COMP2007 - Game Development

Week 7 - “Art” session

The Resources folder

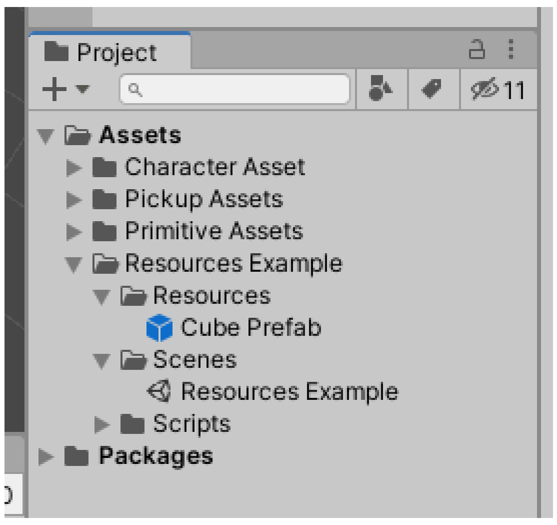
Certain folder names have special usage in a Unity project.
When you create a folder called **Resources**, you can load any items in the folder at runtime.
NOTE: the name must be spelled and punctuated exactly as shown above
Any file type can be loaded from Resources:

- Prefabs
- Scenes
- Models
- Textures
- Materials
- Sound files
- And more

NOTE: you can put C# files in there, but they are compiled code so will already be part of a build (leave them in the scripts folder!)

A project has no limits on how many Resources folders can be present - Unity will read from them all!
NOTE: normal naming conventions apply though - you cannot have two folders with the same name in the same folder etc

A Resources folder with a prefab



In code, you use Resources.Load and specify a path to the file
Optionally, you can use generics to specify the file type, like GameObject for the prefab shown below
NOTE: you don’t have to specify the file type (.asset, jpg etc) only the name

```
GameObject item = Resources.Load<GameObject>("Cube Prefab");
```

Tips

When a game starts up, Unity will load ALL of the assets in ALL of the Resources folder into memory.
This is great for early development, but as a game gets larger (like 100's of files in Resources folders), other strategies are available:

- StreamingAssets folder
    - A folder that will NOT be compiled - Unity will ignore any editable files
        - Code files in there will not be compiled!
    - Uses all the same file types as Resources folder can
    - Unity will not store the files in memory like Resources does
        - Larger files may need a little time to load into memory
        - Great for larger files or files that only need to be loaded once (text files for data etc)
- Addressable assets
    - Good for general asset organisation
    - Can package up all the same files as Resources into special packages
    - Packages can be installed while game is running
    - Often used as updates you may see on game services like steam etc
    - Made for integration with CDN

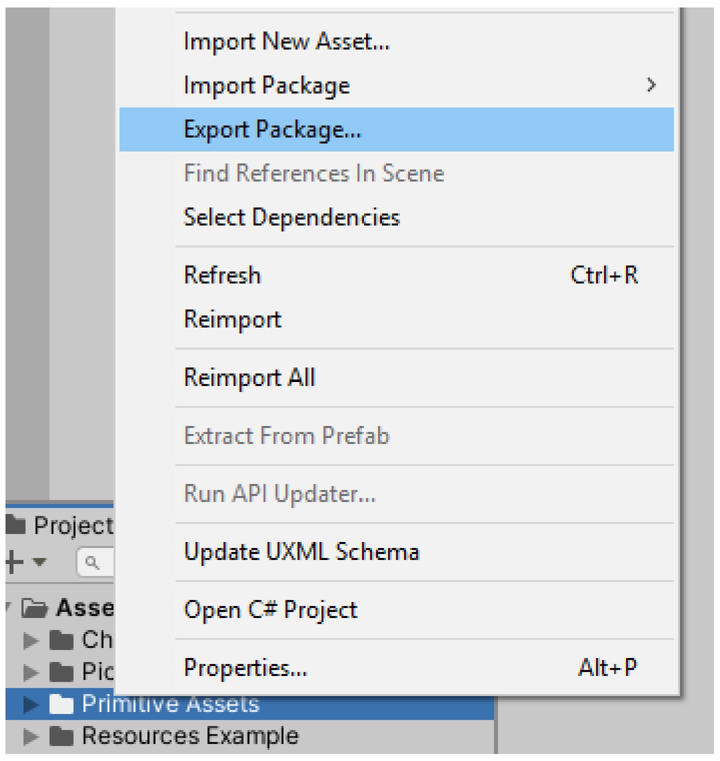
StreamingAssets folder
https://docs.unity3d.com/2020.2/Documentation/Manual/StreamingAssets.html

