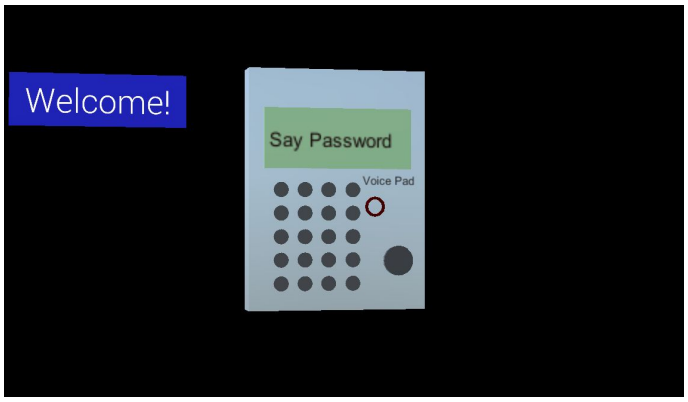
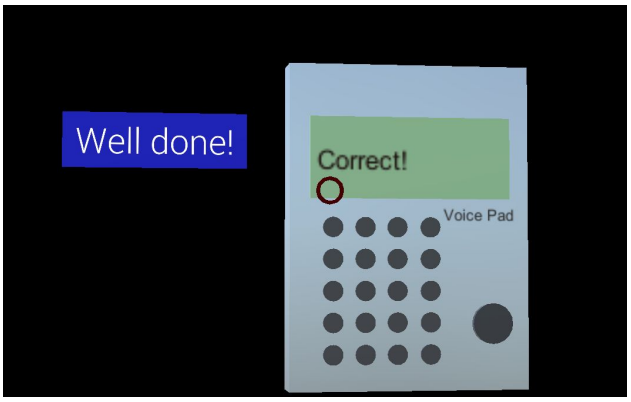
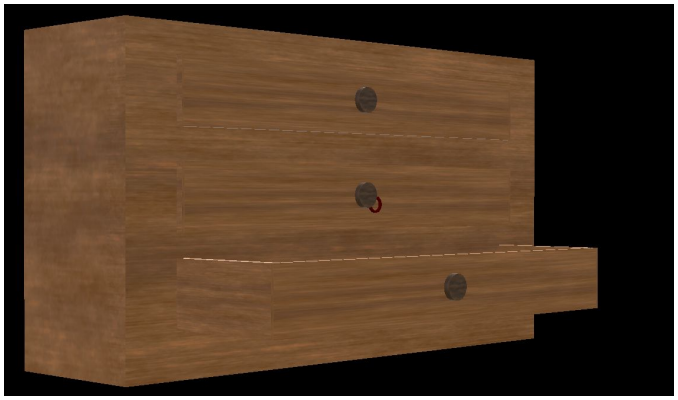
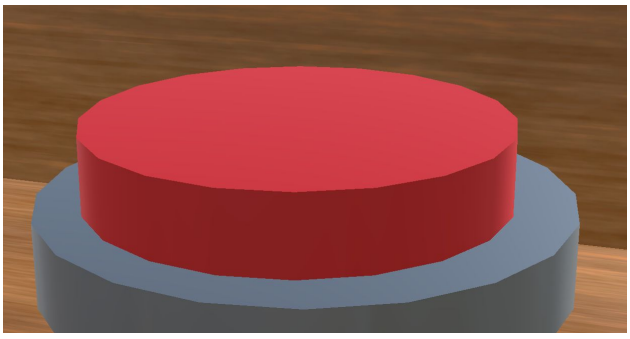
HoloLens Escape Room

