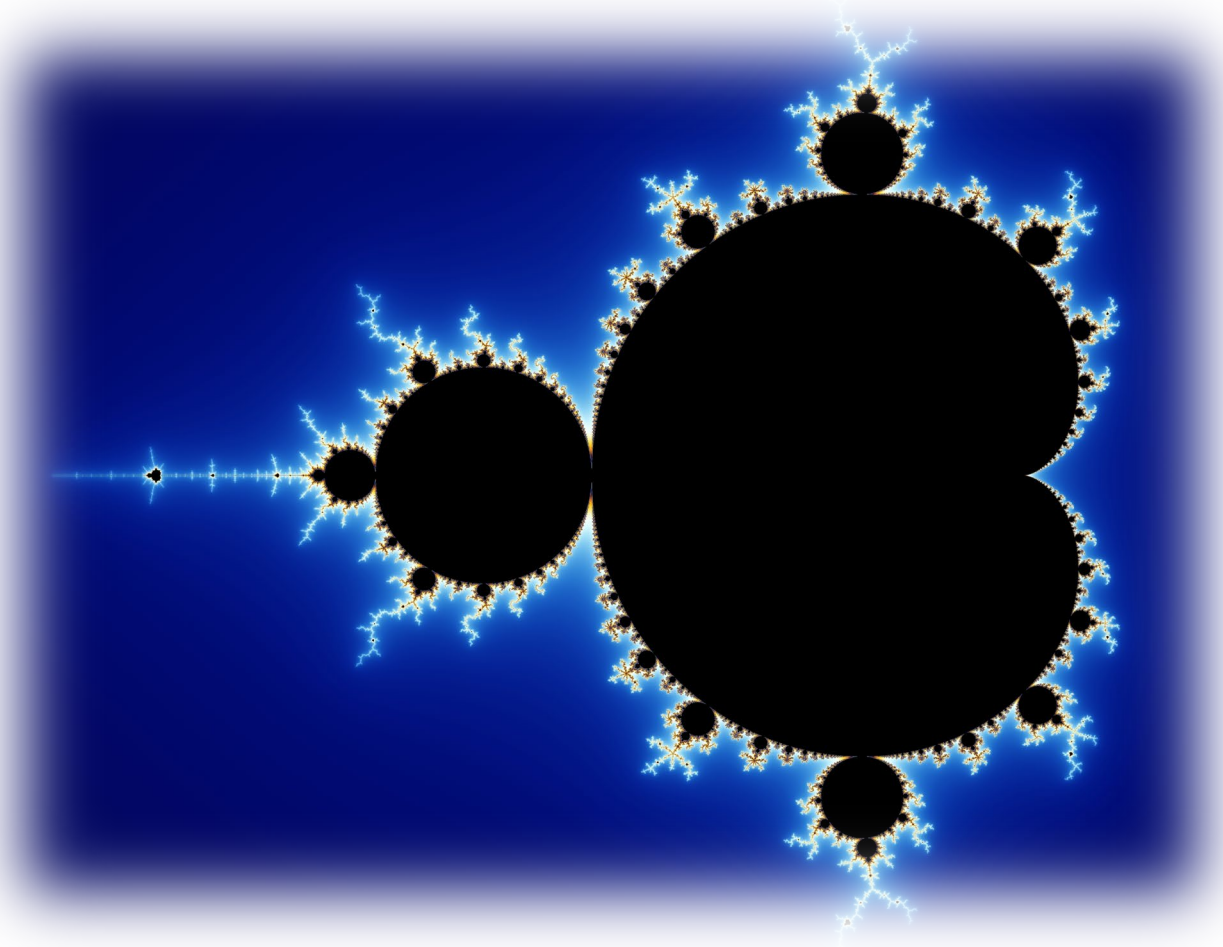


Unity C# programmers toolkit

Remote delivery session
https://tinyurl.com/yd4xkowx
Teams Code: **i1wy2k9**
Gist: https://tinyurl.com/y7louwvc