Addressable Assets
https://docs.unity3d.com/Packages/com.unity.addressables@1.17/manual/index.html

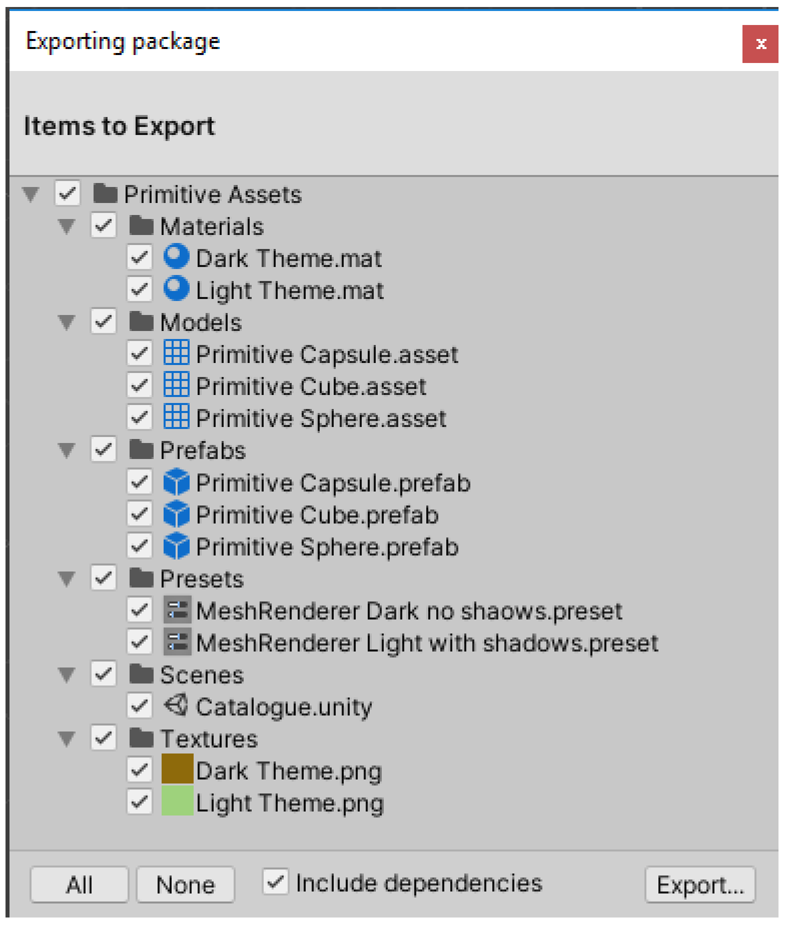
Unity Packages

For creating reusable assets across many projects, you can create a Unity-specific "zip" file of assets called a **Package.**
We can use Packages to create **Game Ready Assets**.
Unity packages are not used for loading content into the game at runtime - only for development in the editor
These can be created from the Project view by right clicking a folder

Right click a folder and select "Export Package"



