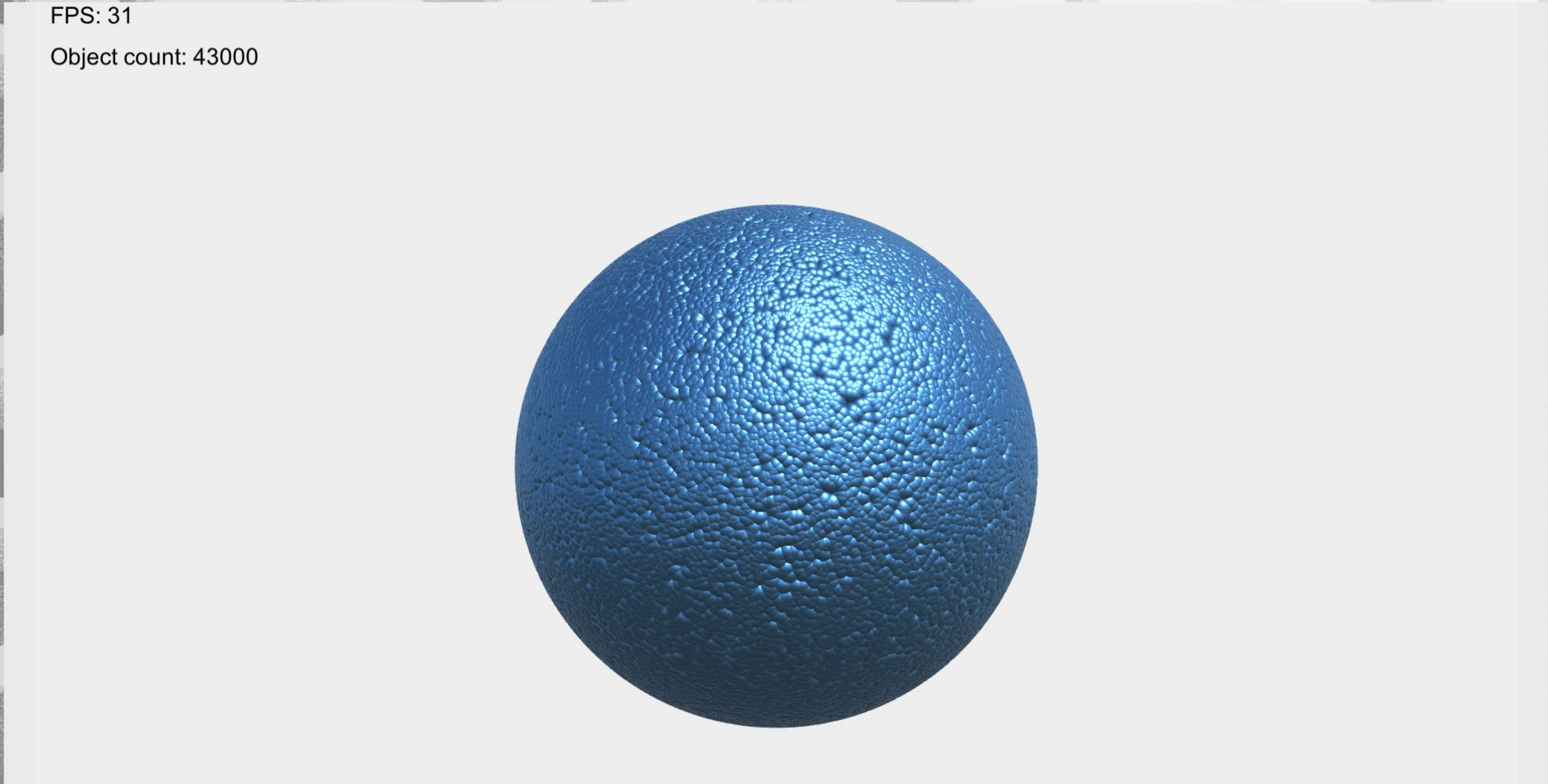
Traditional approach



FPS: 30

Object count: 16000

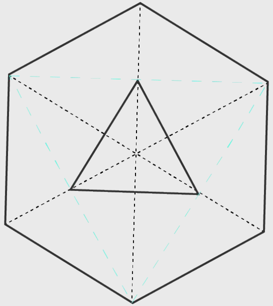
C# Job System



FPS: 31

Object count: 43000

ECS + Burst



Rendering millions objects

Rendering millions objects

Default Unity renderer with instancing

- Too high CPU overhead (completely unusable)

Custom mesh baking

- What Ylands initially used
- Huge memory footprint
