Adventure Game Toolkit

2322033

Contents

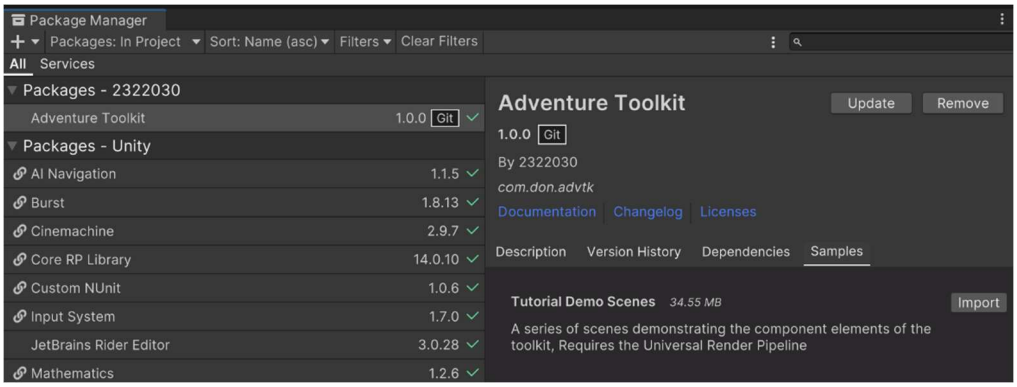
Introduction

Welcome to the adventure toolkit, a series of scripts and prefab components that you can use to build your own adventure game in unity.

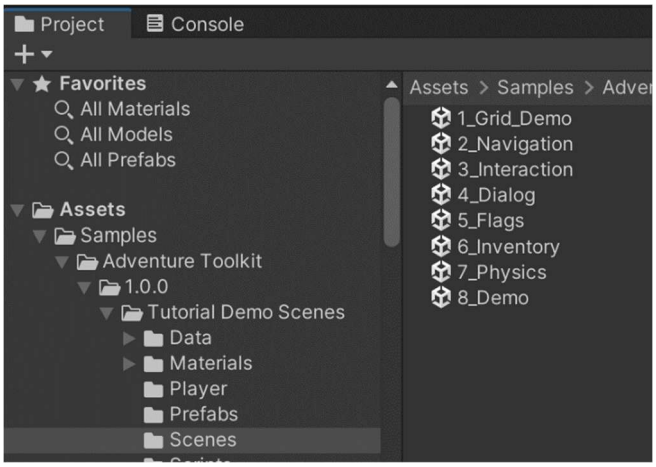
The components break down as follows and will each be covered in their own section.

1. Grid component – for dynamically building levels, maps or any tile-based object.
2. Player Movement-a point and click interface for moving the player in the scene.
3. Interaction – is the heart of an adventure game.
4. Dialog – speaking to non-player characters.
5. Flags – monitoring state in your levels.
6. Inventory – being able to grab, give and take objects.
7. Physics – an example of using a physics-based problem in your game.

You can access Tutorial Demo Scenes by importing them from the Samples tab in the Package Manager panel.



Once installed, they will be available in the following location in the project panel:



This is an optional step as throughout this documentation we will show you how to incorporate these components step by step into a new scene.

Further help can be found in the component scripts themselves, as they are fully commented, well named and easy to read.

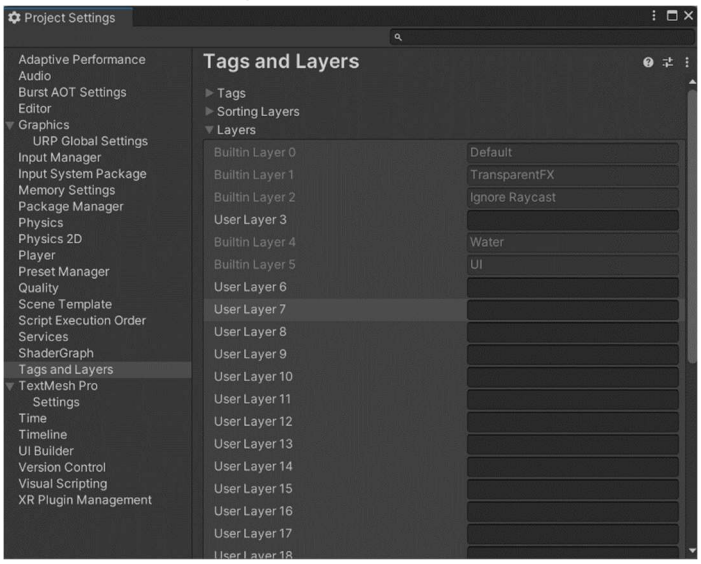
So, let's get started.

Prerequisites

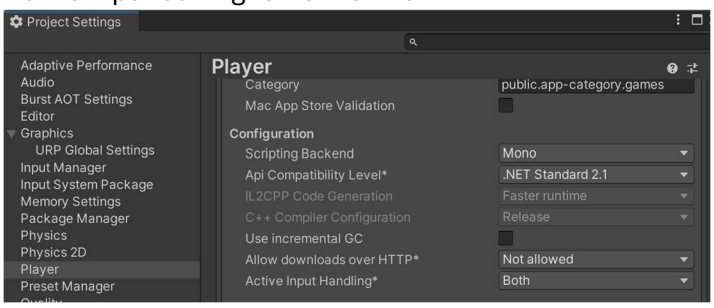
The package will have installed the Universal Render Pipeline as it is used in the Tutorial Sample however, the toolkit is renderer agnostic so you can use Standard, High-Definition, or your own custom render pipeline.

The toolkit does however require two small changes to your project settings.

1. A Ground layer on layer 7, called ground, floor, walkable or whatever you choose, the layer number (7) is the important part, and that any area you wish to have the player move on be set to that layer. The layer can be adjusted later if it conflicts with another, but the default is layer 7.

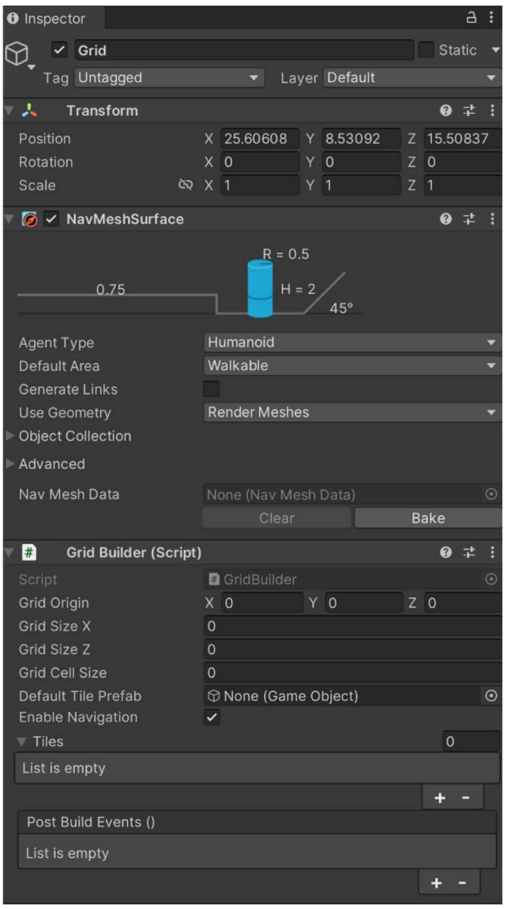


2. Active Input setting to Both or New



With that done we can move on to our first component.