The package keeps the folder structure just like a zip file would

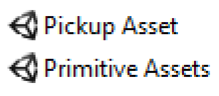

Exporting package

Items to Export

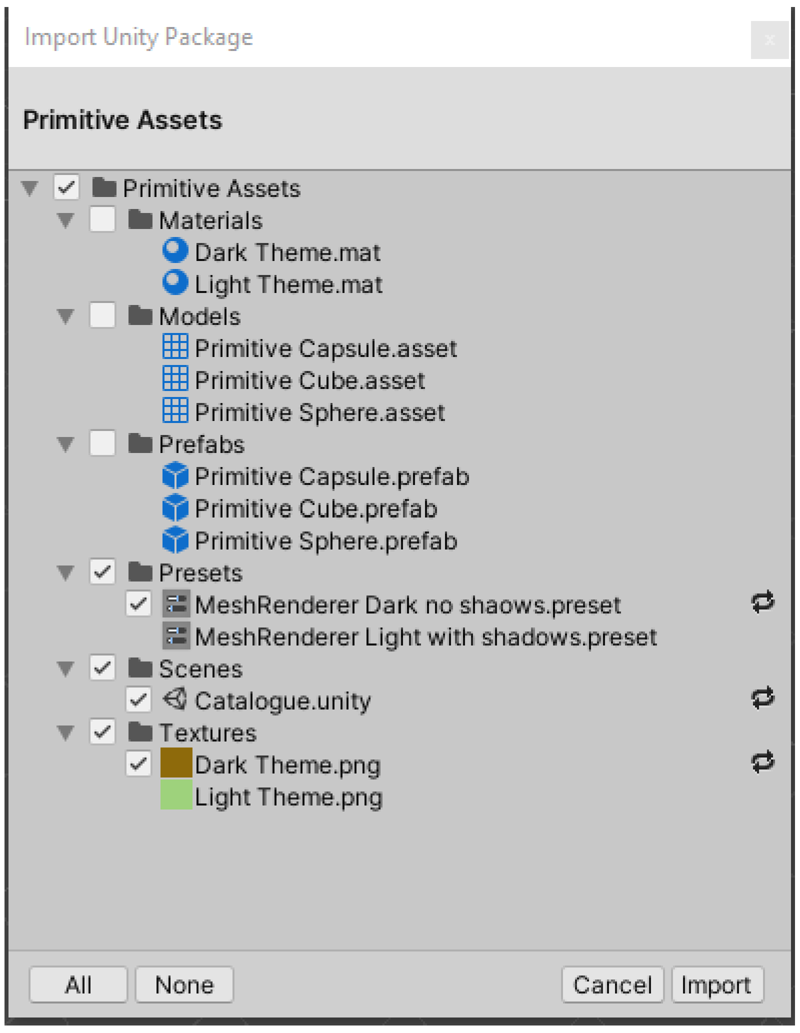
- ✓ Primitive Assets
  - ✓ Materials
    - ✓ Dark Theme.mat
    - ✓ Light Theme.mat
  - ✓ Models
    - ✓ Primitive Capsule.asset
    - ✓ Primitive Cube.asset
    - ✓ Primitive Sphere.asset
  - ✓ Prefabs
    - ✓ Primitive Capsule.prefab
    - ✓ Primitive Cube.prefab
    - ✓ Primitive Sphere.prefab
  - ✓ Presets
    - ✓ MeshRenderer Dark no shaows.preset
    - ✓ MeshRenderer Light with shadows.preset
  - ✓ Scenes
    - ✓ Catalogue.unity
  - ✓ Textures
    - ✓ Dark Theme.png
    - ✓ Light Theme.png

All None ✓ Include dependencies Export...

In an explorer window, the package files will have a unity logo, like scene files do in the project view



To open a package file inside a project, open the project first, then double click the package!


