Contents

[Collections of Things 1](#_Toc13836748)

[Arrays 1](#_Toc13836749)

[Lists 1](#_Toc13836750)

[Creating a List 2](#_Toc13836751)

[In the inspector 2](#_Toc13836752)

[Accessing the values 3](#_Toc13836753)

[Adding to the List 3](#_Toc13836754)

[Removing from the List 3](#_Toc13836755)

[Finding the length of a List 3](#_Toc13836756)

[Foreach Loops 4](#_Toc13836757)

[Creating a List Exercises 5](#_Toc13836758)

[Basic Exercise 5](#_Toc13836759)

[To try. 6](#_Toc13836760)

[Adding Character Exercise 7](#_Toc13836761)

[Setting up the scene 8](#_Toc13836762)

[Set / Reset our input fields 9](#_Toc13836763)

[Creating our character code 10](#_Toc13836764)

[Connect the Button 11](#_Toc13836765)

[Remove the Character 12](#_Toc13836766)

[Group Attack 13](#_Toc13836767)

# Collections of Things

In programming you often want to store a group of the same type. Storing it all together helps make it easier to access, and we can use special techniques to access the data easier. There are a many way we can do this, today we are looking at Lists.

# Arrays

Arrays while very useful. In the C coding field have the limitation of once set cannot be resized. This means you cannot take away or add to the arrays size once it is created. Arrays are great when it doesn’t need to be changed when the game is running, if we wanted to add and remove objects, it is a little annoying.

Think of an Array like a shelf with set shelved. We can change the values inside, but there is always the same amount.

## Lists

A list is like an array, with the bonus of being able to add and remove from it when the game is running. This allows us to make lists of objects that change throughout the game.

Think of a list like a set of folders, you can add and remove them from the pile easily.

This could be:

* Inventory items
* Enemies
* Allies
* Dynamic World information

## Creating a List

List<type> name = new List<type>();

Variable type

List is called first, as it is a class

List name

A List is different from an array as it is a Class.

* To create a list, we have to fist say that we are creating a new List variable
* We use the triangle braces <> to say what type of variable we want to be used as a list.
* We then set the name

## In the inspector

In the Inspector, we can edit the List just like we did with our Arrays.

To add to the list, we:

* Set the Size
* Set each element

## Accessing the values

Just like with we did with an array, we can grab the item from the list by putting in the index value.

This allows us to easily use loops to loop through all the values

## Adding to the List

Listname.Add(typevalue)

We can add to the end of the list by using the Add function. Call the add function and then pass in the types value you wish to add

## Removing from the List

Listname.Remove(typeValue)

you can Remove items from the list. When they are removed, it will NOT leave a gap in between

* remove by the value by using Remove

Listname.RemoveAt(indexValue)

* you can remove an item by its index value by using Remove At

## Finding the length of a List

To find the size of a list you use the Count variable.

Listname.Count

This is different from the Array which uses the Length variable.

# Foreach Loops

The For Each loop is a special loop that works well with Lists and other collections.

It is a cleaner looking for loop which we have used.

A foreach loop you need to:

* create a variable to store each element from the collection
* say what collection we are using

Foreach(varType name in collectionVarName)

{

name.doTask

}

We set up our Foreach loop like this

For each loop the item in the collection is stored in the variable created in the loop.

This means we don’t have to worry about indexes, just call the variable in the loop.

# Creating a List Exercises

## Basic Exercise

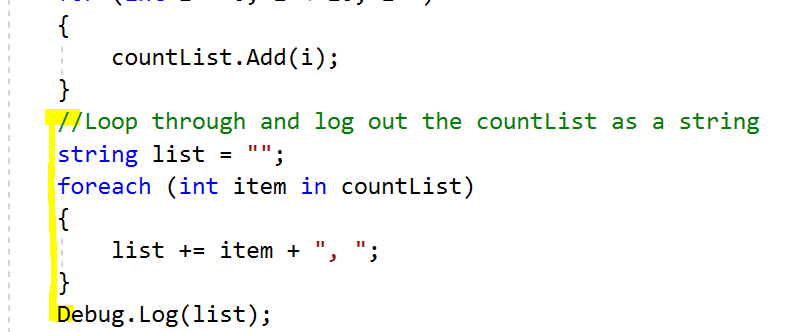
For a test, we are going to create a list that stores the numbers 1 to 9, We will then remove some items.

