INFT3960 – Game Production

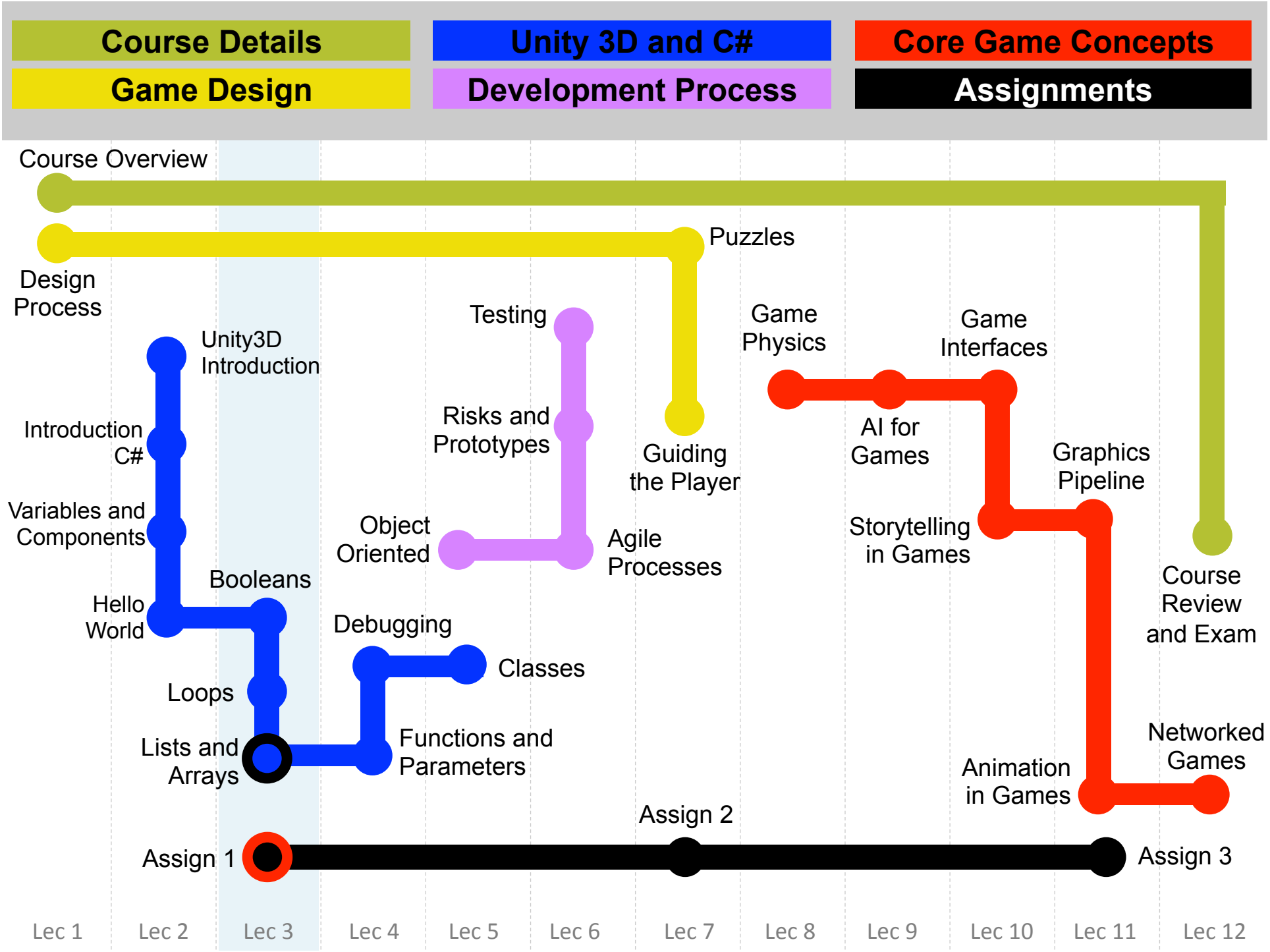
Week 03

Module 3.3

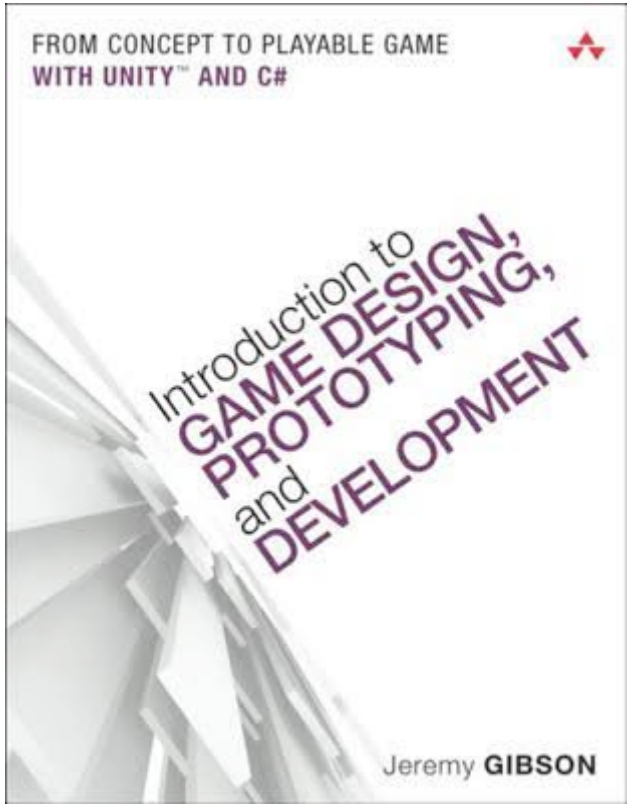
Lists and Arrays

Course Overview

Lec	Date	Modules	Assignments
1	Monday 24 Jul	Mod 1.1, Mod 1.2	
2	Monday 31 Jul	Mod 2.1, Mod 2.2, Mod 2.3, Mod 2.4	
3	Monday 7 Aug	Mod 3.1, Mod 3.2, Mod 3.3	 Assign 1 6 Aug, 11:00 am
4	Monday 14 Aug	Mod 4.1, Mod 4.2	
5	Monday 21 Aug	Mod 5.1, Mod 5.2	
6	Monday 28 Aug	Mod 6.1, Mod 6.2, Mod 6.3	
7	Monday 4 Sep	Mod 7.1, Mod 7.2	 Assign 2 4 Sep, 11:00 am
8	Monday 11 Sep	Mod 8.1	
9	Monday 2 Oct	Mod 9.1	
10	Monday 9 Oct	Mod 10.1, Mod 10.2	
11	Monday 16 Oct	Mod 11.1, Mod 11.2	
12	Monday 23 Oct	Mod 12.1, Mod 12.2	 Assign 3 23 Oct, 11:00am


Course Details
Unity 3D and C#
Core Game Concepts
Game Design
Development Process
Assignments
Course Overview
Puzzles
Design Process
Testing
Game Physics
Game Interfaces
Unity3D Introduction
Risks and Prototypes
AI for Games
Graphics Pipeline
Introduction C#
Guiding the Player
Variables and Components
Object Oriented
Agile Processes
Storytelling in Games
Course Review and Exam
Booleans
Hello World
Debugging
Classes
Loops
Functions and Parameters
Networked Games
Lists and Arrays
Animation in Games
Assign 2
Assign 1
Assign 3
Lec 1
Lec 2
Lec 3
Lec 4
Lec 5
Lec 6
Lec 7
Lec 8
Lec 9
Lec 10
Lec 11
Lec 12

Lists and Arrays – (Chapter 22)



LIST AND ARRAYS

Lists and Arrays – Topics

- C# Collections
- List
- Array
- Multidimensional Arrays
- Jagged Lists & Arrays
- foreach and Collections
- When to Use List or Array
- Other Collection Types

C# Collections

A collection in C# is a group of several things that are referenced by a single variable

Two most important C# collections are:

- `List`
- `Array`

Both `Lists` and `Arrays` do appear in the Inspector

`List` is the most flexible and easiest to use, so we'll start with `List`