Import Unity Package

Primitive Assets

- ✓ Primitive Assets
  - Materials
    - Dark Theme.mat
    - Light Theme.mat
  - Models
    - Primitive Capsule.asset
    - Primitive Cube.asset
    - Primitive Sphere.asset
  - Prefabs
    - Primitive Capsule.prefab
    - Primitive Cube.prefab
    - Primitive Sphere.prefab
  - ✓ Presets
    - ✓ MeshRenderer Dark no shaows.preset
    - MeshRenderer Light with shadows.preset
  - ✓ Scenes
    - ✓ Catalogue.unity
  - ✓ Textures
    - ✓ Dark Theme.png
    - Light Theme.png

If a file is already present in the project, the package will not re import the file, use the tick boxes to specify files to import
If a file has been changed a small refresh icon appears to the right of the file, this will write over the file in the project with the one from the package

Tips for creating and organising your own Game Ready Assets

Game Ready Assets are items that can be immediately used in a project.
The Unity Package file is ideal for storing many small sets of assets, we can create a library of reusable assets for use across our many projects now and in the future.
Most game studios have libraries of tools and assets they can deploy into a project to save time.

The golden rule: you should be able to drag into the scene and use immediately

Provide documentation if there are lots of settings!

There are roughly three categories of use cases - these can overlap, a single item with a code system for example

- A single item
  - Like a 3D model with textures, shader, collision and scripts if required
  - Can be dragged into a scene and used immediately
  - Has a test scene that shows how it works, with a test script and setup if required
- Item pack
  - Several models built around a theme
  - Often share the same materials/textures
  - Collision/Physics already setup
  - Scripts included if required
  - Provide a “catalogue scene” of all the items laid out so you can pick things quickly in future
- Code system
  - A code library consisting of many classes interacting with each other
  - All code should be organised within its own namespaces
  - Make use of private serialised fields so you only have specific fields public
  - Use OOP to organise classes and reuse code (interfaces/abstract classes and design patterns)
  - Consider commenting the code where appropriate
  - Provide simple documentation (can be a text file or PDF if images are required)
  - Provide example scenes
    - Show the main use cases of the system
    - Provide test code to easily run the scenes

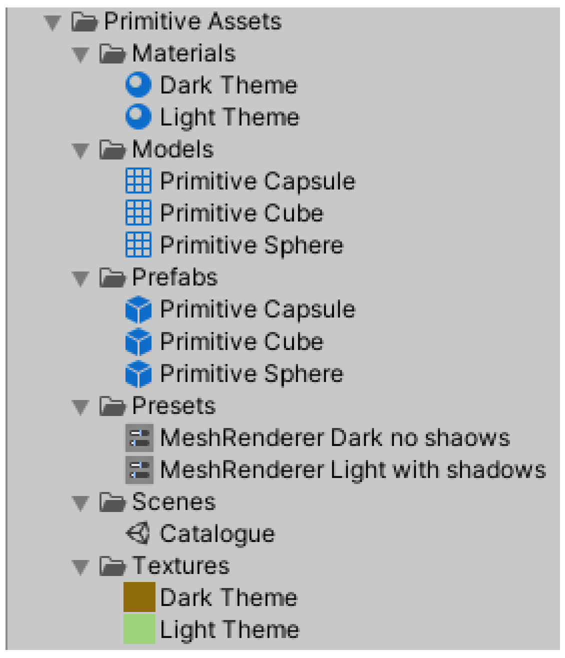
Tips

Folder organisation

Be consistent in your folder organisation:

- Organise the files into their own folders OR
- Organise all the files for an asset into one folder (text, model, material, shader etc)

My personal preference is to organise by file type as below



```
▼ Primitive Assets
  ▼ Materials
      Dark Theme
      Light Theme
  ▼ Models
      Primitive Capsule
      Primitive Cube
      Primitive Sphere
  ▼ Prefabs
      Primitive Capsule
      Primitive Cube
      Primitive Sphere
  ▼ Presets
      MeshRenderer Dark no shaows
      MeshRenderer Light with shadows
  ▼ Scenes
      Catalogue
  ▼ Textures
      Dark Theme
      Light Theme
```

