COMP2007 - Game Development

Week 9 - Code session

During development, we need to measure the games performance on our machines and others.
Measuring a games performance can be the usage of memory, CPU, GPU or the frames per second the game is currently running at.
Unity gives us many tools for measuring performance:

- Stats dropdown
    - Very brief overview of performance per frame
    - Shows FPS, draw calls, memory usage and more
    - Available in Game view
- Profiler
    - In depth frame by frame measurements across all of unitys systems
    - Measures CPU, Memory, Rendering, audio, video, physics, networking, UI and lighting
- Frame debug
    - Shows the amount of draw calls per frame including textures, materials and shaders
- Physics debug
    - A debug overlay in the scene view for all physics objects

- Memory profiler (preview package)
    - Takes a snapshot of memory usage in game
    - Shows a complete list of the assets used and how much memory they take up

Scripting optimisations

Limiting code inside loops

You may have a for or while loop running inside of an update method.
This can be a cause for optimisation if the loop code is doing a lot of work.
You can use a condition to limit when code runs inside of a loop if it is not needed

```
public bool runLoopCode = false;

void Update()
{
    if (runLoopCode)
    {
        for (int i = 0; i < 10000; i++)
        {
            print("Good Loop Code");
        }
    }
}
```

Run code only when it changes

You may have events in your game that update a value over time like score, lives etc
It is often best to only update the values when required, rather than constantly, like in an update method
Add accessor methods which can update the changed value instead

Don’t do this!

```
public int lives = 3;

void Update()
{
    DisplayLives();
}

void DisplayLives()
{
    print("Lives: " + lives.ToString());
}
```

Do this instead

```
public int lives = 3;

void DisplayLives()
{
    print("Lives: " + lives.ToString());
}

public void AddLife()
{
    lives++;
    DisplayLives();
}

public void RemoveLife()
{
    lives--;
    DisplayLives();
}
```

Limit loop code to run every few frames

Some code is required to update at certain intervals, like a timer or often changing values like speed.
You can save some processing by only running them every so often in your update method.
You can use the Time class to calculate when to run the code.

NOTE: This can also be achieved with InvokeRepeating or a Coroutine, it depends on your use case!

The code inside of the condition will only run once per second
NOTE: Time.time is the time in seconds since the game was started

```
public float seconds;

void Update()
{
    if(seconds < Time.time)
    {
        seconds++;
        CustomMethod();
    }
}

void CustomMethod()
{
    print("Seconds: " + seconds.ToString("f0"));
}
```

Caching references

It’s best we store a reference to the component or class if we are going to use it often.
Unity’s GetComponent method actually searches the components on a GameObject to find it.
This causes a little bit of extra processing we want to avoid where we can.

NOTE: Use RequireComponent to guarantee a certain component is available

Here the Rigidbody component has a private field and is cached in the start method
Using the GetComponent method only once to store a reference to the Rigidbody
In the update method we can use the reference as normal

```
[RequireComponent(typeof(Rigidbody))]
public class UseReferencesGood : MonoBehaviour
{
    private Rigidbody body;

    void Start()
    {
        body = GetComponent<Rigidbody>();
    }

    void Update()
    {
        body.velocity = Vector3.zero;
    }
}
```

Unity Camera

If you need to get the main camera component, unity provides Camera.main to access it easily.
However, this is a slightly expensive operation to run, it is recommended to store a reference to the main camera before using it.

This code will do the operation once inside the start method, so the camera reference is ready to use

```
public Camera mainCamera;

void Start()
{
    mainCamera = Camera.main;
}
```

Expensive unity method calls

There are certain operations inside of unity that can be expensive.
Here are some tips for using them less often or avoiding them altogether.

SendMessage

SendMessage calls a method on a specific GameObject with optional parameters.
It will search the all GameObjects components until it finds the one with specified method name and parameter type.
This search is a little expensive to be doing often, especially in an update method like the image below

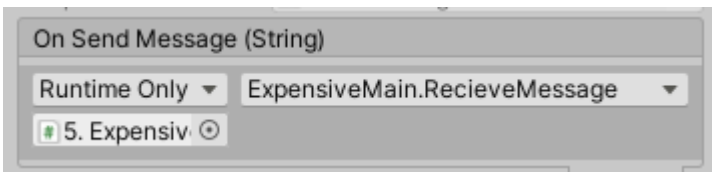
```
void Update()
{
    transform.parent.SendMessage("RecieveMessage", "Send Message Bad");
}
```

Use UnityEvents where possible in this situation.
UnityEvents specify which Component and method to send a message to, so they don't do a search behind the scenes.