List

Requires a new using line at the top of your script

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

List is a generic collection

Generic collections can work for any data type

```
List<string> sList;          // A List of strings

List<GameObject> goList;     // A List of GameObjects
```

List

A List must be defined before it can be used (because Lists default to null)

```
sList = new List<string>();
```

Elements are added to Lists using the Add() method

```
sList.Add("Hello");
sList.Add("World");
```

List

List elements are accessed via bracket access

Bracket access is zero indexed (i.e., starts at [0])

```
print( sList[0] );      // Prints: "Hello"
print( sList[1] );      // Prints: "World”
```

Lists have a Count of the number of elements

```
print( sList.Count );   // Prints: "2"
```

Lists can be cleared of all elements

```
sList.Clear();    // Empties sList
```

List

All these methods act on the List ["A","B","C","D"]

```
print( sList[0] );      // Prints: "A"
sList.Add("Apple");     // ["A","B","C","D","Apple"]
sList.Clear();          // []
sList.IndexOf("B");     // 1   ("B" is the 1st element)
sList.IndexOf("Bob");   // -1 ("Bob" is not in the List)
sList.Insert(2,"X");    // ["A","B","X","C","D"]
sList.Insert(4,"X");    // ["A","B","C","D","X"]
sList.Insert(5,"X");    // ERROR!!!  Index out of range
sList.Remove("C");      // ["A","B","D"]
sList.RemoveAt(1);      // ["A","C","D"]
```

Lists can be converted to Arrays

```
string[] sArray = sList.ToArray();
```

Array

`Array` is a much simpler collection than `List`

- Does not require any using statement at the top of a script

Not actually its own data type, but rather a collection of any data type

```
string[] sArray;        // An array of strings
GameObject[] goArray;   // An array of GameObjects
```