Grid Builder Component



The component can be simply added to an empty gameObject by clicking “Add Component” and searching for Grid Builder.
The component will automatically add a NavMeshSurface component.

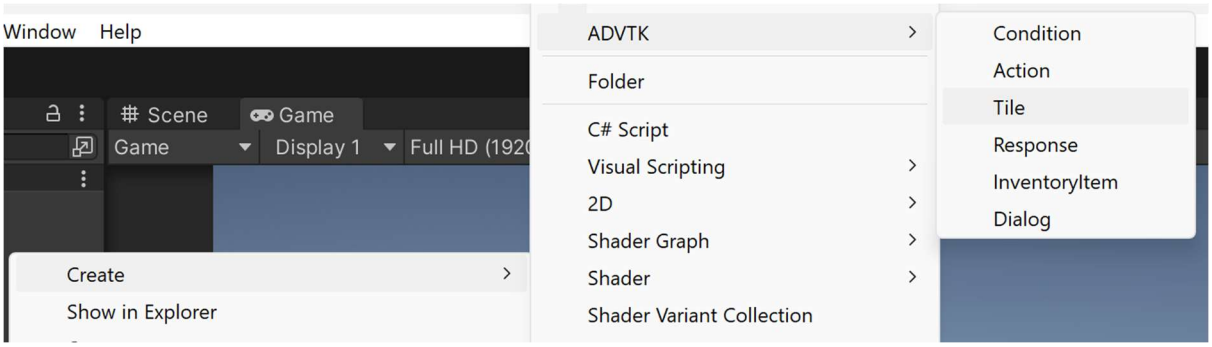
As the name suggests this builds a two-dimensional grid of prefabs from an array of scriptable objects.

The component allows you to specify the length and width of your grid and the unit of size for an individual "tile".

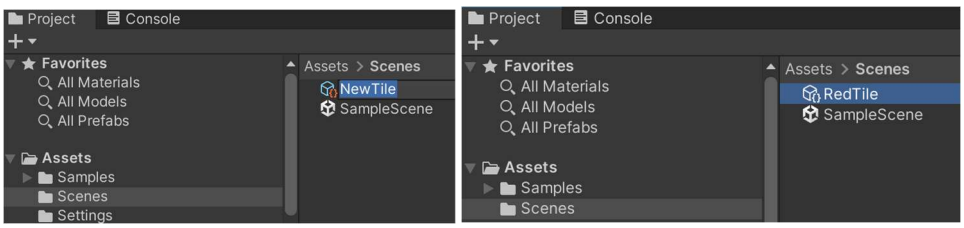
The component looks for an array of Tile Scriptable Objects (See below).

You do not need to define every tile in the grid, a default tile can be provided.

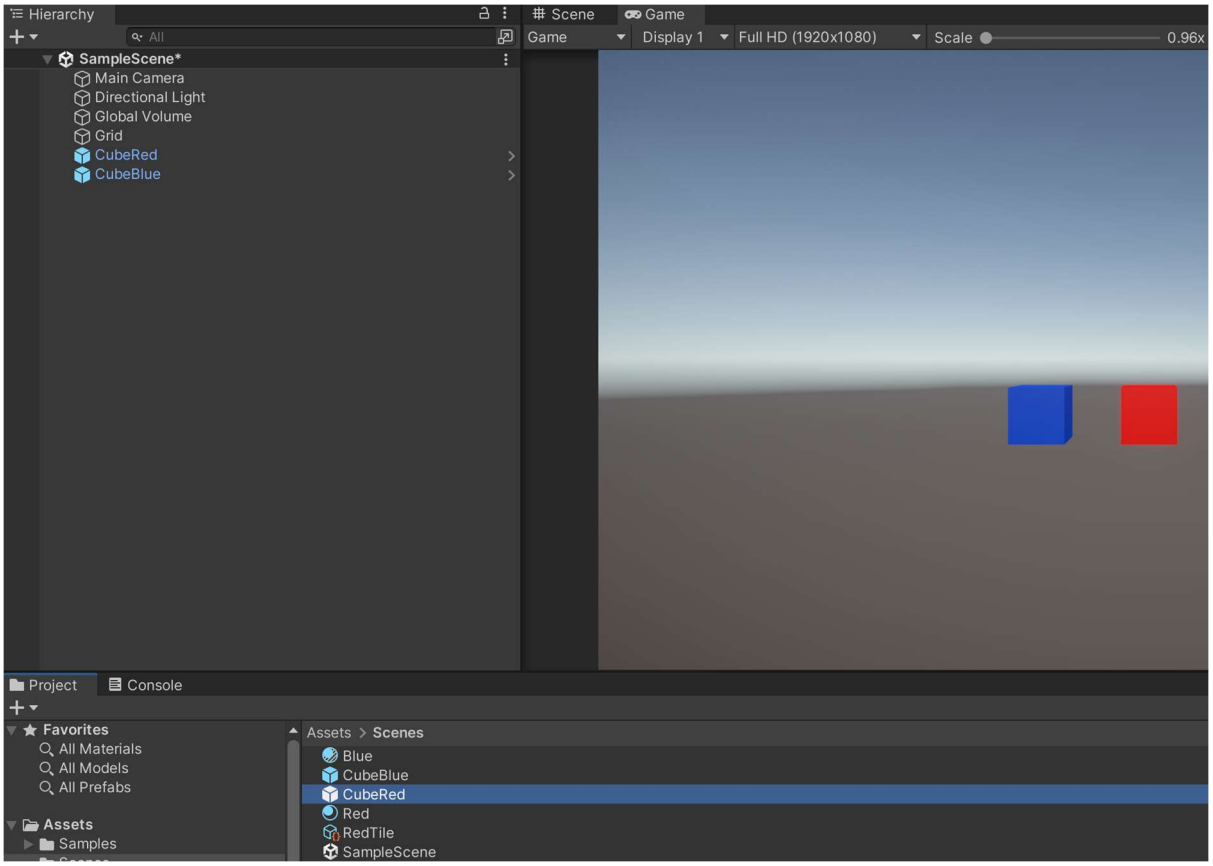
Grids can be used for many purposes, if you are using it for level building, please tick the Enable Navigation field and ensure that any meshes in your prefabs are set to read/write.