1. Create a new script
   1. Called BasicExercise
2. Attach it to an empty game object

* In the Start function

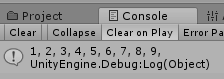
1. Create a List that stores integers
2. Create a for loop that goes from 1 to 10
3. Add the index value to the list

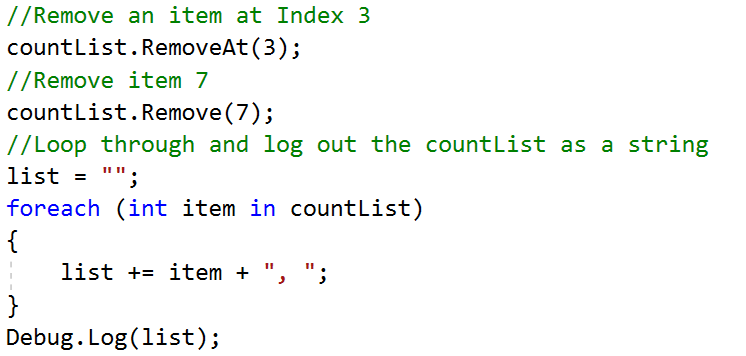
Now we have our list, we need to display it. We will store the values in a string and then log it

1. create a string variable called list
   1. Set it to “”
2. Create a foreach loop
   1. Have a Int item in countList
3. Add the item to the list pluss “, “

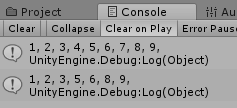
* When loop is done

1. Log it to the list

We should see the list displayed

Now we will remove a few items, one that uses the index and another that uses the object of similar value.

1. Call countList’s RemoveAt
   1. Pass in the index 3
2. Call countList’s Remove
   1. Say to remove 7
3. Create a new for each loop that loops through the countList as well
4. Log it to the scene.

Now we can see that the value at index 3 (the value 4) is removed and the value 7 is removed.

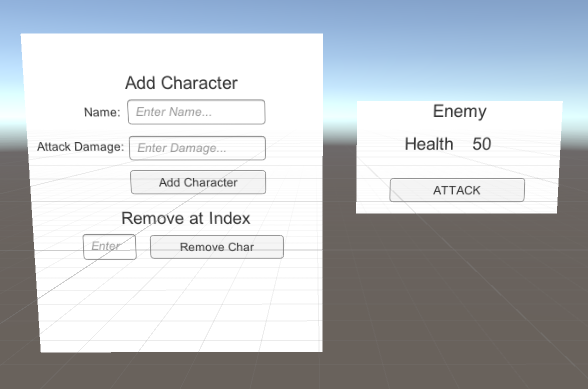
### To try.

Where you are removing from the list

1. Use a remove Range
   1. Pass in an index and a count of how many to remove
2. Use remove All to clear the list

## Adding Character Exercise

Let’s create an array that stores multiple enemies



We are going to create a List that ties in with a character creation tool.

Here are the tools we will create

#### Add character

* Get and check a name and attack has been properly added
* If it is, create a new game object
* Set the name and damage
* Add it to the list

#### Remove Character

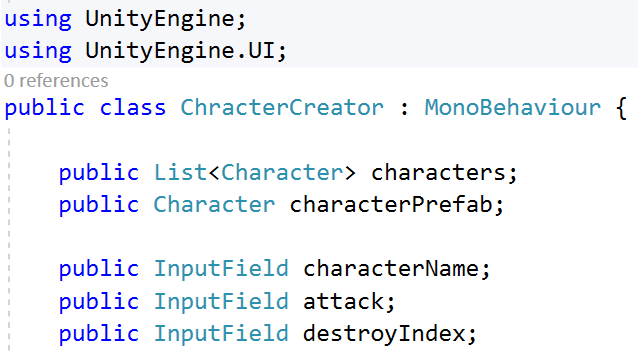
* Check to see if input is a value in the list
* If it is destroy game object and remove from list

#### Attack Enemy

* When the Attack button is pressed
* Loop through each character and damage the enemy

## Setting up the scene

1. Download and import the Add character Example Scene

This will include the UI, a character game Object and an enemy.