Arrays are created with a fixed length -Arrays cannot expand to add more elements like Lists can

```
sArray = new string[4];// An array of four strings
sArray = new string[] {"A","B","C","D"};
```

- Either of these arrays will only ever have a Length of 4

Array

Array elements are accessed and assigned via bracket access

```
sArray[1] = "Bob";  // Assigns "Bob" to the 1st element
print( sArray[1] ); // Prints: "Bob"
```

It's possible to skip elements in an array - the skipped elements are the default value for that type

```
string[] sArray = new string[4]; //4-element string array
sArray[0] = "A";        // ["A",null,null,null]
sArray[2] = "C";        // ["A",null,"C",null]
```

Arrays have `Length` instead of `Count`

```
print( sArray.Length );    // Prints: 4
```

- This is similar to strings (which are collections of chars)

Array

All these methods act on the sArray ["A","B","C","D"]

```
print( sArray[0] );     // Prints: "A"
sArray[2] = "Cow";      // ["A","B","Cow","D"]
sArray[10] = "Test";    // ERROR!!!   Index out of range
```

Static methods of the System.Array class

```
print( System.Array.IndexOf(sArray,"B") ); // 1
System.Array.Resize( ref sArray, 6);   // Sets Length to 6
```

Arrays can be converted to Lists

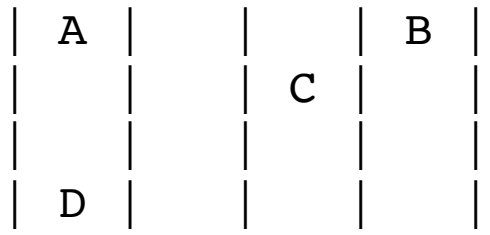
```
List<string> sList = new List<string>( sArray );
```

Multidimensional Arrays

Arrays can have more than one dimension

```
string[,] s2D = new string[4,4]; // Makes a 4x4 array
s2D[0,0] = "A";
s2D[0,3] = "B";
s2D[1,2] = "C";
s2D[3,1] = "D";
```

This would make the 2-dimensional array



Length is still the total length of the array

```
print( s2D.Length );   // Prints: 16
```

Multidimensional Arrays

```
string str = "";

for ( int i=0; i<4; i++ ) {
    for ( int j=0; j<4; j++ ) {
        if (s2D[i,j] != null) {
            str += "|" + s2D[i,j];
        } else {
            str += "|_";
        }
    }
    str += "|"+"\n";
}
print( str );
```

This prints:

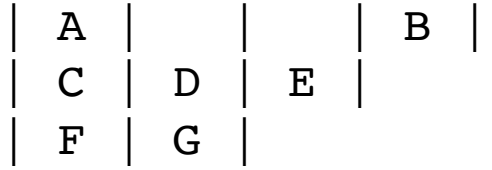
```
|A|_|_|B|
|_|_|C|_|
|_|_|_|_|
|D|_|_|_|
```

Jagged Lists and Arrays

Both Lists and arrays can be composed of other Lists or arrays

```
string[][] jArray = new string[3][];   //Makes a 3x? array
jArray[0] = new string[4];
jArray[0][0] = "A";
jArray[0][3] = "B";
jArray[1] = new string[] {"C","D","E"};
jArray[2] = new string[] {"F","G"};
```

This would make the jagged array



Length is now accurate for each part of the jagged array

```
print( jArray.Length );     // Prints: 4
print( jArray[1].Length );  // Prints: 3
```

`foreach` and Collections

Lists and arrays can be iterated over using foreach

```
string[] sArray = new string[] {"A","B","C","D"};
string str1 = "";

foreach (string s in sArray) {
    str1 += s;
}
print( str1 );                  // Prints: "ABCD"

List<string> sList = new List<string>( sArray );
string str2 = "";
foreach (string s in sList) {
    str2 += s;
}
print( str2 );                  // Prints: "ABCD"
```

Why can string s be declared twice?
Because string s is local to each foreach loop

Using `List` or `Array` ?

Each have pros and cons:

- List has flexible length, whereas array length is more difficult to change.
- Array is very slightly faster.
- Array allows multidimensional indices.
- Array allows empty elements in the middle of the collection.

Lists are simpler to implement and take less forethought (due to their flexible length)

- This is especially true when prototyping games, Lists might be best since prototyping requires a lot of flexibility

Other Collection Types

`ArrayList`

- `ArrayList` is like a `List` except without a set type
- Extremely flexible, but can be slower

`Dictionary<key,value>`

A key / value pair

- The key could be a string, like a username
- The value would be the user data

Used in some of the later tutorials in the book

Requires using `System.Collections.Generic;`

Very useful

But don't appear properly in the Unity Inspector

Summary

The collections we use in this book can hold any number of objects (of a single type)

`List` is the most easily usable collection type

- It requires using `System.Collections.Generic;`

`Arrays` are less flexible but also very useful

- Only arrays can be multidimensional

Both `Lists` and `Arrays` can be jagged

If your not sure – use `Lists` over `Arrays`