To create a tile scriptable object right click in your folder of choice and select Create > ADVTK >Tile and give your new tile a name. For demonstration purpose I’ll call it RedTile and come back to it in a moment.



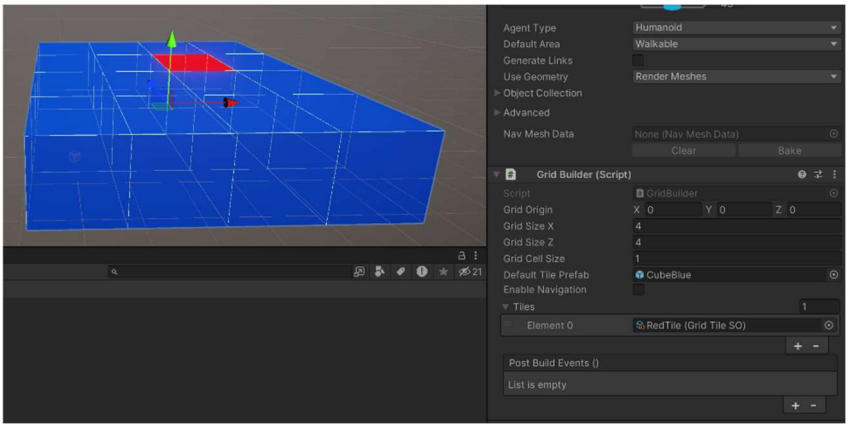
In this folder I'll also create a couple of simple materials one red one blue. The grid system uses Prefabs, so in the hierarchy I'll also create two 3D Cubes called CubeRed and CubeBlue and apply the materials to them. We can then drag them from the Hierarchy to the Project folder to create two new prefabs, then remove them from the scene.



Coming back to our RedTile in the Inspector window we can drag the CubeRed prefab in and set our X & Z.
The Y offset is only used to raise or lower a tile out of the grid.

Going back and filling out our Grid Builder with the default (blue) prefab and some dimensions will create your grid when you run the scene!

Prefabs can be anything, not just cubes 😊, walls doors, floors, map pieces etc.



But they must be a uniform size to fit in the grid.

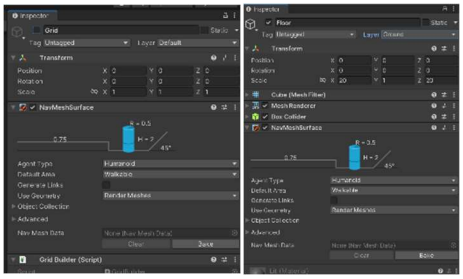
Player Movement

For ease of use this component comes as a prebuilt prefab called **PointAndClickPlayer** found in the Prefabs folder. The asset uses Unity's Robot Kyle (used under Companion Licence) as a base, but can be easily swapped out for your own character, this YouTube video has a five minute tutorial on how.

Internally the movement script uses Unity's NavMesh for pathfinding and all of the characters movement characteristics are customisable in the NavMeshAgent component.

As we mentioned on the previous page, we can generate a NavMeshSurface on our grid component by checking the "Enable Navigation" checkbox, but for this example we will show you how to implement it on custom geometry.

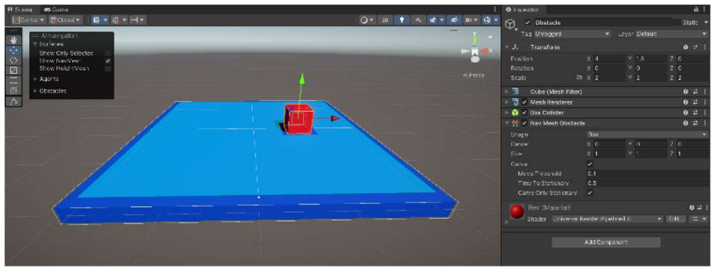
So, in our sample scene I will disable the grid GameObject .



Then add in a 3D Cube rename it Floor, scale it to 20,1,20 and add a NavMeshSurface Component, and most importantly change the layer to layer 7: Ground.

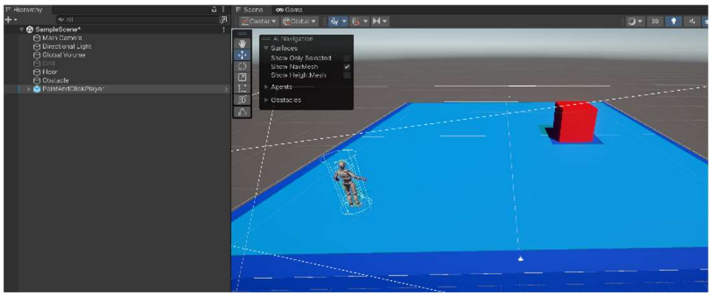
Add a second cube named obstacle, scale it to 2,2,2 add a NavMeshObstacle component, tick the carve option and move it to somewhere on our floor.

To distinguish a little better drag the blue material we created onto our floor and the red one to our obstacle. Then on our NavMeshSurface, click bake and view it in the scene view.



With our preperation finished we can now add in our player controller by dragging the **PointAndClickPlayer** from the prefab folder into the scene and move them to somewhere on the floor.

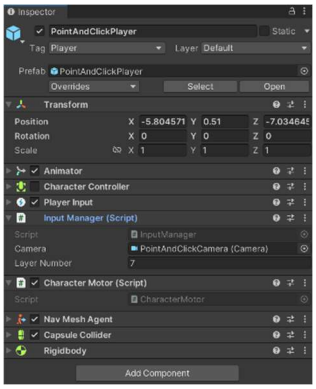
Lastly disable the scene's main camera, as our prefab has one of it's own.



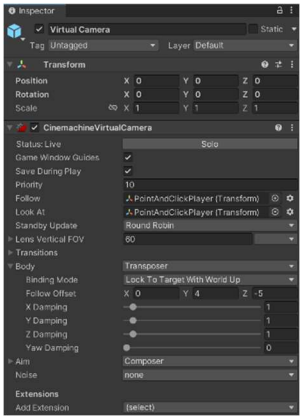
Press play, and you will be able to click on the floor to move the character around.

The prefab contains the normal geometry and skeleton that you would see in any skinned mesh model, as well as a camera, Cinemachine virtual camera and a empty game object called player camera root that is used as the point that the virtual camera looks at and follows.

The two toolkit components on the player are the Input Manager and the Character Motor. As mentioned above if you have to change the default ground layer it can be changed in the Input Manager>Layer Number.