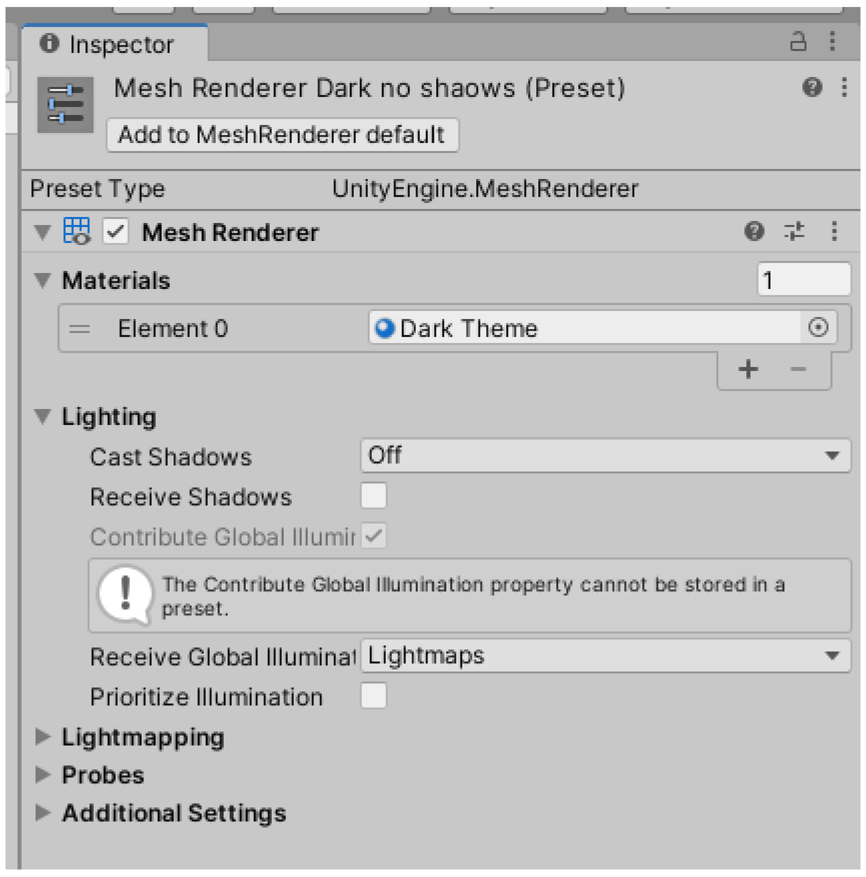
Presets

Provide presets for easy component setups
A **Preset** is a file that stores the settings for a component

Preset file for a mesh renderer component

```
▼ Presets
    MeshRenderer Dark no shaows
```

The inspector view for the mesh renderer preset



Inspector

Mesh Renderer Dark no shaows (Preset)

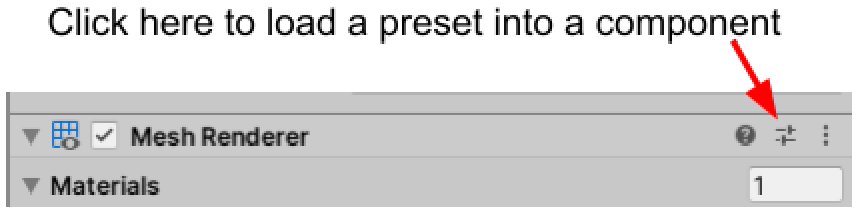
Add to MeshRenderer default

Preset Type	UnityEngine.MeshRenderer
Mesh Renderer	
Materials	1
Element 0	Dark Theme
Lighting	
Cast Shadows	Off
Receive Shadows	☐
Contribute Global Illumir	✓
Receive Global Illuminat	Lightmaps
Prioritize Illumination	☐

