Code Lab 1

Data Structures

What is a data structure?

Data structures are the way we store and retrieve data

Data structures provide two things: clarity and performance

Data structures handle four things:

- Inputting information
- Processing information
- Maintaining information
- Retrieving information

Abstract Data Types (in C#)

- List
- 2D Array
- Dictionary
- Queue
- Stack

List

Lists are just arrays with extra juice.

- An array is of fixed size, where the size of a List is adjustable
- Easy to add and remove elements from a List
- Arrays are slightly faster, *sometimes*, lists are more flexible

Iteration is extremely cheap.

Add(item) is cheap, Insert(item) is expensive.

Useful List Properties

.**Count** - Number of objects currently in the list

.**Sort()** - Sorts the list as best it can.

.**AddRange(Collection)** - add multiple things to a list at once

.**Contains(object)** - returns true if a list has something in it, false if not

.**Insert(index, object)** - puts an object into the list at a specific place, shifting everything else down.

.**Remove(object)** - removes that specific object from a list

.**ToArray()** - turns the list into an array

When is a List useful?

- We want things to be *in an order*
- We want to do *something to everything in a collection*.
- We want things to be *sorted easily*
- We want *the first or last object quickly*

2D Array

- Useful in representing a 2 dimensional grid
- Easy to use a nested for loop to look at everything in the grid

```
int[,] grid = new int[8, 8];

for (var x = 0; x < grid.GetLength(0); x++) {
    for (var y = 0; y < grid.GetLength(1); y++) {
        if (grid[x,y] == 100)
            Debug.Log("Contains 100 at (" + x + ", " + y + ")");
    }
}
```

Dictionary

Removing or adding something is extremely cheap.
Going through every value is expensive

Dictionary: A ***unique*** key-value ***relationship***

- Define a type of key, and a type of value.
- You can use the key to get the value quickly.
- Each key *can only refer to one value.*

When is a Dictionary useful?

- When you need to establish a **unique, one-directionaly, key-value** relationship between two things.
- When you need a “phone book” of different entities, but their *order doesn’t matter*.

Useful Dictionary Properties

.**Count** - Number of items in the dictionary

.**Keys** - Gets a collection of all the keys in the dictionary

.**Values** - Gets a collection of all the values in the dictionary

.**Add(Key, Value)** - Adds a new key-value pair to the dictionary

.**Remove(Key)** - Removes the value and key pair from the dictionary

.**ContainsKey(Key)** - Returns true if the dictionary contains that key

.**ContainsValue(Value)** - Returns true if the dictionary contains that value

Queue

A Queue is first in - first out (FIFO)

```
Queue<Player> waitingPlayers = new Queue<Player>();

private void PlayerConnected(Player connectedPlayer) {
    waitingPlayers.Enqueue(connectedPlayer);        // same as a List.Add()
}

private void SpaceAvailable() {
    if (waitingPlayers.Count != 0) {
        Debug.Log("Adding player" + waitingPlayers.Peek()); // look at next
        Game.Add(waitingPlayers.Dequeue());
    }
}
```

Stack

A stack is first in, last out (FILO).

```
private Stack<Spell> waitingSpells = new Stack<Spell>();

public void SpellCast(Spell spell) {
    waitingSpells.Push(spell);      // like Add for a List
}

public void ResolveSpells() {
    while (waitingSpells.Count != 0) {
        Debug.Log("Resolving spell " + waitingSpells.Peek());
        Cast(waitingSpells.Pop());  // removes and returns the last thing
    }
}
```