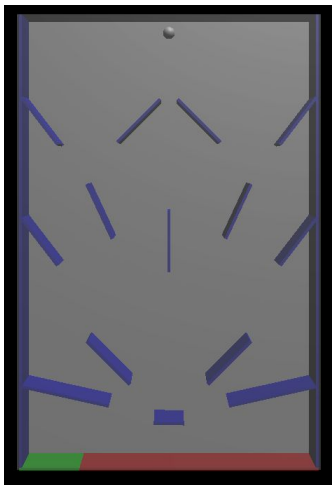
Clue Creation Guide

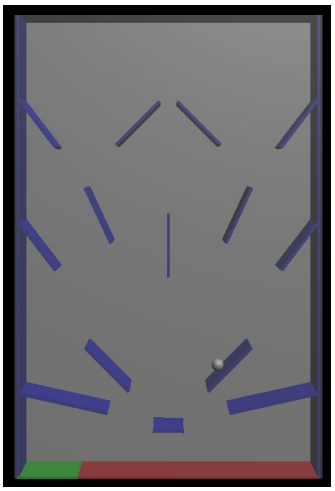








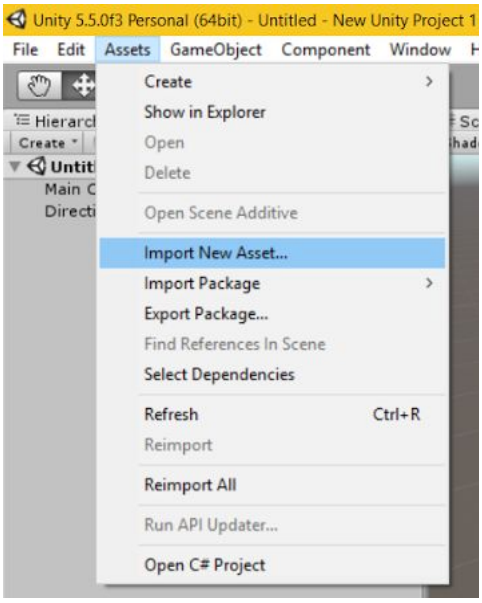




This document will help explain how to go about creating your own clues in Unity for use in creating new escape room levels. You should first read the provided classes Clue and TriggerAction to familiarise yourself with their contents.

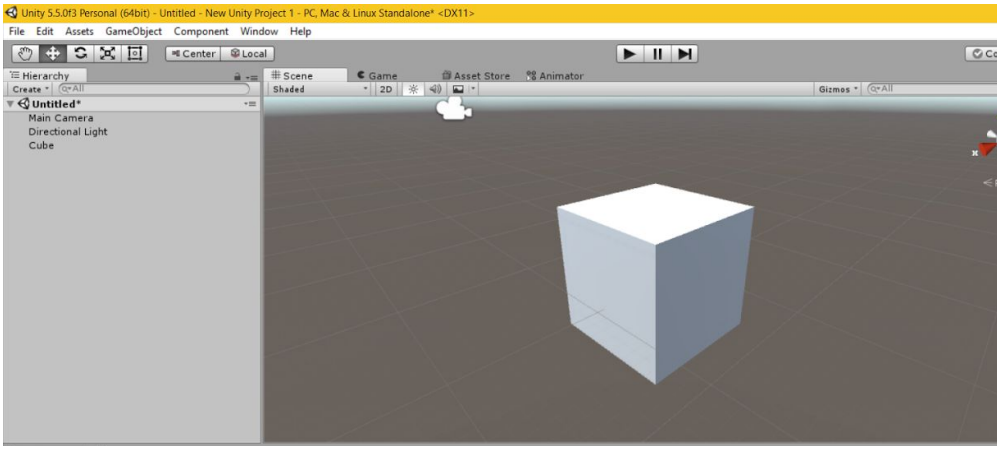
Creating a 3D Model

You will need a 3D model to represent your clue. You can either create your own from scratch, or import a pre-made model. To import a prefab, you can click on the *Assets* tab on the menu, and from the drop-down menu select *Import New Asset…* and then select the prefab or model you wish to add to the Unity project from your own files.