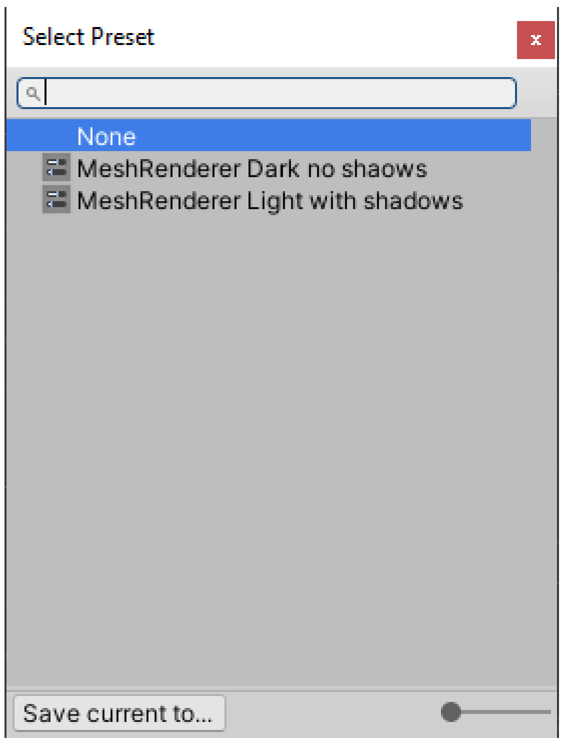
The Contribute Global Illumination property cannot be stored in a preset.

▶ **Lightmapping**
▶ **Probes**
▶ **Additional Settings**

Click here to load a preset into a component



You can load other presets or save the current component setup

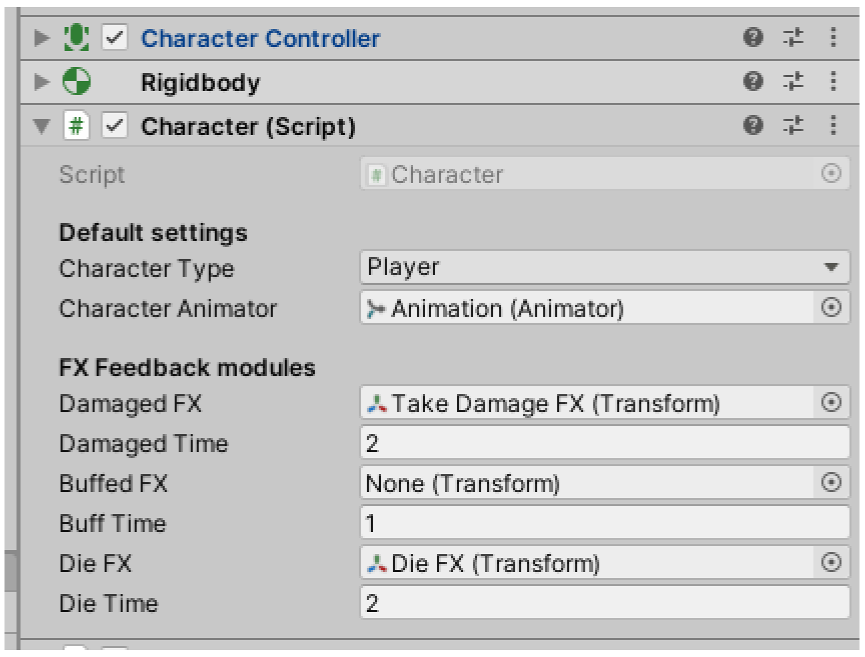

Select Preset
None
MeshRenderer Dark no shaows
MeshRenderer Light with shadows
Save current to...

Code system guidelines example: Character

For player controlled or AI characters it's often good to have a main "Character" class that everything hooks into
The Character can handle general settings:

- Animations
    - Part of the class, handled in the update
- Health
    - A separate component that provides events for:
        - Taking damage
        - Adding health (buff)
        - Death
- FX
    - Part of the class, a simple on/off switch to GameObjects containing FX
- Input
    - A separate component, handles user input

The Character component requires CharacterController and Rigidbody components



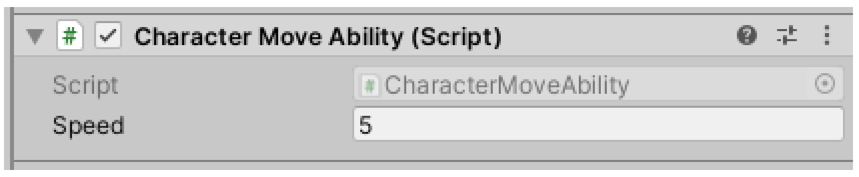
Character Controller
Rigidbody
Character (Script)

Script	Character
Default settings	
Character Type	Player
Character Animator	Animation (Animator)
FX Feedback modules	
Damaged FX	Take Damage FX (Transform)
Damaged Time	2
Buffed FX	None (Transform)
Buff Time	1
Die FX	Die FX (Transform)
Die Time	2

Character Abilities

An abstract class is provided for creating abilities the character can do.
Has a Character field to access the Character component
An example ability for movement is provided.