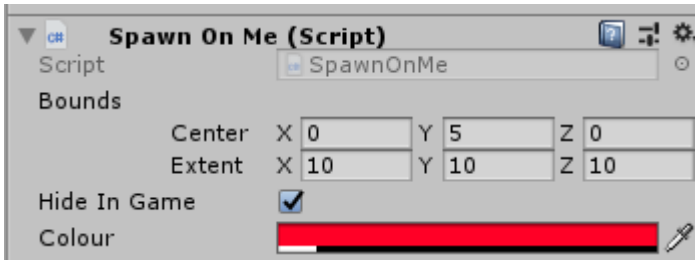
Making editing easier

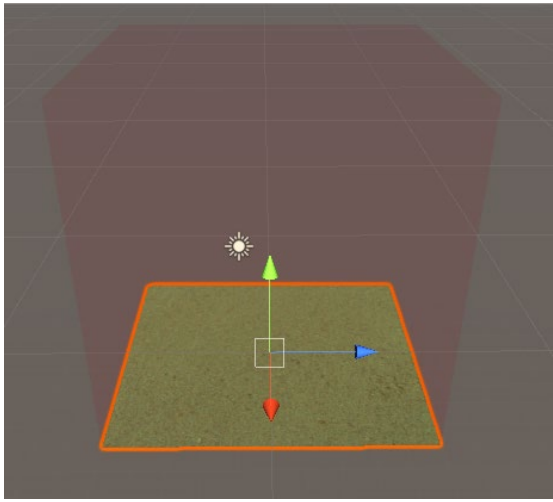
- Helper code to visualise functionality (see vs guess where it is)
- Running code in editor
- Code designed for reuse
- Optimisation
- GitHub Link https://github.com/RLTeachGit/UnityPools.git

Running code while in the editor

When editing visual elements it can be useful to show in edit mode

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

[ExecuteInEditMode] //Also run this code in editor, useful for dynamic editing
public class SpawnOnMe : MonoBehaviour
{
    [SerializeField]
    Bounds Bounds; //Can Be editied in IDE

    [SerializeField]
    bool HideInGame = true;  //Hide if playing, can be overridden in IDE

    [SerializeField]
    Color Colour=new Color(1.0f, 0, 0, 0.1f); //Default Gizmo colour

    //Draw Gizmo which show current spawning volume
    void OnDrawGizmosSelected()
    {
        if ((!Application.isPlaying) || (Application.isPlaying && !HideInGame)) //Only draw in editor or in game if not hidden
        {
            Gizmos.color = Colour;     //Draw spawning bounds in preset colour
            Gizmos.DrawCube(Bounds.center, Bounds.extents);
        }
    }
}
```





Gizmos useful for visualisation

https://docs.unity3d.com/ScriptReference/Gizmos.html

OnDrawGizmos()

- Draws Gizmos all the time

OnDrawGizmosSelected()

- Draws Gizmos on GO which are selected in hierarchy

Need to set colour

- Gizmos.color = Colour;

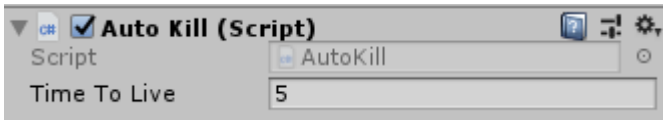
Static Methods

DrawCube	Draw a solid box with center and size.
DrawFrustum	Draw a camera frustum using the currently set Gizmos.matrix for it's location and rotation.
DrawGUITexture	Draw a texture in the Scene.
DrawIcon	Draw an icon at a position in the Scene view.
DrawLine	Draws a line starting at from towards to.
DrawMesh	Draws a mesh.
DrawRay	Draws a ray starting at from to from + direction.
DrawSphere	Draws a solid sphere with center and radius.
DrawWireCube	Draw a wireframe box with center and size.
DrawWireMesh	Draws a wireframe mesh.
DrawWireSphere	Draws a wireframe sphere with center and radius.

Getting rid of Objects

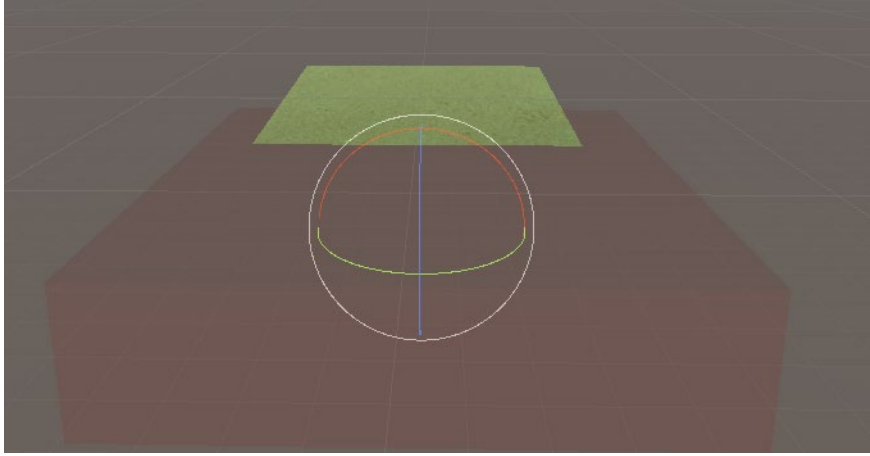
Timeout

- AutoKill, object only have finite Time to live




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

public class AutoKill : MonoBehaviour
{
    [SerializeField]
    float TimeToLive = 1.0f; //Set to Seconds to live

    void Start()
    {
        if(TimeToLive>0)
        {
            Destroy(gameObject, TimeToLive);
        }
    }
}
```