Should you wish to adjust the camera position or angle relative to the player it can be modified in the Body > Follow Offset properties of the CinemachineVirtual Camera. This should allow for a very wide range of views for your adventure game.

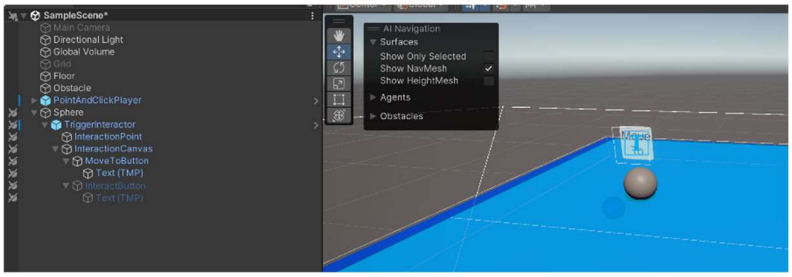


Interaction

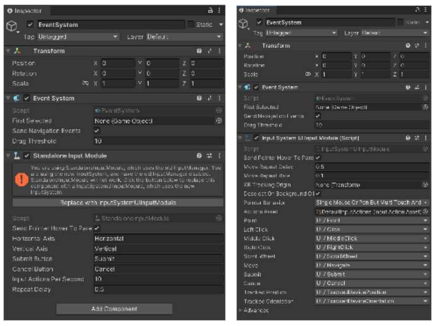
The toolkit follows the principal that a player should be within range to interact with an object.

So, to this end our interaction component will display one of two interaction buttons, a “Move To” button and a “Interact button”. These can easily be customised to your needs as you would with any other UI button.

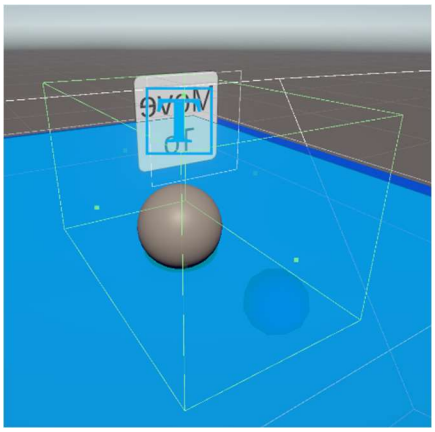
So, in our scene let’s add a sphere on the ground and call it Interactable, then from the prefabs folder drag the TriggerInteractor in as a child of the sphere.



As the component uses a UI canvas we will also need to add an EventSystem into the scene. UI>Event System and then click the “Replace with InputSystemUIInputModule” button.

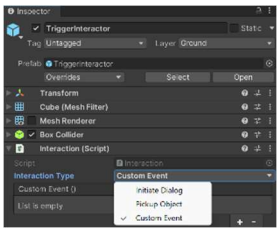


As the component can make objects of any size or shape interactable you will need to edit the box collider, so your object fits inside it (allowing room for the player) and that the interaction point child object is also within that area. The Interaction Canvas is a world-space canvas that will rotate towards the player so that it is always visible, after setting up the trigger box move the dialog to a place above or beside your object that makes sense for your game.



When the player is outside the trigger box the “Move To” button will show, and when inside the “Interact button will show, so some experimentation with the sizes and position of the component elements is expected to fit your game.

The Interactor has three types:

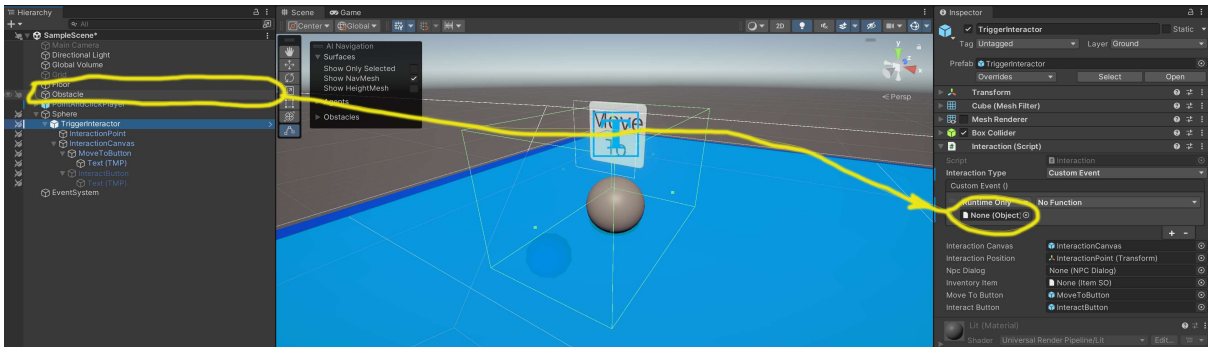


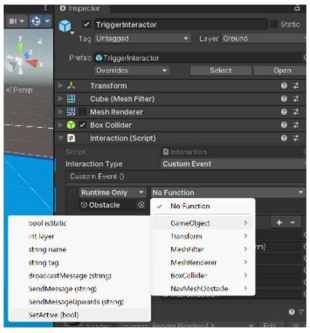
- Initiate Dialog
- Pickup Object
- And Custom Event

We will look at the first two later in the documentation

An select custom event for now, and click the + button to add a slot for a new UnityEvent

In this case we will do some magic and make our obstacle disappear 😊.





Drag our Obstacle object from the Hierarchy to the object slot on our custom event, then in the function dropdown select GameObject>SetActive (bool) and leave the Boolean value unticked.

This should deactivate the Obstacle in our scene when the Interact button is pressed. This is incredibly powerful as it will also allow you to run *any* public method (with a single parameter) on any script, on any object.

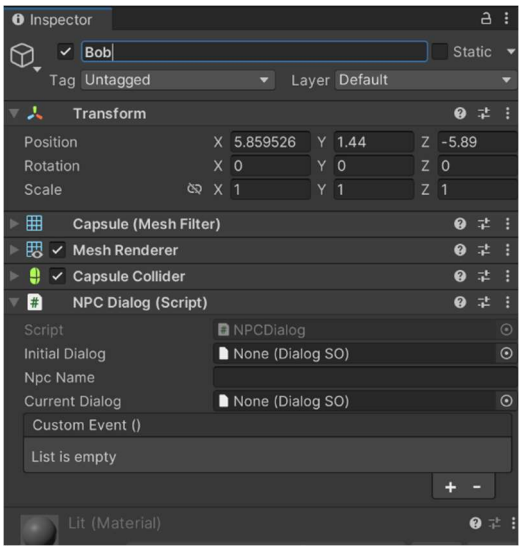
Run your scene, click on the move to button to move the player to the interaction point, then the interact button will hide your cube.

Dialog and Object pickup will be covered in their respective sections below.