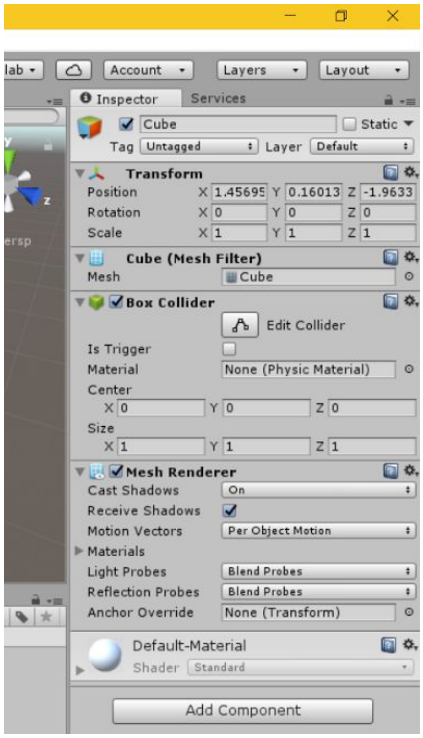


To create your own model, you can select *Create* from within the *Hierarchy* panel in Unity, and then you can choose one of Unity's standard shapes from the options in the drop-down menu.

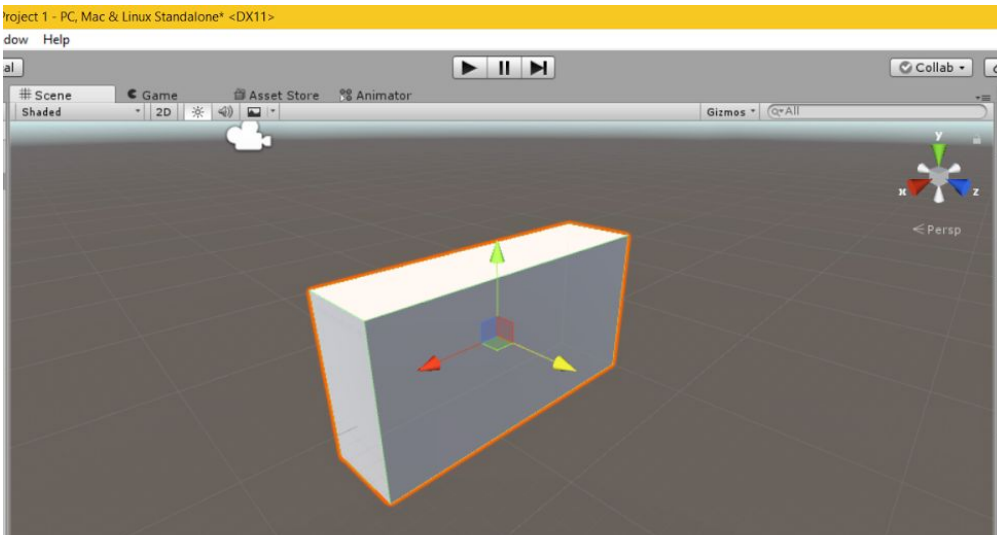
Unity will then place the chosen shape within the scene. Below is what the standard cube shape looks like.



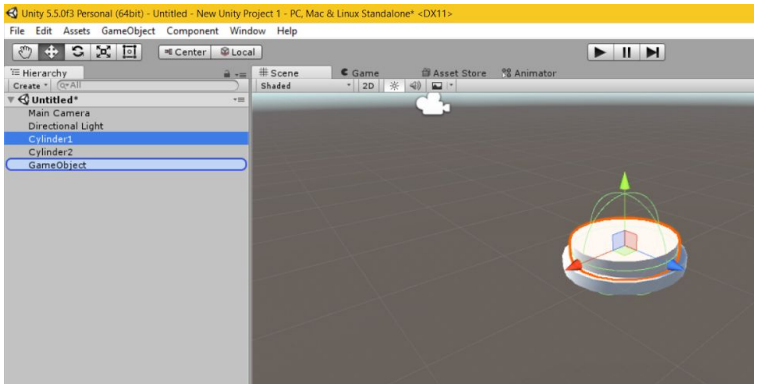
The dimensions and orientation of the object can be edited using the *Transform* option within the *Inspector* panel after selecting the object in question--there are options to change the rotation, position and scale of the object in each of the x, y and z dimensions. This allows cubes to be rotated or stretched into longer cuboids.



The position of the object can also be altered by selecting the object in the *Scene* view and clicking on the arrow in the appropriate dimension and then dragging the object along that axis.