- Significantly affects loading times
- Need to manually rebake after every object change

Rendering millions objects

Custom renderer

- Move workload to the GPU (great at parallel tasks)
- No Unity MeshRenderers, MeshFilters nor LodGroups
- Keep meshes and materials in custom data structures

Custom Renderer

Upload object data to **compute shader** (transforms, colors, etc.)

Frustum culling and LODing in compute shader **in parallel** (1 thread per obj)

Build **instance lists** (objects with the same mesh/material combo)

Issue instanced **indirect draw call** for each instance list

Custom renderer

Pros:

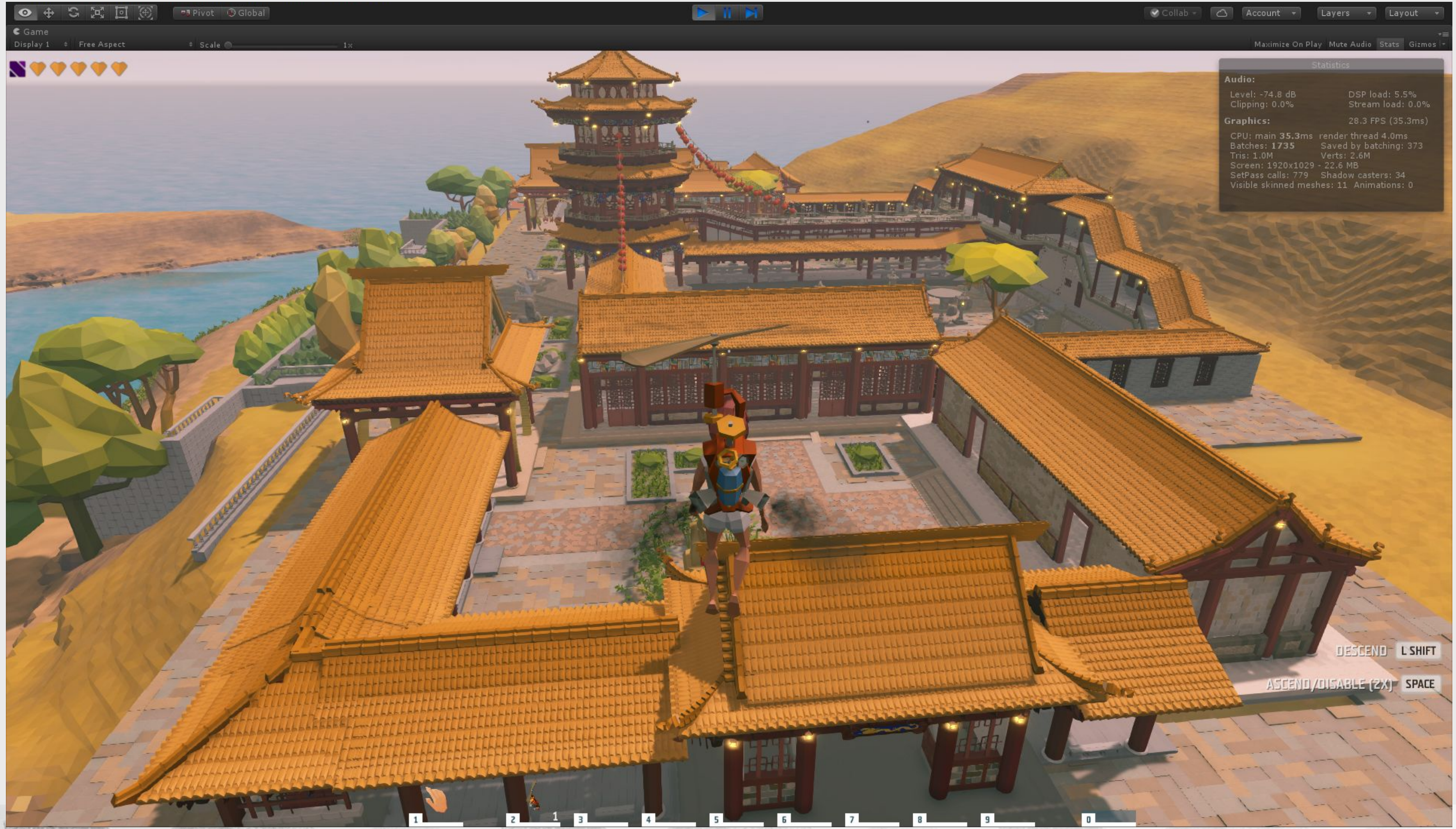
- Minimal CPU overhead because of greatly reduced draw call count
- Great GPU utilization
- GPU and CPU frame time up to 10x faster
- Scenes with millions of objects are possible