Dialog

Dialog is a little move complex at first but easy one you have done it a few times.

Let's create a capsule called Bob in the scene to be our NPC and place it on our floor, then add the NPCDialog component. You've probably guessed already but I'm going to set the Npc Field to "Bob". The Initial dialog is the one the NPC starts with and the current dialog is for debug purposes as it can change during the game.

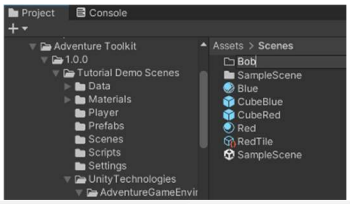


You will notice that our dialog fields expect a DialogSO datatype, this is a scriptable object (similar to the tile created in the Grid documentation) that holds the data on our dialog.

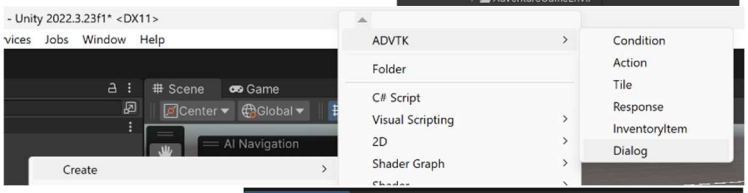
To ensure that our dialog is as realistic, and flexible as possible, there are two main types of dialog, non-conditional and conditional.

Non- Conditional will simply show the NPC's text and the players responses on the dialog canvas.

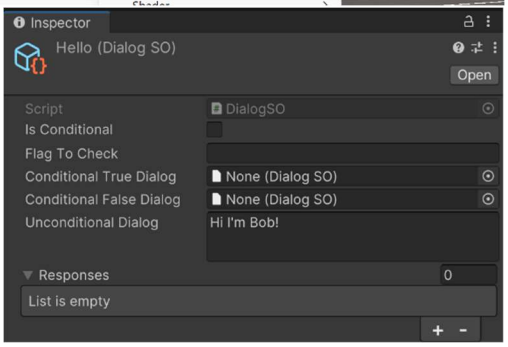
Starting simply, create a new folder called Bob in our scene folder in the Project panel.



Then in that folder right click and create a new Dialog scriptable object called Hello

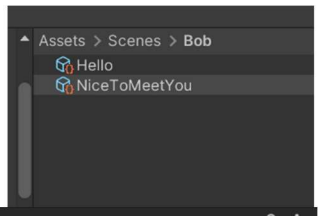


Inspect the scriptable object and give Bob some Unconditional Dialog. As you can see the conditional dialog uses our Flag system, so we'll cover more in that section.



You can also see that the SO expects some responses from the player, our dialog canvas supports up to four responses, but the system could be extended to handle more, comments in the scripts show where this could be done.

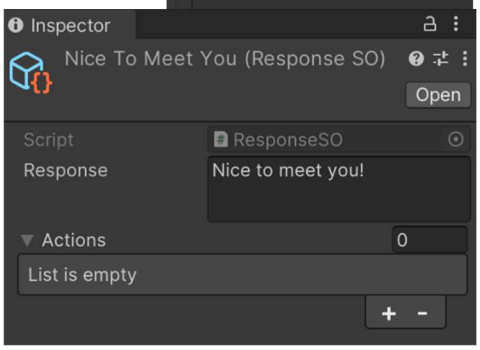
So, in the same folder I am going to create a Response (from the same menu) and call it “nicetomeetyou”.



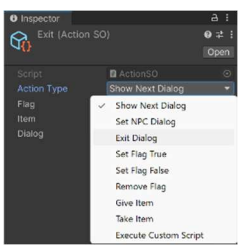
Inspecting the item, I’ll add that text, and we can see that the Responses SO is expecting one or more Actions.

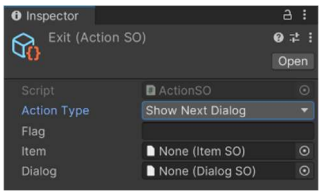
Actions as you would expect are what happens when the player clicks that response and are another scriptable object that we can create from the same ADVTK menu.

So, I’ll add one in the folder and call it exit.



Inspecting it you can see that you have a wide range of options but for this example I will simply choose “exit dialog” The other fields on the script are for other types of action, so for example if I wished to set a flag called “metBob” to true I would need to fill the Flag field.
If I wished to go to bobs next dialog.. provide a Dialog etc.

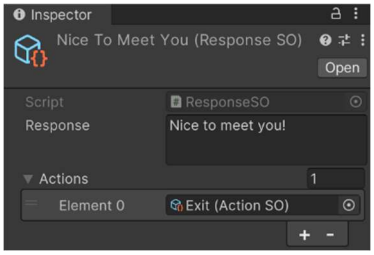


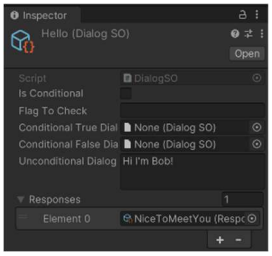


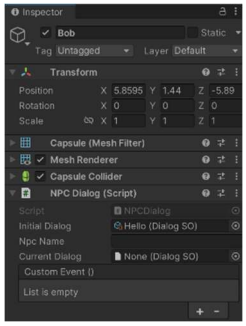
The Execute Script will run any Event on the NPCDialog custom events in a similar way to the interactor of the TriggerInteractor described previously.

Other options are covered in Flags and inventory below and shown in the package samples available through the Package Manager>Samples.

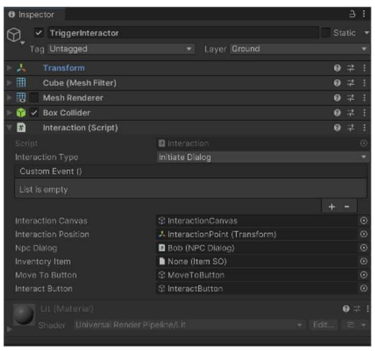
We now need to hook these things together, so the Exit action is added to the NiceToMeetYou Response the NiceToMeetYou is added to the Hello Dialog and the Hello Dialog is added to Bobs NPCDialog script