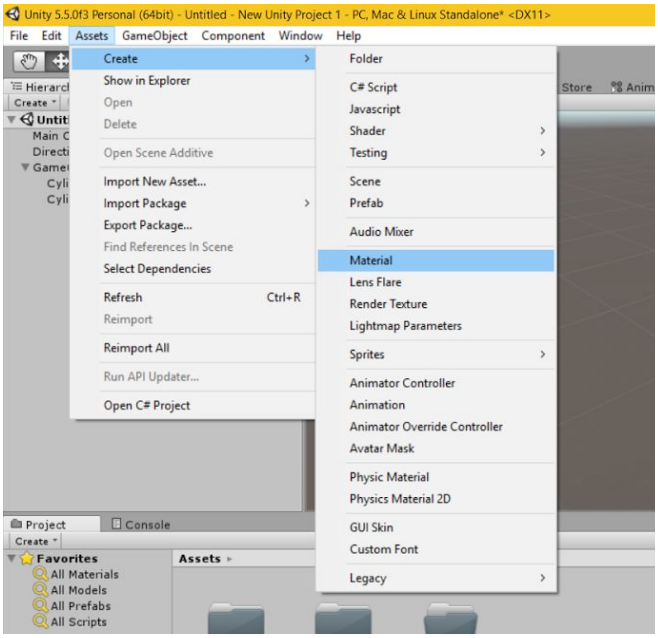
When creating clues, you will probably want to join shapes together, as it is unlikely that a clue will just be a simple cube, cylinder or sphere shape. Multiple shapes can be combined into one object by selecting *Create* at the top of the *Hierarchy* panel, and then selecting the option to *Create Empty*, and then dragging the sub-objects onto the newly created object within the *Hierarchy* panel. When the parent object is moved, all the children parts will move with it, whereas if just one of the child parts is edited, the rest of the object remains unaffected.



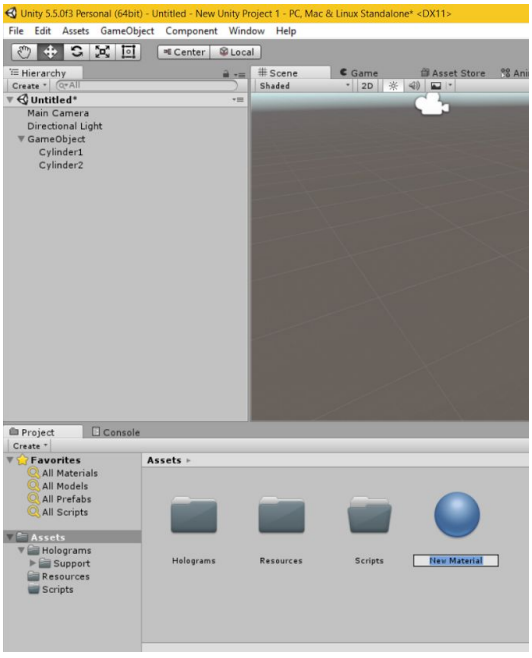
In the example above, I wish to join together two cylinder objects into one overall object (to make a button clue). After clicking on *Create Empty*, a new *GameObject* is made in the *Hierarchy* view. Both Cylinder1 and Cylinder2 have to be dragged onto the new *GameObject*, and then the result should look like the final image of the *Hierarchy* view. If the *GameObject* is moved, both cylinders will move together, as though they are joined.