1. Create a new script called CharacterCreator
   1. Attach it to the Characters GameObject

* Inside the Script

1. Add using UnityEngine.UI

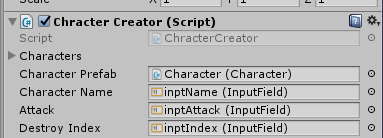
* In the Class Block

Fist we will create our List

1. Create a new public List
   1. Set the variables to Character
   2. Give it the name characters

Then we need to add a variable to store the characterPrefab our characters will use.

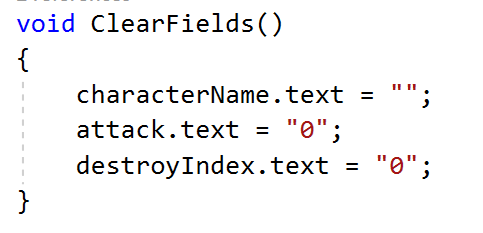
1. Create a public Character variable called characterPrefab

Lastly we will get the Input fields we need

1. Create an InputField for
   1. The characterName
   2. The attackValue
   3. The index on which character to Destroy
2. Plug in the Input fields and character prefab in the Inspector

## Set / Reset our input fields

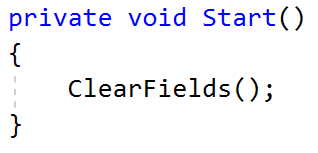
When we start or when we create a character, we want to reset all the fields.

We set it at the start to make sure that the fields that need a number have at least a 0 value

1. Create a void function called ClearFields
2. Set each texts variables to a default value.

* In the Start function

1. Call ClearFields

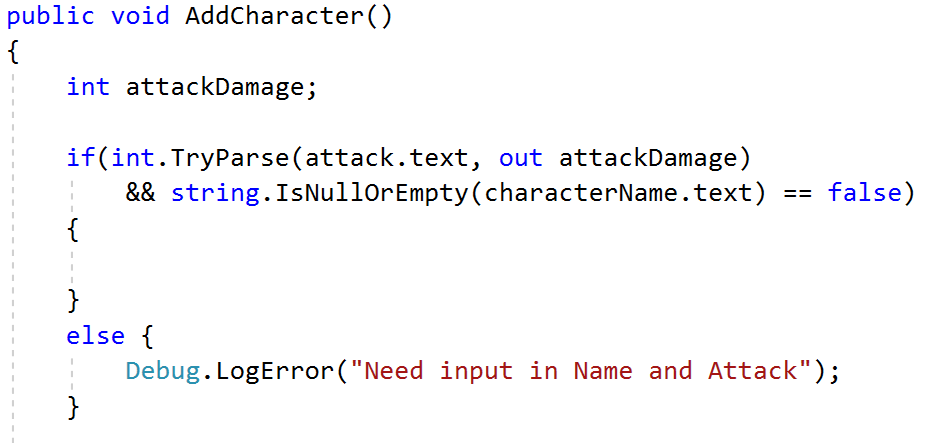


## Creating our character code

Let’s create our add character code

1. Create a public void function called AddCharacter()
2. Create a variable called int attackDamage

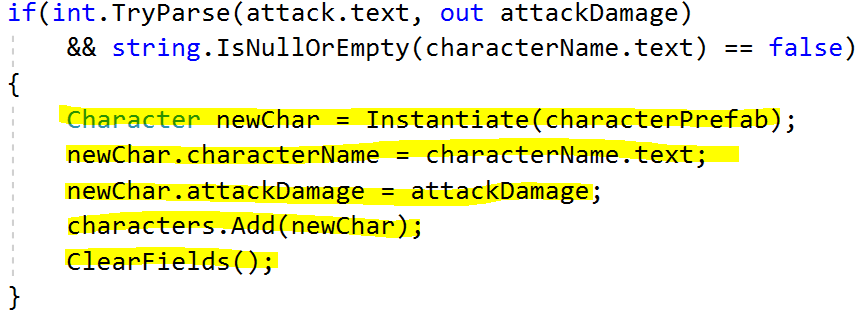
We are going to use this to check if the string our text field has can be turned into an integer. To do this we are going to use TryParse

TryParse will return true if it works, otherwise we can ignore it.