KillZones



Objects which enter a certain volume are killed

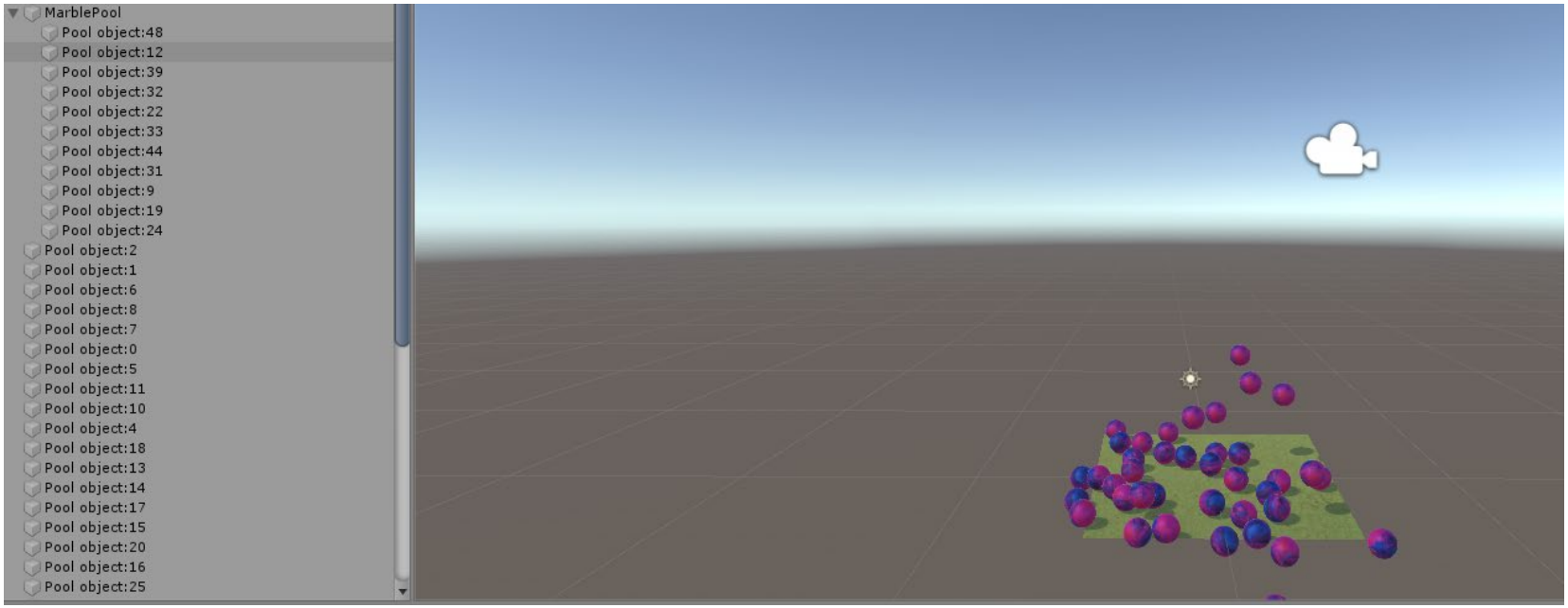
- Uses the bound visualiser from Spawner
- Must take care with [ExecuteInEditMode] to stop items being created while editing

```
BoxCollider mBC; //Cached BoxCollider

private void Start()
{
    if(Application.isPlaying) //Only call if playing, so its not created in edit
    {
        mBC = gameObject.AddComponent<BoxCollider>(); //Make & Add collider
        mBC.center = Bounds.center; //Take measurements from Bounds
        mBC.size = Bounds.size/2; //For some reason size is double
        mBC.isTrigger = true;
    }
}

private void OnTriggerEnter(Collider other)
{
    Destroy(other.gameObject);      //Kill all who enter
}
```