The move ability has a custom field for move speed.
This component will use the Character components' input manager to move on user input
NOTE: this should only be used with the player, another solution would need to be created for NPC characters



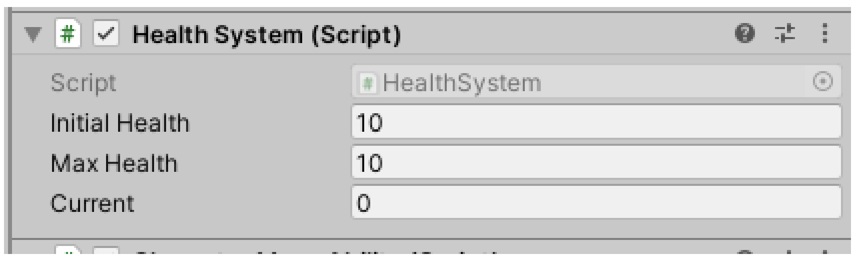
Character Move Ability (Script)

Script	CharacterMoveAbility
Speed	5

Health System

The health system controls the health of the character.
This component focuses solely on the change in health and sends events when the health changes
The Character component listens to the changes and triggers appropriate responses including any FX.

Set the starting or initial health and the max health value
"Current" is a debug view for the current health while the game is running



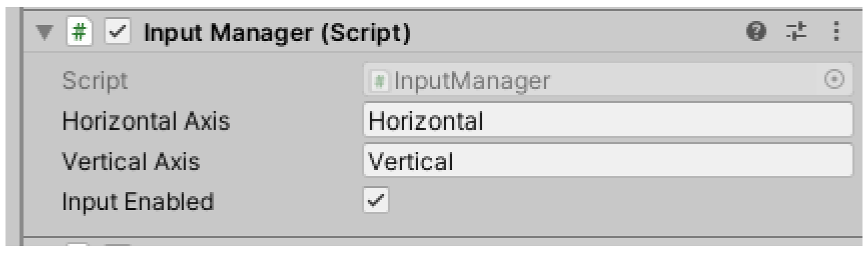
Health System (Script)

Script	HealthSystem
Initial Health	10
Max Health	10
Current	0

Input Manager

Listens for input from the user.
The example is set up for horizontal and vertical axes, but any control should be routed through this component.
The Character component will look for an input manager component if a player character is selected

Set the name for the axes to be used from the Input system
“Input Enabled” is a public field for other classes (like the Character) to easily disable input if required (like pausing the game, opening a menu etc)



Input Manager (Script)

Script	InputManager
Horizontal Axis	Horizontal
Vertical Axis	Vertical
Input Enabled	✓

Links

Resources.Load
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/Resources.Load.html

CharacterController.Move
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/CharacterController.Move.html

Vector3.Normalize
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/Vector3.Normalize.html

RequireComponent
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/RequireComponent.html

UnityEvent
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/Events.UnityEvent.html

UnityEvent.AddListener
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/Events.UnityEvent.AddListener.html

UnityEvent.RemoveListener
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/Events.UnityEvent.RemoveListener.html

SerializeField
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/SerializeField.html

HideInInspector
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/HideInInspector.html

Input.GetAxisRaw
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/Input.GetAxisRaw.html

C# Abstract class
https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/abstract

C# enum
https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/enum

C# typeof
https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/type-testing-and-cast#typeof-operator

C# nameof
https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/nameof

GAME DEV. TECH.

UNIVERSITY OF PLYMOUTH