Custom renderer

Cons:

- Very tricky to implement
- Quite a few workarounds and hacks
- Makes sense only with scenes with a lot of objects

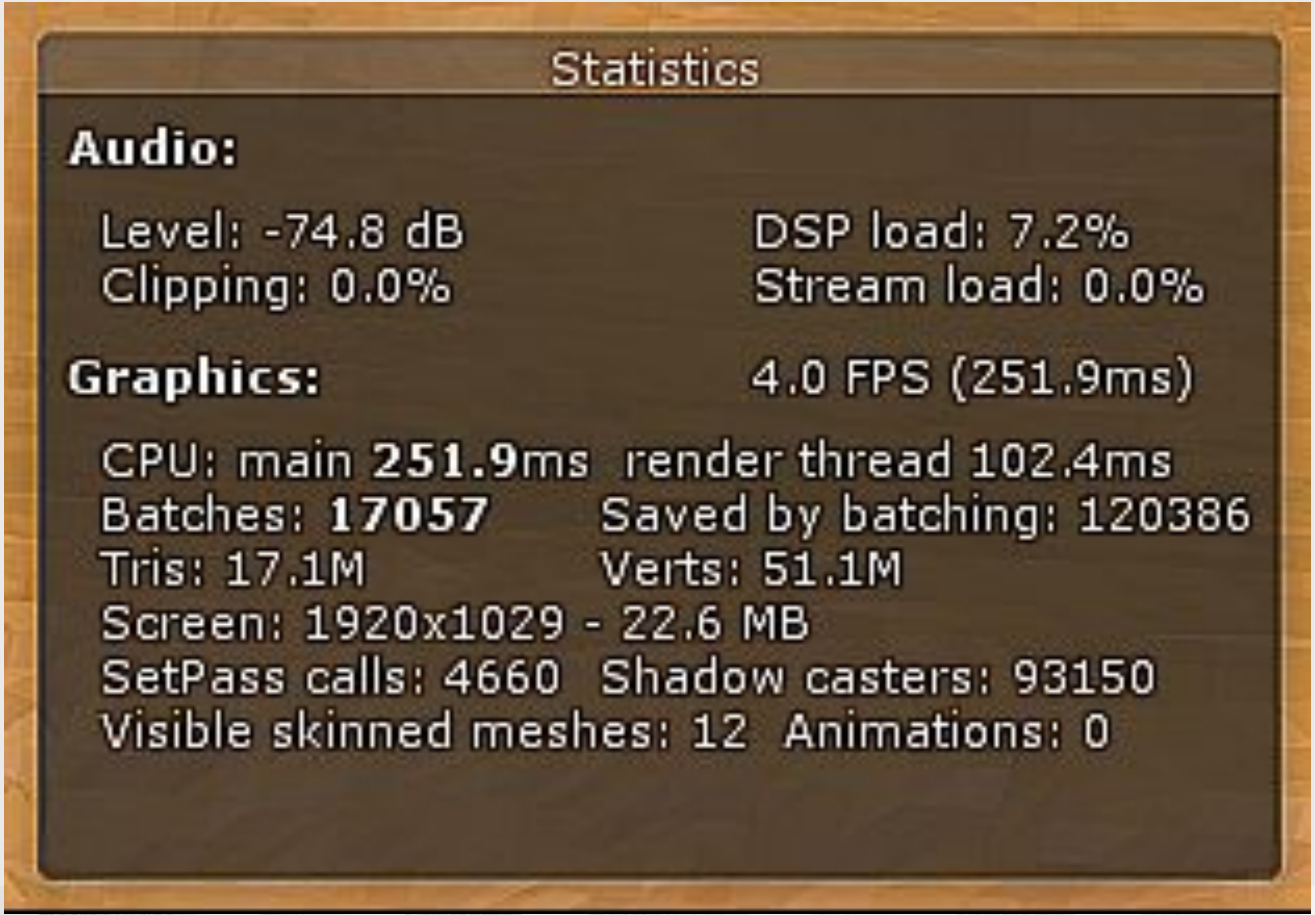
Several hundred thousand objects



```
Statistics
Audio:
  Level: -74.8 dB              DSP load: 5.5%
  Clipping: 0.0%               Stream load: 0.0%

Graphics:                      28.3 FPS (35.3ms)
  CPU: main 35.3ms  render thread 4.0ms
  Batches: 1735      Saved by batching: 373
  Tris: 1.0M         Verts: 2.6M
  Screen: 1920x1029 - 22.6 MB
  SetPass calls: 779    Shadow casters: 34
  Visible skinned meshes: 11  Animations: 0
```