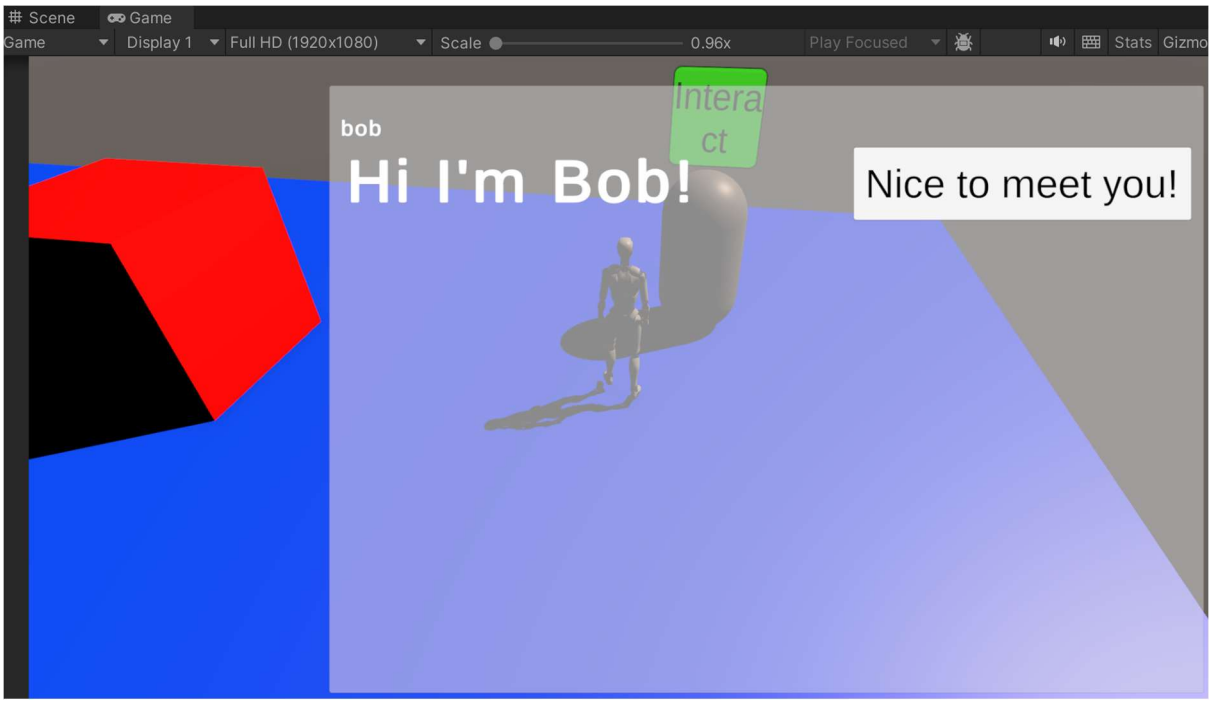
Now in order to get Bob to talk to us we need to interact with him, so drag in the TriggerInteractor prefabs exactly as we did in the Interaction documentation above, but changing the “Interaction Type” to “Initiate Dialog” and drag Bob’s NPCDialog script onto the Npc Dialog field.



Two last things to do are add the DialogDisplay prefab and the DialogManager prefab and allocate the DialogDisplay on the manager script.

It should be noted that the display is very simple, and it is expected that you theme the canvas to your game.

Now just walk up to the capsule – interact and you should see the following.



Clicking Nice to meet you will exit the dialog, and re-interacting will show the same thing.

Not entirely natural for a character to continuously re-introduce themselves but we can fix that with flags.

Note that dialog does not have to be with humanoid “characters” the system could also be used for mechanisms, locked doors, inspecting items or similar tasks just by changing the phrasing.

E.g.

Old oak door

Examining the old oak door you see a hidden switch…

[flip the switch]

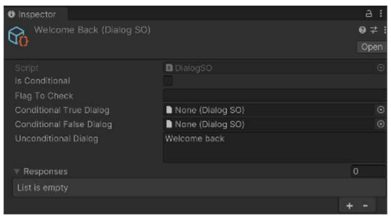
[back away]

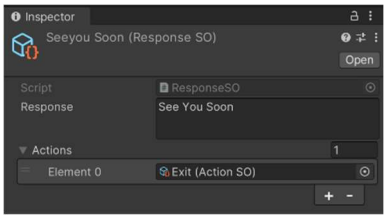
Flags

A flag is a simple way of maintaining state, for example we want to know when bob has met the player for the first time and respond with different dialog.

Create an empty object in the scene and add the FlagManager component.

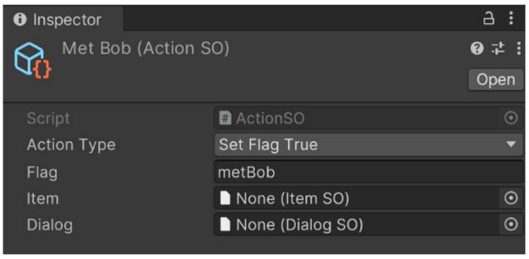
So, let's start by creating a new dialog called welcome back, and a new response called see you soon.





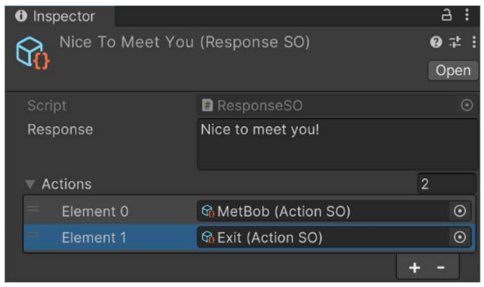
We'll also assign the Exit action to the response.

Next, we'll create a new action called MetBob and we will change the Action type to "Set Flag True" and name the flag "metBob" **importantly this is case sensitive** so please be careful with flag names and agree a naming standard with your team.



Now we want this action to happen after our first chat with Bob

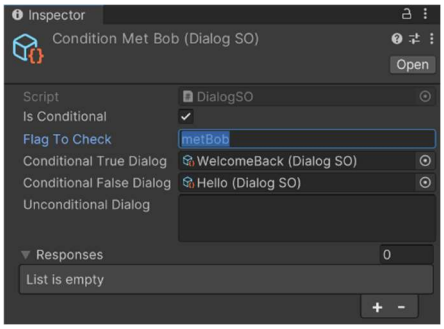
So, we go back to our "NiceToMeetYou" response and add in our new action. Note Exit Dialog actions should always be last in the list.



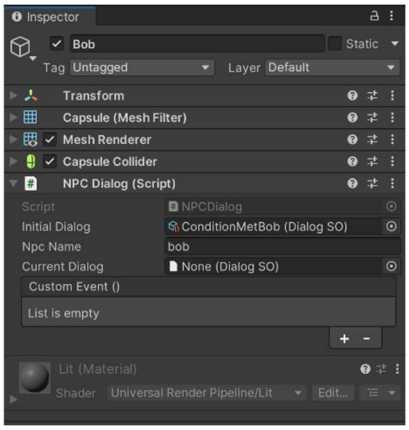
Lastly, we need to create a new dialog and call it "ConditionMetBob", the names don't really matter but good naming will help as your dialogs get larger and more complex

We tick Is Conditional, enter the flag and drag on the two dialogs, nothing else is needed.