Creating a Material

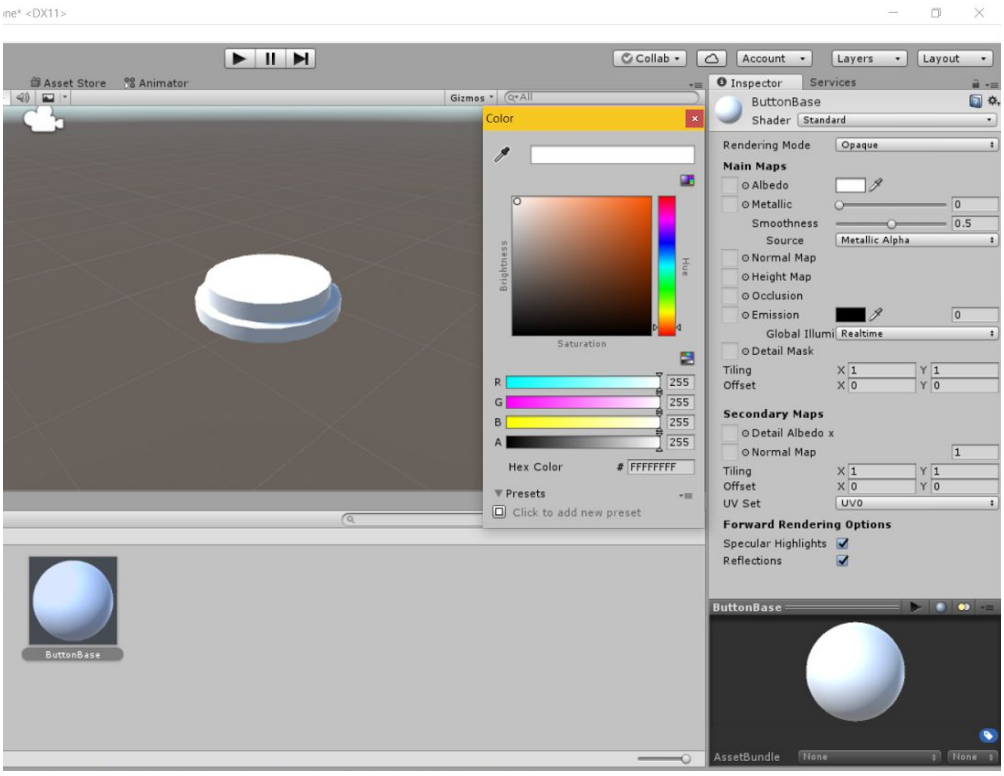
Now we have created our 3D model for our clue, we can see that Unity uses a default colour for the object. If we want our holograms to look good, we will want to add colour or texture to the object. This can be done by creating a material. Materials and textures can be purchased and imported from Unity's asset store, otherwise they can be created from scratch.



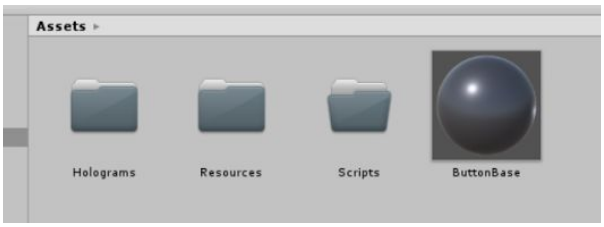
To create a material, you need to first select the *Assets* tab from the menu bar, then select *Create*, and finally select *Material* from the drop-down menu. The new material will appear within the *Project* panel at the bottom, and you can name your new material something meaningful.



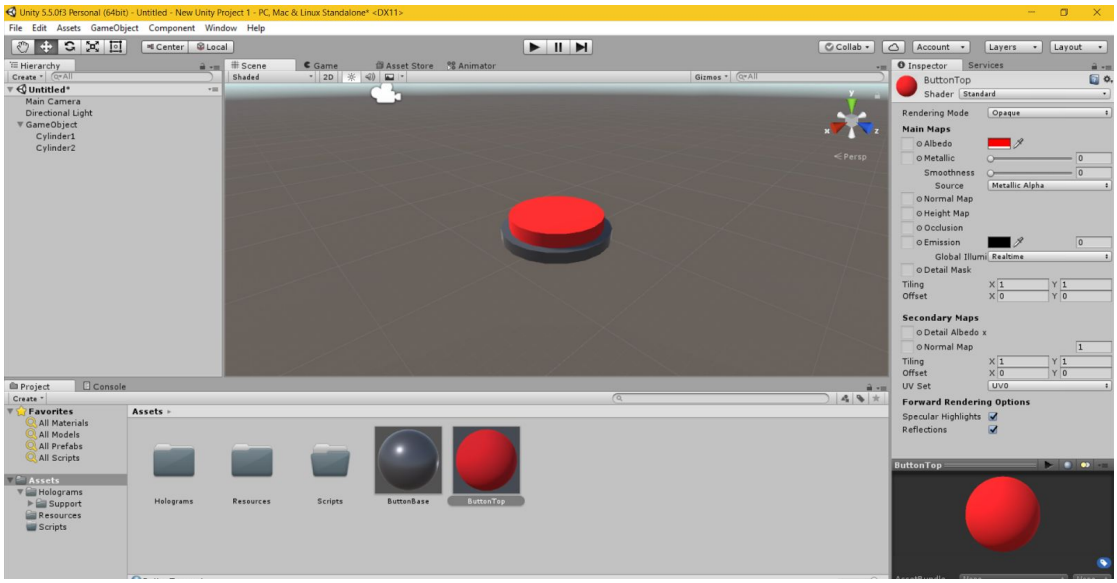
The properties for the material can then be altered within the *Inspector* panel after selecting the material. The colour can be altered by selecting the boxes showing the current colour, and then a new colour can be chosen. Other properties can be selected, such as the smoothness, or how metallic it is.