NOTE: you can create a custom UnityEvent class to send your specific data types to, up to 4 values can be sent!

```
[System.Serializable]
public class SendMessageEvent : UnityEvent<string> { }

public class SendMessageGood : MonoBehaviour
{
    public SendMessageEvent onSendMessage;

    void Update()
    {
        onSendMessage?.Invoke("Send Message Good");
    }
}
```

The inspector view of the custom unityevent



Find

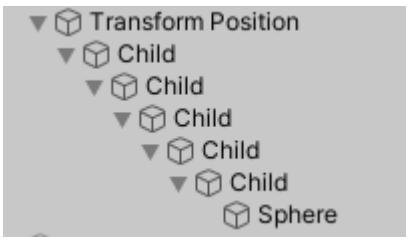
GameObject.Find is a convenient method for finding a specific GameObject in the hierarchy.
You can find by name, tag or any components on the GameObject.
This process of searching is expensive though, particularly if you have lots in the Hierarchy.
It's recommended that you run it once and cache any references.
Because the search may come up empty, you also may want to check for null objects to avoid errors.

```
private Transform target;

void Start()
{
    target = GameObject.Find("Find Cube").transform;
}

void Update()
{
    if (target != null)
    {
        target.Rotate(new Vector3(0, 0.1f, 0));
    }
}
```

Transform.position

When setting the position or rotation of a transform, you can use local or global
Transform.position is the global, transform.localPosition is the local.
When setting the global position, unity will also set the position of all child gameobjects, doing extra work.
This is an extra bit of work we can avoid using the local position in certain situations.

If we have a GameObject with a lot of children, we move all of them from the root GameObject.
Using Transform position will move each child object as well
Using Transform.localPosition will move the root, moving the children as well



Using this on the root object will move the children as well, without having to pre-calculate their positions

```
void Update()
{
    transform.localPosition += new Vector3(Time.deltaTime, 0, 0);
}
```

Remove unused update functions

Unity adds a connection to any update method added in your scripts.
When the game starts, each of these update methods will run, regardless of if there is no code present in the method.
This is a small processing overhead we can easily avoid.
If your code has empty update methods, remove them, this includes the following methods:

- Update
- LateUpdate
- FixedUpdate

Use OnBecameInvisible/OnBecameVisible to run code only when something is on screen

Renderer components can detect if they are seen by a camera and switch themselves off automatically.
We can detect this using the following MonoBehaviour methods:

- OnBecameVisible
    - Triggers when the Renderer is in view of any Camera
- OnBecameInvisible
    - Triggers when the Renderer is not in view

This can be used to stop any expensive methods in your code when the Renderer cannot be seen.

Example usage of the methods

```
private void OnBecameInvisible()
{
    print("Not Visible");
}

private void OnBecameVisible()
{
    print("VISIBLE");
}
```

Links

Rendering statistics
https://docs.unity3d.com/2020.2/Documentation/Manual/RenderingStatistics.html

Frame Debugger
https://docs.unity3d.com/2020.2/Documentation/Manual/FrameDebugger.html

Profiler window
https://docs.unity3d.com/2020.2/Documentation/Manual/ProfilerWindow.html

Physics debug visualisation
https://docs.unity3d.com/2020.2/Documentation/Manual/PhysicsDebugVisualization.html

Memory profiler (package docs)
https://docs.unity3d.com/Packages/com.unity.memoryprofiler@0.2/manual/index.html

Unity Learn extended article on optimisation
https://learn.unity.com/tutorial/fixing-performance-problems-2019-3?uv=2019.4&courseId=5c87de35edbc2a091bdae346#

Scripting

Update
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/MonoBehaviour.Update.html

FixedUpdate
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/MonoBehaviour.FixedUpdate.html

LateUpdate
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/MonoBehaviour.LateUpdate.html

Start
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/MonoBehaviour.Start.html

Time.time
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/Time-time.html

GetComponent
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/GameObject.GetComponent.html

RequireComponent
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/RequireComponent.html

Camera.main
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/Camera-main.html

SendMessage
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/GameObject.SendMessage.html

Transform.Rotate
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/Transform.Rotate.html

Time.deltaTime
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/Time-deltaTime.html

OnBecameVisible
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/MonoBehaviour.OnBecameVisible.html

OnBecameInvisible
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/MonoBehaviour.OnBecameInvisible.html

GAME DEV. TECH.

UNIVERSITY OF PLYMOUTH