So, if the flag metBob is true we jump to the WelcomeBack dialog or if false to the Hello dialog.



Lastly, we need to adjust Bobs Initial dialog to the conditional one.



Running this will then switch between the two dialogs on the second interaction.

Flags are incredibly useful throughout any adventure game, and they can be set, checked, or removed from any script in your program from the public methods available. There are also public save and load methods for persistance between play sessions.

For more information see comments in the FlagManager.cs file.

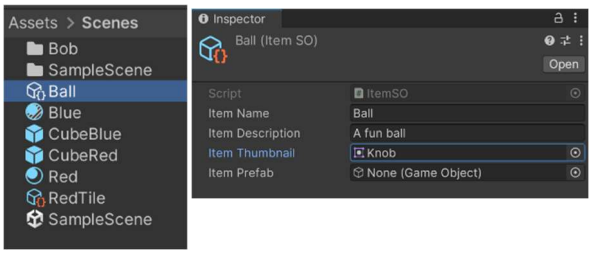
Inventory

Collectable items are another staple of adventure games commonly items can be collected from the environment, received from NPC or bought from shops. Unfortunately shops are beyond the scope of this toolkit, but could be integrated easily from the public `AddItemToInventory(ItemSO) and RemoveItemFromInventory(ItemSO)` in the InventoryManager.cs script.

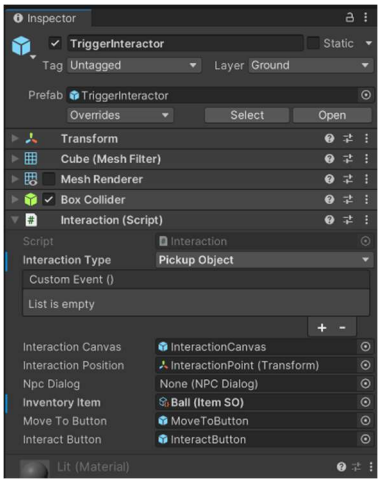
Firstly, add a new empty object to the scene and add the InventoryManager component. Then drag the InventoryCanvas into the scene. As with the dialog the presentation is kept very simple so you can customise to the theme of your game.
As with our other systems a scriptable object is used for storing Item Data the InventoryItem. We also need to drag the PopUpCanvas prefab into the scene as it notifies the player of inventory changes.

In our scene folder use the same ADVTK menu to create an InventoryItem called Ball.

Then fill out the name, description and sprite. The prefab field is optional an is include for those that wish to impliment a drop mechanic.

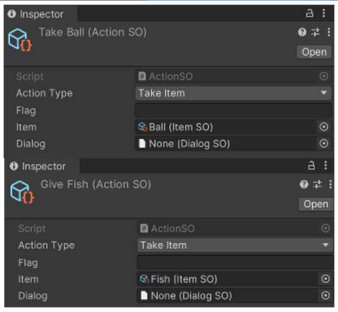


In our scene go to the TriggerInteractor in our Sphere and change the interaction type from “Custom Event” to “Pickup Object”. Remove the custom event by clicking the minus button, then drag the Ball InventoryItem from the project folder onto the Inventory Item.

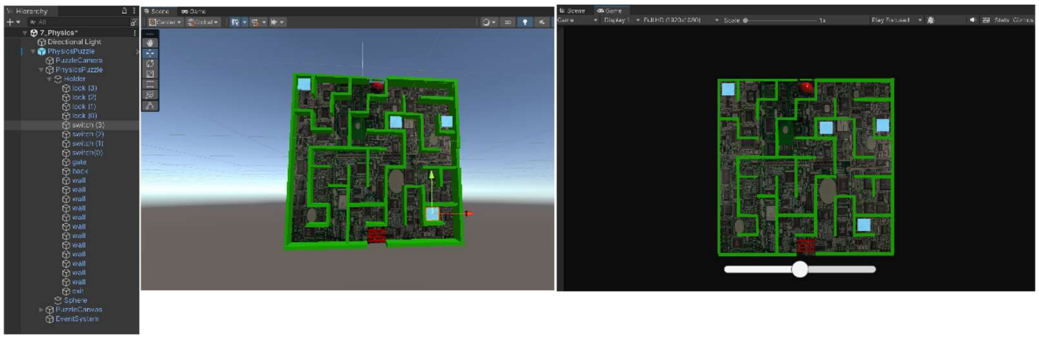


Running the scene and interacting with the sphere will now pick the ball up and add it to inventory, clicking the inventory button will show it in an alphabetically sorted list.

Whenever an item is added a flag with itemName[name] is created as true so in out case a flag called itemNameBall has been added as true, obviously very useful in our dialogs. Our dialog actions can also be used to take or give items to the player.
Note that when items are removed so are their associated flags. Remember we can use flags like hasItemFish in our conditional dialogs 😊.



Physics



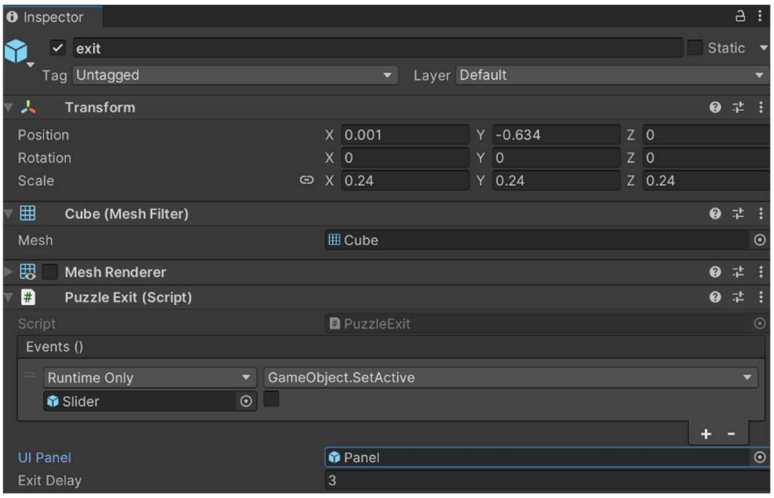
The physics puzzle can be used in a scene for many purposes, in the example it has been themed as a hacking minigame but can be rethemed to suit your needs.

The maze is comprised of four switches the ball has to touch to unlock the four bars blocking the exit. The maze rotates via the slider at the bottom and the ball falls and rolls with gravity.

The switches are simple trigger objects that could be moved anywhere in the maze for variety.

The PhysicsPuzzle prefab can simply be dropped in the required scene, it should be disabled until needed as it has it's own camera that will conflict with other cameras in the scene if left on.