Unity renderer



Statistics

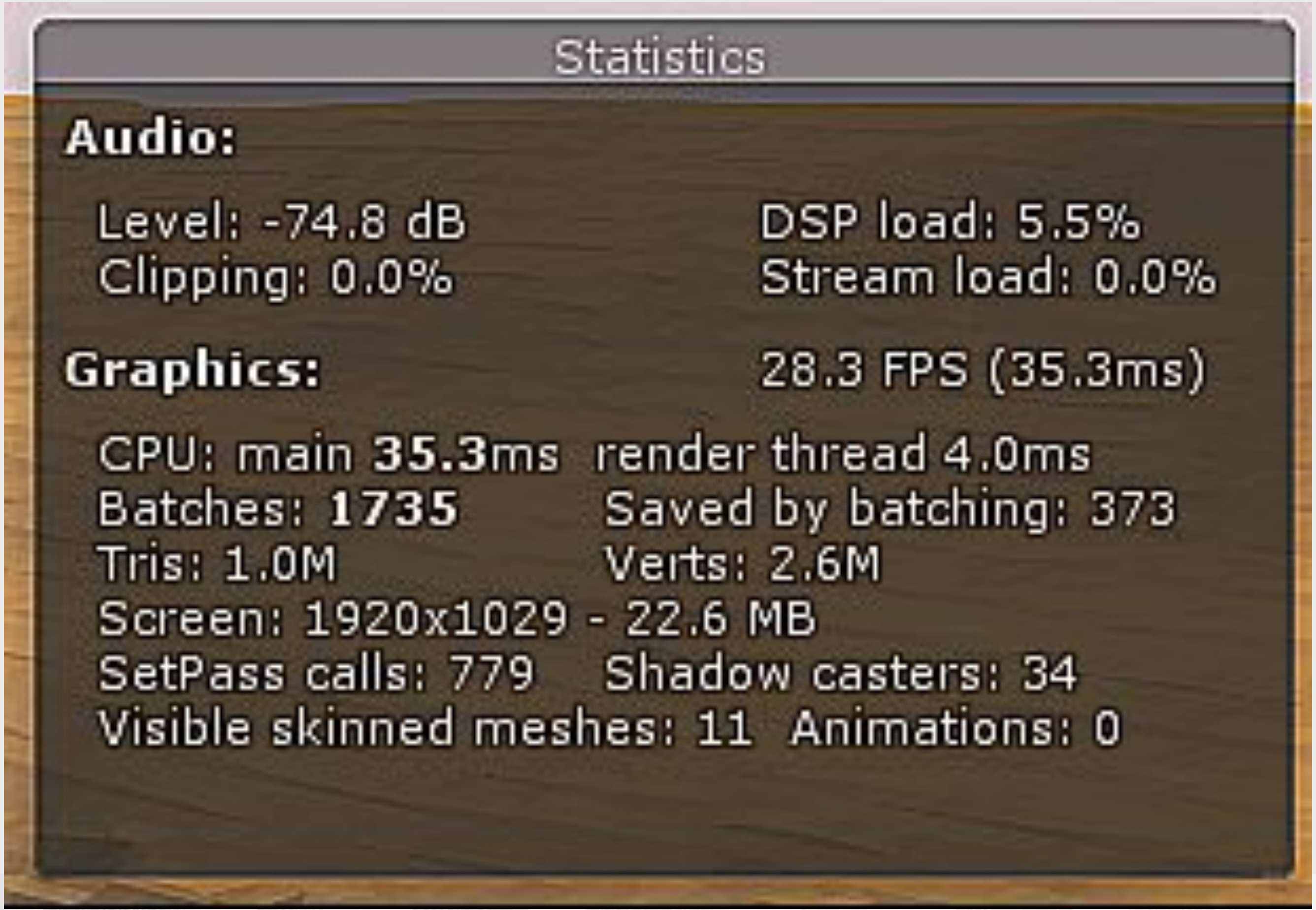
Audio:

Level: -74.8 dB DSP load: 7.2%
Clipping: 0.0% Stream load: 0.0%

Graphics: 4.0 FPS (251.9ms)

CPU: main **251.9**ms render thread 102.4ms
Batches: **17057** Saved by batching: 120386
Tris: 17.1M Verts: 51.1M
Screen: 1920x1029 - 22.6 MB
SetPass calls: 4660 Shadow casters: 93150
Visible skinned meshes: 12 Animations: 0

Our renderer



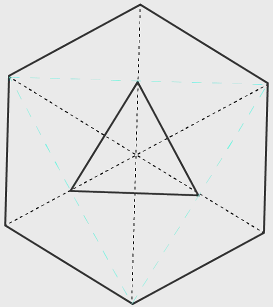
Statistics

Audio:

```
Level: -74.8 dB                  DSP load: 5.5%
Clipping: 0.0%                   Stream load: 0.0%
```

Graphics: 28.3 FPS (35.3ms)