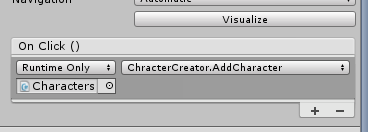
We are also going to do a check to see if our string is empty. To do this we pass our string into the field into the strings function IsNullOrEmpty

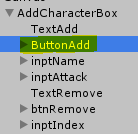
1. Create an if statement that checks
   1. Int.TryParse
      1. Of the attack.text
      2. If it works pass it into attackDamgae
   2. String.IsNullOrEmpty is false
      1. Pass in characterName.text

This will check if both have values.

Now that we have a check to see if we have filled out the list, we can now create our character and add it to the list

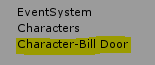
1. Create a new Character var called newChar
   1. Instantiate a new character Prefab.
2. Set the characterName and attackDamage of newChar
3. Call the characters array Add function
   1. Pass in newChar
4. Call ClearFields





## Connect the Button

Now we have our create function, we need to have the Add Character button call the function

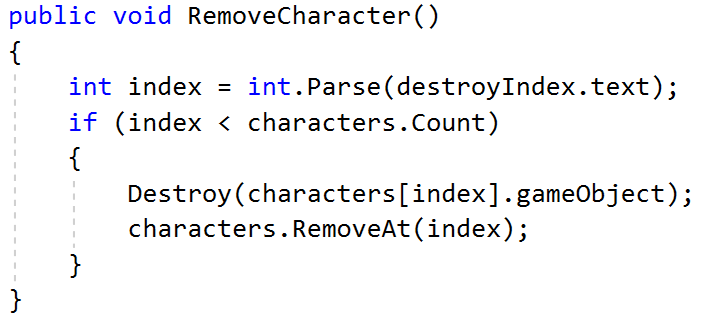
1. Select the button
2. Pass in the Game Object with the Character Creator script
3. Select the AddCharacter function

We should now be able to put in information and add a character to the game

## Remove the Character

We can add a character, but how about removing the character.

1. Create a public void function called RemoveCharacter()

We need to get the index of the character we want to remove, we should have this stored in the destroy index text variable

1. Create an int called index
   1. Pass in a Parsed destroyIndexs text
2. Check if index is in the characters List count

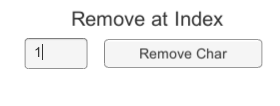
* If true

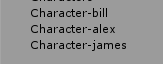
Destroy the object from the game world

1. Destroy the game object at characters index position

Remove the game object from the List

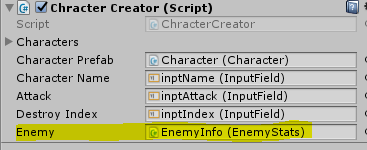
1. Call characters RemoveAt function
2. Pass in Index



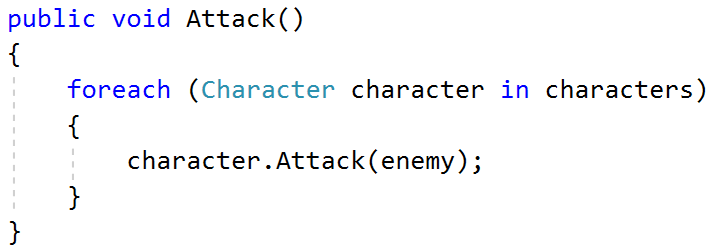


## Group Attack

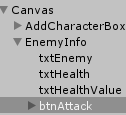
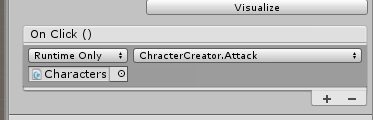
Now we have our groups, it would be good for them to take out an enemy together.

We can have a simulated character with the Attack button on the UI

1. Create a public EnemyStats variable called enemy
2. Pass in the EnemyInfo Game Object
3. Create a public void function called Attack()

Now we have our function, we want to loop through all the characters and hit the enemy with there attack damage

1. Create a foreach loop
   1. Character character in characters
2. Call character’s Attack function

We have the function. Let’s connect it to the button

* In the Inspector

1. Select the btnAttack button
2. Pull in the Game Objet Characters
3. Call the Attack Script

This should now damage the enemy with each characters attack.

If there are no characters it will do nothing