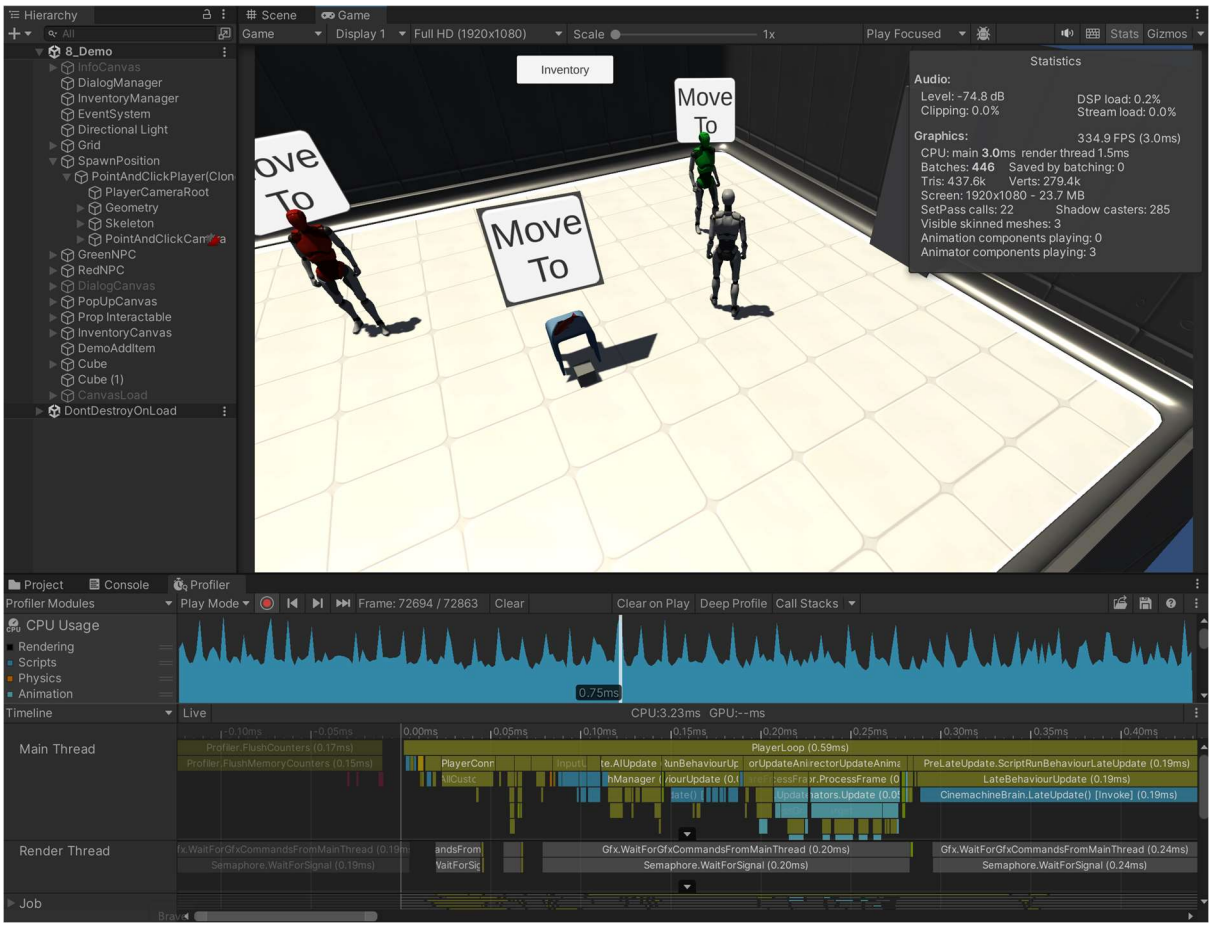
The area where the ball exits the maze (wins) has a trigger script that will display a congratulation message and can be customised via custom events to run any other logic you may need.



Testing

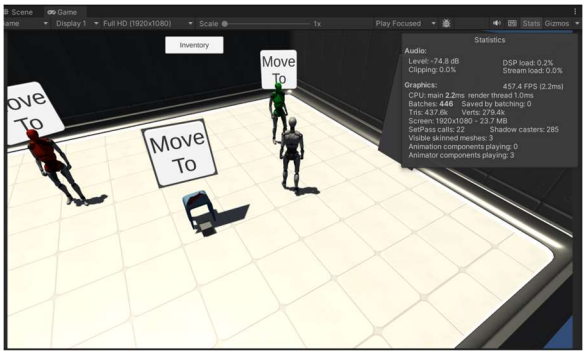
Performance Testing

As our toolkit is predominantly a scripting asset it has been tested with our most complex 8_Demo scene that contains all the components working together, in a standard URP scene with default graphical settings and one real-time light with shadows.



As you can see the overall performance is over 330 FPS and the toolkits scripting completion per frame is less than 0.75ms.

Without the overhead of the profiler the scene is hitting over 450 FPS.



So, whatever your destination hardware platform you can be confident that our toolkit won't be a problem for your performance.

Functionality

Every component has been “Black Box Tested” against a range of criteria listed below to provide consistent quality control.

All of these tests are reproducible in the Package Manager>Samples>Tutorial. download

Component	Test	Scene	Date	Passed
Grid	Create a simple grid with some tile variations.	1_Grid_Demo	17/04/24	Yes
Grid	Ensure the grid can generate a NavMesh	1_Grid_Demo	17/04/24	Yes
Navigation	Ensure the point and click character moves around the NavMesh avoiding obstacles	2_Navigation	17/04/24	Yes
Interaction	Ensure the “Move To” interaction moves the player to the interaction point	3_Interaction	17/04/24	Yes
Interaction	Ensure the “Interact” button triggers the custom UnityEvents	3_Interaction	17/04/24	Yes
Dialog	Ensure the NPC interaction displays a non conditional dialog properly	4_Dialog	17/04/24	Yes
Flag / Dialog	Conditional Dialog & Flag Setting between two NPCs	5_Flags	17/04/24	Yes
Inventory	Pick up fish and add to inventory	6_Inventory	17/04/24	Yes
Inventory	Conditional dialog and remove fish using dialog	6_Inventory	17/04/24	Yes
Inventory	Conditional dialog and adding apple to inventory via dialog	6_Inventory	17/04/24	Yes
Inventory	Alphabetically sort displayed inventory items	6_Inventory	17/04/24	Yes
Physics	Puzzle rotates with slider control	7_Physics	17/04/24	Yes
Physics	Triggers unlock door blockers	7_Physics	17/04/24	Yes
Physics	Ball triggers win condition on exit	7_Physics	17/04/24	Yes
All	Test interoperability of all components across all variations	8_Demo	17/04/24	Yes

Unit

Due to constraints around release, unity testing will be included in version 1.0.1.

Support

If you do find any issues with use of the components, please reach out via our support channel and we will work with you to resolve any issues as quickly as possible.