COMP140-GAM160: Games Programming

7: 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

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

FALMOUTH UNIVERSITY

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**
  - C++ Visual Studio - **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
```

Stack Memory Example 2

```
class MonsterStats
{
private:
    int health;
    int strength;
public:
    MonsterStats()
    {
        health=100;
        strength=10;
    };

    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 main()
{
    //Create an instance of the class on the stack
    MonsterStats stats=MonsterStats();
    stats.ChangeHealth(10);
    stats.ChangeStrength(-2);
}//<-- stats drops out of scope here
```

Heap Memory

- Otherwise known as dynamic memory
- Types allocated with the **new** keyword are allocated on the heap
- The new operator returns a reference to the type and can be allocated to a pointer (C++)
- This heap is managed by the programmer in C++ (see **delete** keyword) or the garbage collector in C#
- In C++ is very important that you delete anything allocated on the heap
- **for every new, you need a matching delete**
- In the Unreal Engine objects can be Garbage Collected

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);
}
```

FALMOUTH UNIVERSITY

Heap Memory Example 2 - C++

```
class MonsterStats
{
private:
    int health;
    int strength;
public:
    MonsterStats()
    {
        health=100;
        strength=10;
    }

    .....
}

void main()
{
    //Create an instance of the class on the Heap
    MonsterStats * stats=new MonsterStats();
    stats->ChangeHealth(10);
    stats->ChangeStrength(-2);

    if (stats)
    {
    delete stats;
    stats=nullptr;
    }
}
```

FALMOUTH UNIVERSITY

Data Types in C#

- Value types are primitives such as int, bool, float etc
- Structs are custom value types (see example)
- 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

FALMOUTH UNIVERSITY

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 Types

- 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)
- Reference types are always passed by reference
- In C++, we can also pass by value or reference. We can also pass in a pointer
- We can mark parameter with **&** to pass by Reference
- Larger data types should be passed by Pointer or Reference

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);
```

Passing Example 1 - C++

```
int x=10;

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

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

Passing Example 2 - C++

```
void SetupMonster(MonsterStats &stats, int health, ↩
     int strength)
{
    stats.health=health;
    stats.strength=strength;
}

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

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
- In C++ you should use the **stringstream** class

FALMOUTH
UNIVERSITY

Memory Management

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 is tuned for maximum performance
- However you should think of caching GetComponent calls using the **Start** or **Awake**

Memory Management in C++

- In C++ there is no Garbage Collection, you have to manually delete objects when no longer needed
- Worth repeating **for every new, you need a matching delete**

Static Keyword & Singletons

FALMOUTH
UNIVERSITY

Coffee Break

FALMOUTH
UNIVERSITY

Exercise