
GAM160: Further Games Programming

# 8: Memory

# Learning outcomes

- **Understand** Memory in modern object orientated languages
- **Compare** memory models in managed and unmanaged languages
- **Understand** the role of the profiler in measuring performance in games

FALMOUTH
UNIVERSITY

# Memory

# Memory Refresher

# Memory Refresher

- Recall that:

# Memory Refresher

- Recall that:
  - Dynamic memory, allocated on the **Heap** and is **growable**

# Memory Refresher

- Recall that:
  - Dynamic memory, allocated on the **Heap** and is **growable**
  - Static memory, allocated on the **Stack** and is **fixed size**

# Stack Memory

# Stack Memory

- When you allocate value types (int, float, short, char etc), these are allocated on the stack

# Stack Memory

- When you allocate value types (int, float, short, char etc), these are allocated on the stack
- Values allocated on the stack are local, when they drop out of scope they are deallocated

# Stack Memory

- When you allocate value types (int, float, short, char etc), these are allocated on the stack
- Values allocated on the stack are local, when they drop out of scope they are deallocated
- Values passed into functions are copied onto the stack

# Stack Memory

- When you allocate value types (int, float, short, char etc), these are allocated on the stack
- Values allocated on the stack are local, when they drop out of scope they are deallocated
- Values passed into functions are copied onto the stack
- The stack is of fixed size
  - C# - **1MB**

# Stack Memory Example 1

```
void Update()
{
    int x=10;
    int y=10;

    Vector2 pos=Vector2(x,y);
} //<-- x, y and pos drop out of scope here
```

# Heap Memory

# Heap Memory

- Otherwise known as dynamic memory

# Heap Memory

- Otherwise known as dynamic memory
- Types allocated with the **new** keyword are allocated on the heap

# Heap Memory

- Otherwise known as dynamic memory
- Types allocated with the **new** keyword are allocated on the heap
- This heap is managed by the garbage collector in C#

# Heap Memory Example 1 - C#

```
public class MonsterStats
{
    private int health;
    private int strength;

    public MonsterStats()
    {
        health=100;
        strength=10;
    }

    public void ChangeHealth(int h)
    {
        health+=h;
    }//<- h drops out of scope here

    void ChangeStrength(int s)
    {
        strength+=s;
    }//<- s drops out of scope here
}

void Start()
{
    //Create an instance of the class on the Heap
    MonsterStats new stats=MonsterStats();
    stats.ChangeHealth(10);
    stats.ChangeStrength(-2);
}
```

# Data Types in C#

- Value types include primitives such as int, bool, float etc
- Structs are custom value types (see example below)
- Reference types are anything declare with the **class**, **interface** & **delegate**
- In addition to this strings are also **reference types**
- Value types are allocated on the stack
- Reference type are allocated on the heap

# Struct Example - C#

```
public struct MonsterStats
{
    private int health;
    private int strength;

    public MonsterStats()
    {
        health=100;
        strength=10;
    }
}

void Start()
{
    //Create an instance of the struct on the stack
    MonsterStats stats=new MonsterStats();
    stats.ChangeHealth(10);
    stats.ChangeStrength(-2);
}
```

# Passing Variables

# Passing Variables

- In C#, when we call a method and pass some data as a parameter we either pass by value or we pass by reference.

# Passing Variables

- In C#, when we call a method and pass some data as a parameter we either pass by value or we pass by reference.
- We can mark a parameter with the **ref** or **out** keyword (see example)

# Passing Variables

- In C#, when we call a method and pass some data as a parameter we either pass by value or we pass by reference.
- We can mark a parameter with the **ref** or **out** keyword (see example)
- If you pass a variable using **ref** keyword it has to be initialised before hand

# Passing Variables

- In C#, when we call a method and pass some data as a parameter we either pass by value or we pass by reference.
- We can mark a parameter with the **ref** or **out** keyword (see example)
- If you pass a variable using **ref** keyword it has to be initialised before hand
- If you pass a variable using **out** keyword it can be initialised in a function

# Passing Variables

- In C#, when we call a method and pass some data as a parameter we either pass by value or we pass by reference.
- We can mark a parameter with the **ref** or **out** keyword (see example)
- If you pass a variable using **ref** keyword it has to be initialised before hand
- If you pass a variable using **out** keyword it can be initialised in a function
- Reference types are always passed by reference, you do not need to use the **keyword**

# Passing Example 1 - C#

```
int x=10;

void Adder(ref int value,int v)
{
    value+=v;
}

Adder(ref x,10);
//x would now be 20 after this
```

# Passing Example 2 - C#

```
void SetupMonster(ref MonsterStats stats, int health, int strength)
{
    //if we use the ref keyword MonsterStats has to be initialised
    stats.health=health;
    stats.strength=strength;
}

void CreateMonster(out MonsterStats stats,int health, int strength)
{
    //when we use out, it means we can initialise inside the function
    stats=new MonsterStats();
    stats.health=health;
    stats.strength=strength;
}

//Calling code
MonsterStats goblinStats=new MonsterStats();
SetupMonster(ref goblinStats,10,2);

MonsterStats orcStats;
CreateMonster(out orcStats,20,4);
```

FALMOUTH
UNIVERSITY

# Strings

# Strings

- Strings act like value types but they are actually reference types (C#)
- This means we need to be careful in allocating new strings
- **And** doing any operations using strings such as concatenation using +
- In C# you should use the **StringBuilder** class