The changes made to the material will then be visible from the *Assets* panel.



You can then drag the material onto the object, either in the *Scene* view or in the *Hierarchy* panel. You will then instantly be able to see the material applied to that part of the object in the *Scene* view. Below we have added a material to the top of the button, and a different material to the bottom of the button.



Adding Scripts to Clue Objects

For the clue to be able to cause changes in other clues, a C# script needs to be added to the object. We have provided example scripts for a button, some drawers containing a button, a ball-drop game, a voice pad, and also a screen that displays text. To be able to trigger other objects, a clue needs to have an action added--this can be added using “AddAction”, and then giving the action a name. This also requires a GameObject and a TriggerName as extra arguments, but we do not want to specify what these are as we want these to be able to be set dynamically in the level editor--in this way, a button can be linked up to trigger the text on the screen to change, or to instead control the ball-drop game. This means we have to write “public GameObject ObjectToTrigger” and “public string TriggerName” so these can be set outside. Then, when the current clue needs to trigger another clue, it calls “Trigger(“OnSelect”)”, where the “OnSelect” can be replaced by whatever action name was chosen when the action was added, and will then find the correct ObjectToTrigger and send a message to it with the TriggerName.

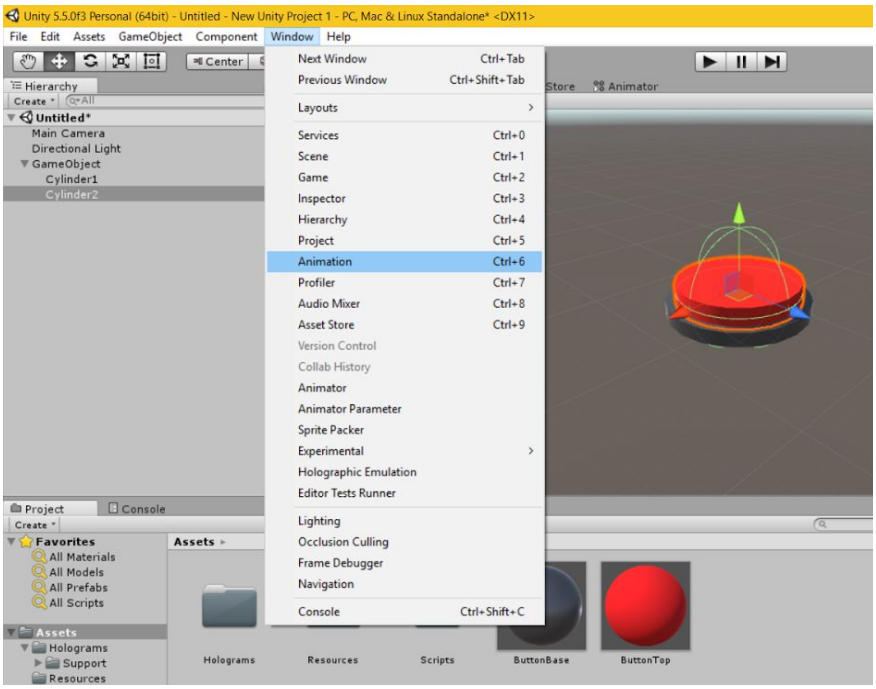
All clue objects will have a SetVisible and SetInvisible action and a visible property (a public bool), as this will give the editor the option to choose whether clues are visible from the start, or only appear once another clue is solved, or disappear when solved. This is already implemented by the Clue class, so does not need to be specifically added to the individual Clue script.