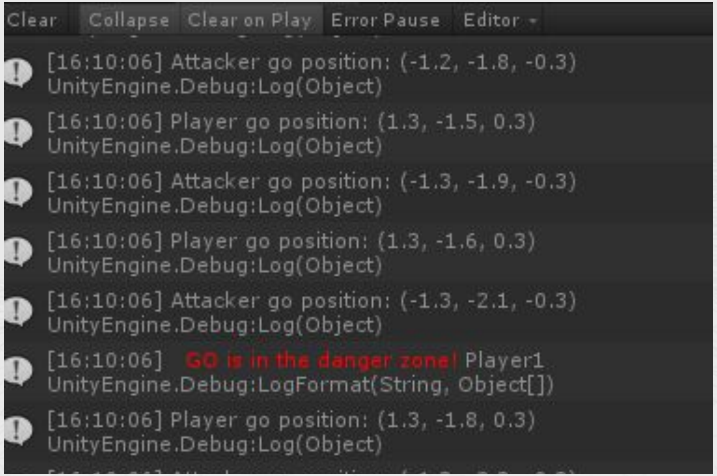
```
CPU: main 35.3ms  render thread 4.0ms
Batches: 1735         Saved by batching: 373
Tris: 1.0M            Verts: 2.6M
Screen: 1920x1029 - 22.6 MB
SetPass calls: 779    Shadow casters: 34
Visible skinned meshes: 11  Animations: 0
```



Short list of handy tricks

Short list of handy tricks

1. The Debug.Log method supports Rich Text markup tags



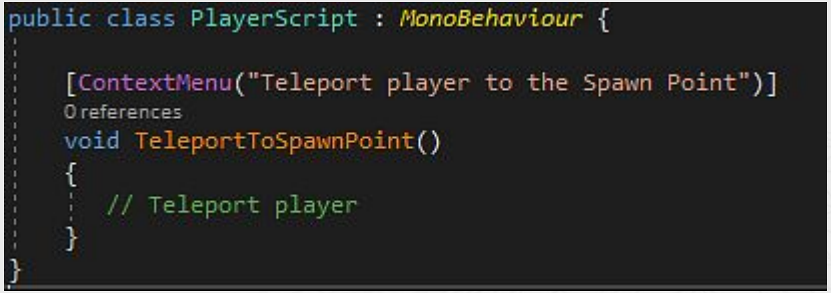
```
[16:10:06] Attacker go position: (-1.2, -1.8, -0.3)
UnityEngine.Debug:Log(Object)
[16:10:06] Player go position: (1.3, -1.5, 0.3)
UnityEngine.Debug:Log(Object)
[16:10:06] Attacker go position: (-1.3, -1.9, -0.3)
UnityEngine.Debug:Log(Object)
[16:10:06] Player go position: (1.3, -1.6, 0.3)
UnityEngine.Debug:Log(Object)
[16:10:06] Attacker go position: (-1.3, -2.1, -0.3)
UnityEngine.Debug:Log(Object)
[16:10:06]   GO is in the danger zone! Player1
UnityEngine.Debug:LogFormat(String, Object[])
[16:10:06] Player go position: (1.3, -1.8, 0.3)
UnityEngine.Debug:Log(Object)
```

Short list of handy tricks

2. Use go.CompareTag("tag") instead of go.tag == "tag"

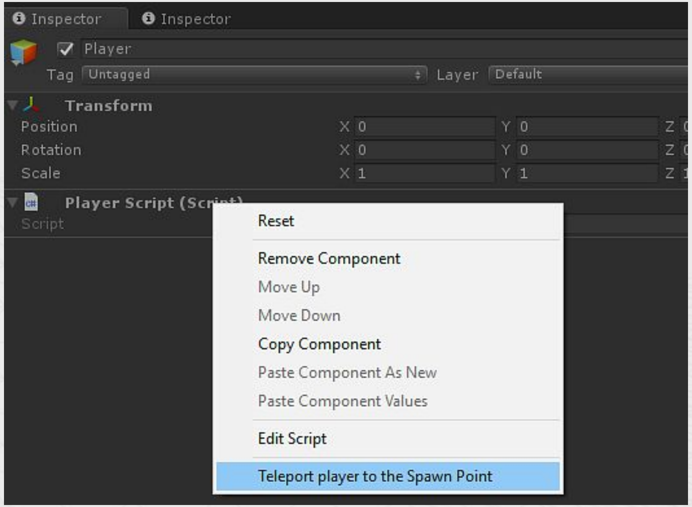
Short list of handy tricks

3. Mark your method with [ContextMenu] attribute to be able to call it with context menu of the component



```
public class PlayerScript : MonoBehaviour {

    [ContextMenu("Teleport player to the Spawn Point")]
    0 references
    void TeleportToSpawnPoint()
    {
        // Teleport player
    }
}
```