# String Builder Example - C#

```
//We need to use the namespace - System.Text
using namespace System.Text

//Create the string builder with a capacity of 1024
StringBuilder sb=new StringBuilder(1024,1024);

//Append some text
sb.Append("Name: ");
sb.Append("Brian");
sb.Append(" Health: ");
sb.Append(100);

//Get the String from the String Builder
string s=sb.ToString();
```

FALMOUTH
UNIVERSITY

# Memory Management

# Garbage Collection in C#

# Garbage Collection in C#

- In C# there is an inbuilt Garbage Collector which will walk through the Object Graph

# Garbage Collection in C#

- In C# there is an inbuilt Garbage Collector which will walk through the Object Graph
- It will check to see if the object is still allocated, if not, the Garbage Collector will cleanup the object

# Garbage Collection in C#

- In C# there is an inbuilt Garbage Collector which will walk through the Object Graph
- It will check to see if the object is still allocated, if not, the Garbage Collector will cleanup the object
- This process is automatic and tuned for maximum performance

# Garbage Collection in C#

- In C# there is an inbuilt Garbage Collector which will walk through the Object Graph
- It will check to see if the object is still allocated, if not, the Garbage Collector will cleanup the object
- This process is automatic and tuned for maximum performance
- `https://unity3d.com/learn/tutorials/topics/performance-optimization/optimizing-garbage-collection-unity-games`

# Memory - General Good Practices

# Memory - General Good Practices

- Avoid allocate or deallocate during game update (this includes Instantiate)

# Memory - General Good Practices

- Avoid allocate or deallocate during game update (this includes Instantiate)
- Use Memory Pools to recycle objects, especially objects which act like bullets or units

# Memory - General Good Practices

- Avoid allocate or deallocate during game update (this includes Instantiate)
- Use Memory Pools to recycle objects, especially objects which act like bullets or units
  - Unity - `https://unity3d.com/learn/tutorials/topics/scripting/object-pooling`

# Memory - General Good Practices

- Avoid allocate or deallocate during game update (this includes Instantiate)
- Use Memory Pools to recycle objects, especially objects which act like bullets or units
  - Unity - `https://unity3d.com/learn/tutorials/topics/scripting/object-pooling`
  - C++ - `http://gameprogrammingpatterns.com/object-pool.html`

# Memory - General Good Practices

- Avoid allocate or deallocate during game update (this includes Instantiate)
- Use Memory Pools to recycle objects, especially objects which act like bullets or units
  - Unity - `https://unity3d.com/learn/tutorials/topics/scripting/object-pooling`
  - C++ - `http://gameprogrammingpatterns.com/object-pool.html`
- Cache frequently used objects

FALMOUTH
UNIVERSITY

# ScriptableObjects

# Scriptable Objects

- Is a class which allows you to store data separate from Script instances

# Scriptable Objects

- Is a class which allows you to store data separate from Script instances
- You have to inherit from the **ScriptableObject** class

# Scriptable Objects

- Is a class which allows you to store data separate from Script instances
- You have to inherit from the **ScriptableObject** class
- All properties are serialized and can be stored in a asset

# Scriptable Objects

- Is a class which allows you to store data separate from Script instances
- You have to inherit from the **ScriptableObject** class
- All properties are serialized and can be stored in a asset
- This means that you can add an instance of your ScriptableObject to a script and then assign the asset to it

# Scriptable Objects

- Is a class which allows you to store data separate from Script instances
- You have to inherit from the **ScriptableObject** class
- All properties are serialized and can be stored in a asset
- This means that you can add an instance of your ScriptableObject to a script and then assign the asset to it
- This is great for storing stats, weapons etc

# Scriptable Objects

- Is a class which allows you to store data separate from Script instances
- You have to inherit from the **ScriptableObject** class
- All properties are serialized and can be stored in a asset
- This means that you can add an instance of your ScriptableObject to a script and then assign the asset to it
- This is great for storing stats, weapons etc
- TIP: You can decorate your Scriptable Object with a CreateAssetMenu Attribute. This will allow you to create the Scriptable Object via a menu item.

`https://unity3d.com/learn/tutorials/modules/beginner/live-training-archive/scriptable-objects`

# Scriptable Object Example

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

//This will add an entry to the Asset/Create menu
[CreateAssetMenu(fileName ="Item",menuName ="Items/ ↩
    Item",order =1)]
public class ItemScriptableObject : ScriptableObject
{
    public string Name;
    public float Quality;
    public float Strength;
    public Sprite Image;
}
```

FALMOUTH
UNIVERSITY

# Exercise

# Exercise - All Students

- Complete exercise from last week
- Then move onto this weeks (on next slides)

# String Exercise - All Students

- Download one of the following Projects
  - BA Students - `https://github.com/Falmouth-Games-Academy/GAM160-Exercises`
  - BSc Students - `https://github.com/Falmouth-Games-Academy/COMP140-Exercises`
- Replace all string processing with StringBuilder if using C# or StringStream if using C++

# Debugging Exercise - BA Students

- Watch the following video - `https://unity3d.com/learn/tutorials/topics/scripting/debugging-unity-games-visual-studio`
- Open DebugExercise from the GAM160 Exercises Repository
- Use the debugger to find answers to the questions which are shown as comments within the Start() function

# Debugging Exercise - BSc Students

- Read the following - `https://tutorials.visualstudio.com/vs-get-started/debugging`
- Open DebugExercise project
- Use the debugger to find answers to the questions Q1 and Q2 which are shown as comments within the main function