Object pooling

Its typically faster to turn object on/off rather then to create/destroy them

- With pooled objects you create them at the start & then hide them until needed
- By creating a set pool (per object type) you can keep the framerate relatively constant



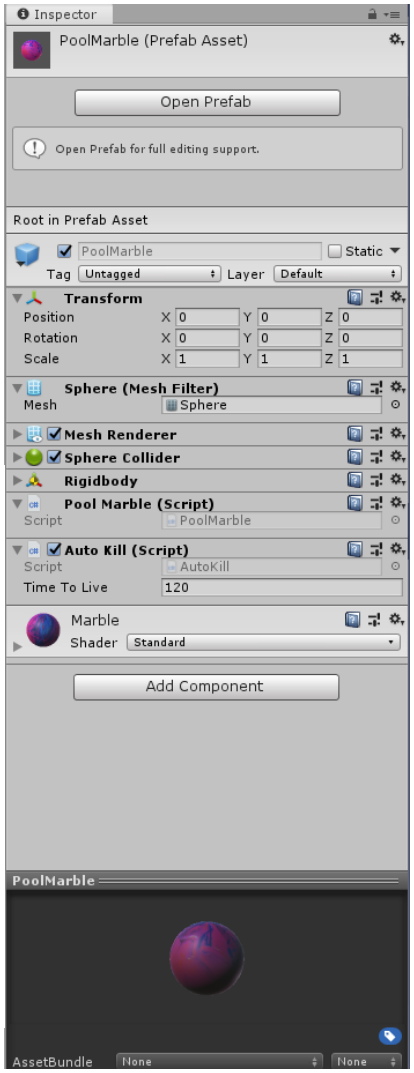
Pools should be object specific, such as bullets

In this case a pool of marbles

Each has a Script which handles pooling

- Based on the Abstract base class SpawnObject

```
public abstract class SpawnObject : MonoBehaviour //Base class for Spawn Object
{
    private   SpawnPool mPool;
    public  void    Create(SpawnPool vPool)
    {
        mPool = vPool; //Remember our pool
        OnCreate(); //Call child's OnCreate
    }
    public  void Reset(Vector3 vPosition, Quaternion vRotation)
    {
        OnReset(vPosition,vRotation);
    }
    protected abstract void OnCreate(); //Implemented in child to create object in first place
    protected abstract void OnReset(Vector3 vPosition, Quaternion vRotation); //Implemented in child to reset object when reused
    public abstract void Show(bool vShow = true); //Implemented in child
    public virtual void Remove()
    {
        mPool.PoolDestroy(this);        //Tell Pool we are done
    }
}
```



The Child has to implement its own pooling behaviour

As each pool object mat have different components which need to be turned on/off / initialised