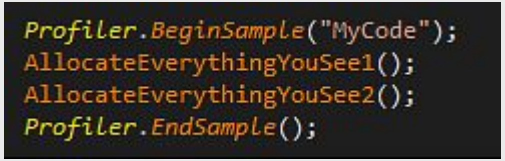
Player
Tag Untagged Layer Default
Transform
Position X 0 Y 0
Rotation X 0 Y 0
Scale X 1 Y 1
Player Script (Script)
Script

Short list of handy tricks

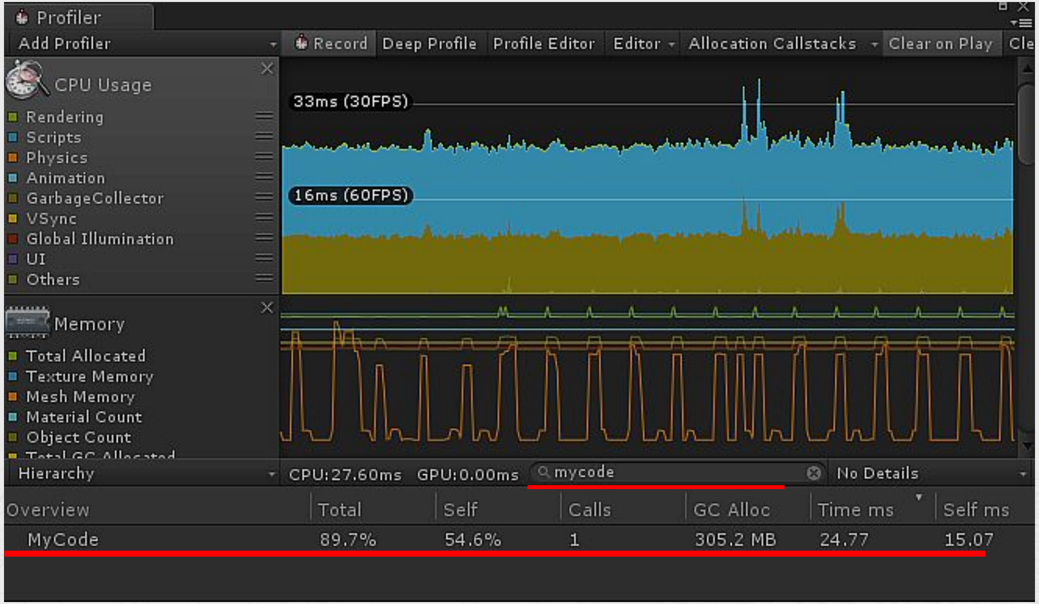
4. Use [FormerlySerializedAs("PreviousName")] in case you want to change a field name without losing its already serialized value.
    - It even supports multiple renames!

Short list of handy tricks

5. Measure your potentially performance demanding code in the profiler with "Profiler.BeginSample("MyCode:"); MyCode(); Profiler.EndSample();"



```
Profiler.BeginSample("MyCode");
AllocateEverythingYouSee1();
AllocateEverythingYouSee2();
Profiler.EndSample();
```



CPU:27.60ms GPU:0.00ms mycode

Overview	Total	Self	Calls	GC Alloc	Time ms	Self ms
MyCode	89.7%	54.6%	1	305.2 MB	24.77	15.07

THANK YOU FOR LISTENING

xeniya.valentova@bistudio.com
filip.vondrasek@bistudio.com

https://data.bistudio.com/download/gds2018.pdf

Follow us on:

- @bohemiainteract
- facebook.com/BohemiaInteractive/
- linkedin.com/company/bohemia-interactive/