- In this case to showing/hiding requires the renderer and Physics to be turned on/off
- Instead of Instantiate & Destroy we move the object between 2 List in the SpawnPool

```
List<SpawnObject> UnusedPool;  //Items free to use

List<SpawnObject> UsedPool;  //Used Items
```

```
public class PoolMarble : SpawnObject
{
    MeshRenderer mMR; //Cache Mesh Renderer
    Rigidbody mRB;

    protected override void OnCreate()
    {
        mMR = GetComponent<MeshRenderer>();
        mRB = GetComponent<Rigidbody>();
        Debug.Assert(mMR != null && mRB !=null);
    }

    protected override void OnReset(Vector3 vPosition, Quaternion vRotation)
    {
        transform.position = vPosition;
        transform.rotation = vRotation;
        Show();
    }

    public override void Show(bool vShow = true)
    {
        gameObject.SetActive(vShow); //Even better
    }

    public override void Remove()
    {
        base.Remove();   //Call Base to take care of moving back to Unused Pool
    }
}
```

Object hide/show

- When inactive we turn off what we can and hide the object
- When its needed it comes back to life
- The max number of objects
- Depends on the pool size

```
public  void    PoolDestroy(SpawnObject vSO)
{
    Debug.AssertFormat(vSO != null,"Invalid Object");


    if(UsedPool.Remove(vSO))
    {
        vSO.Show(false);        //Turn Off
        vSO.transform.SetParent(transform); //Parent to pool
        UnusedPool.Add(vSO);    //Add back to Unused Pool
    }
    else
    {
        Debug.LogFormat("Object {0} not in Used Pool", vSO.name);
    }
}
```


```
public  SpawnObject    PoolSpawn(Vector3 vPosition, Quaternion vRotation)
{
    if(UnusedPool.Count>0) //Do we have items to use
    {
        SpawnObject tSO = UnusedPool[0];      //Get first one from pool
        UnusedPool.RemoveAt(0);               //Remove it
        tSO.transform.SetParent(null);  //Unparent
        UsedPool.Add(tSO);                  //Add it to used Pool
        tSO.Reset(vPosition, vRotation);             //Reset to defaults
        Debug.LogFormat("SpawnObject {0} allocated from pool {1} remaining",tSO.name, UnusedPool.Count);
        return tSO;     //Success
    }
    Debug.LogFormat("Pool empty cannot allocate SpawnObject");
    return null;    //Pool Deleted
}
```

The pool manager can decide how to reallocate

For examples, reallocating objects which are far away or old

- The pool is a fixed resource and once exhausted one has to be deallocated before a new one is created
- However this can be decided by the object, in this case a child of PoolMarble which will reallocate if player not near

```
public class PoolMarbleRealloc : PoolMarble
{

    PlayerCast mPlayerCast;

    PlayerCast PlayerCast {
        get {
            if(mPlayerCast==null)
            {
                mPlayerCast = FindObjectOfType<PlayerCast>(); //Lazy Getter
            }
            return mPlayerCast;
        }
    }

    public override bool CanReallocate {
        get {
            if (!PlayerCast.PlayerVisible) return true; //If player cant be seen reallocate
            return ((PlayerCast.PlayerPosition - transform.position).magnitude > PlayerCast.Radius); //If player is far away reallocate
        }
    }
}
```

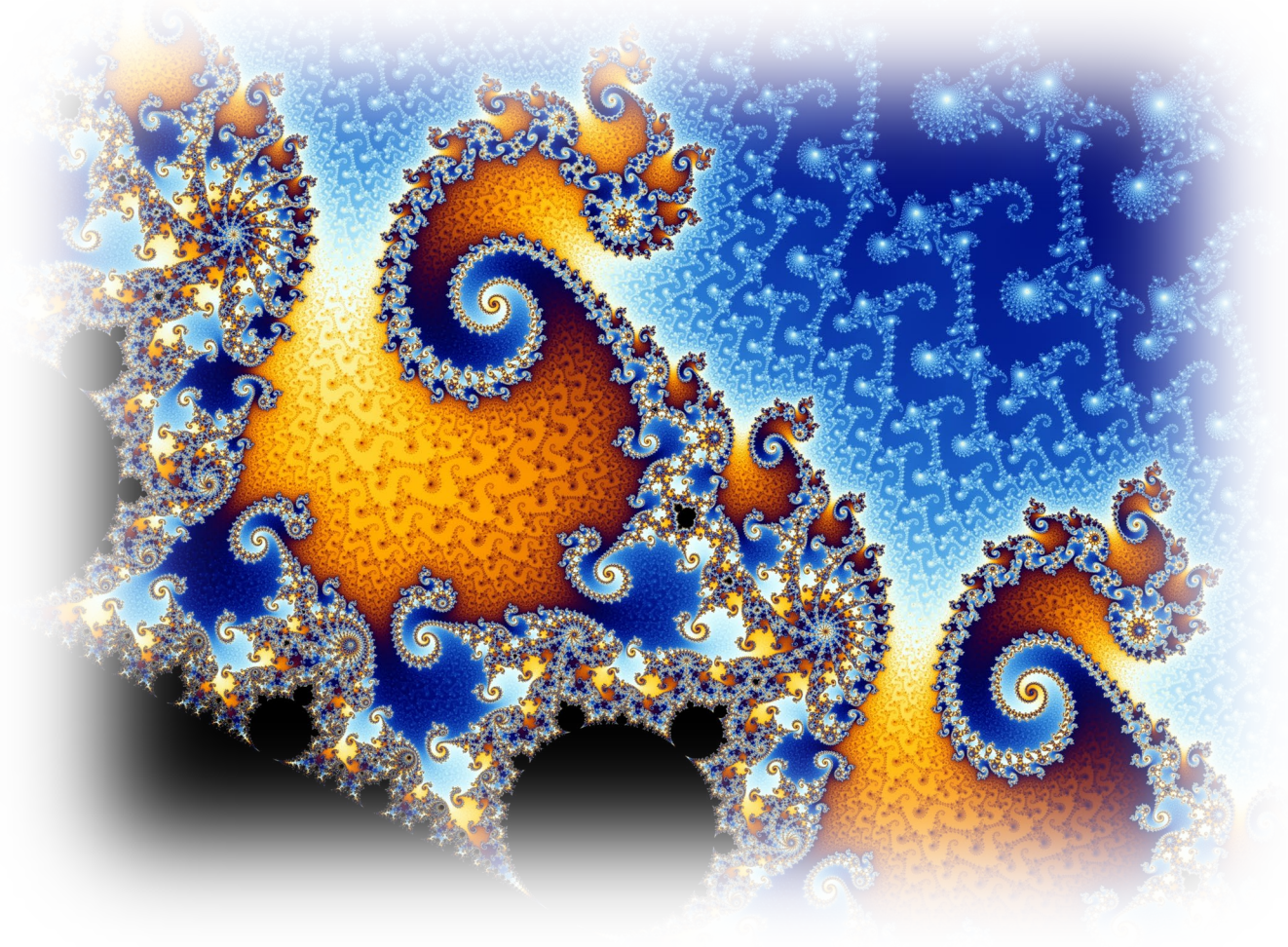
Lazy Getters

You can link GameObjects in the IDE, but this means lots of maintenance if things move

You could also use FindComponent<> to find it at runtime (SLOW)

A lazy getter will only look if its not already found it, so if its cached its fast, but it will only have to find it once if not

```
PlayerCast mPlayerCast;

PlayerCast PlayerCast {
    get {
        if(mPlayerCast==null)
        {
            mPlayerCast = FindObjectOfType<PlayerCast>(); //Lazy Getter
        }
        return mPlayerCast;
    }
}
```



Coursework Clinic Workshop

Remote delivery session
https://tinyurl.com/yd4xkowx
Teams Code: **i1wy2k9**
Gist: https://tinyurl.com/y7louwvc

Project coursework clinic

Group & individual project support

- Help with design, code & writeup