All clue objects will need to have a “private new void Initialise()” function, and this sets all the properties for the clue. This can be seen in the scripts for the example clues. You will need to add this in any clues that have properties that can be set.

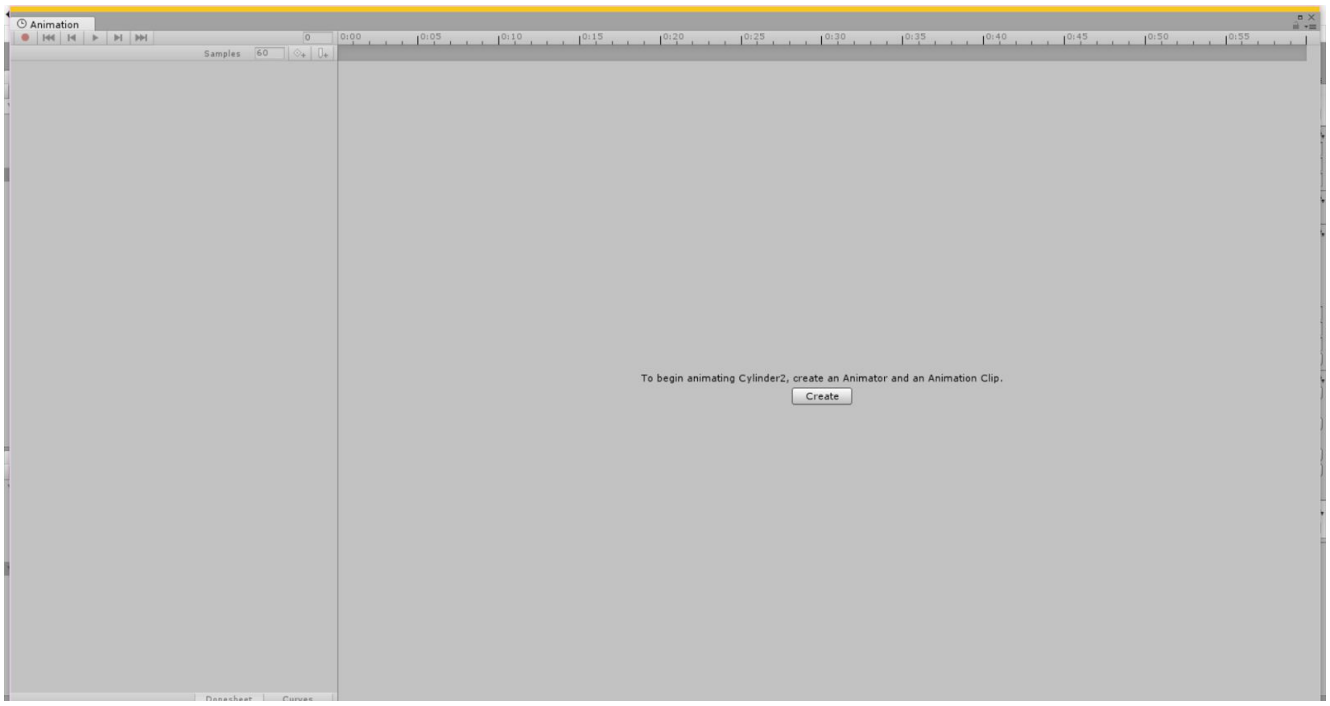
Adding an Animation to a Clue Object

To make our clues more interesting, we may wish for them to play an animation when we interact with them--for instance, when we select a button, it would make sense for the button’s top to move down and then back up, as if it were actually pressed. Similarly, when we open a drawer, we expect the drawer to smoothly open, rather than instantly jumping to the position with the drawer being fully open.

To add an animation, you need to select the specific part of the object you wish to animate, and select the *Window* tab on the menu bar at the top of Unity, and select *Animation* from the drop-down menu.



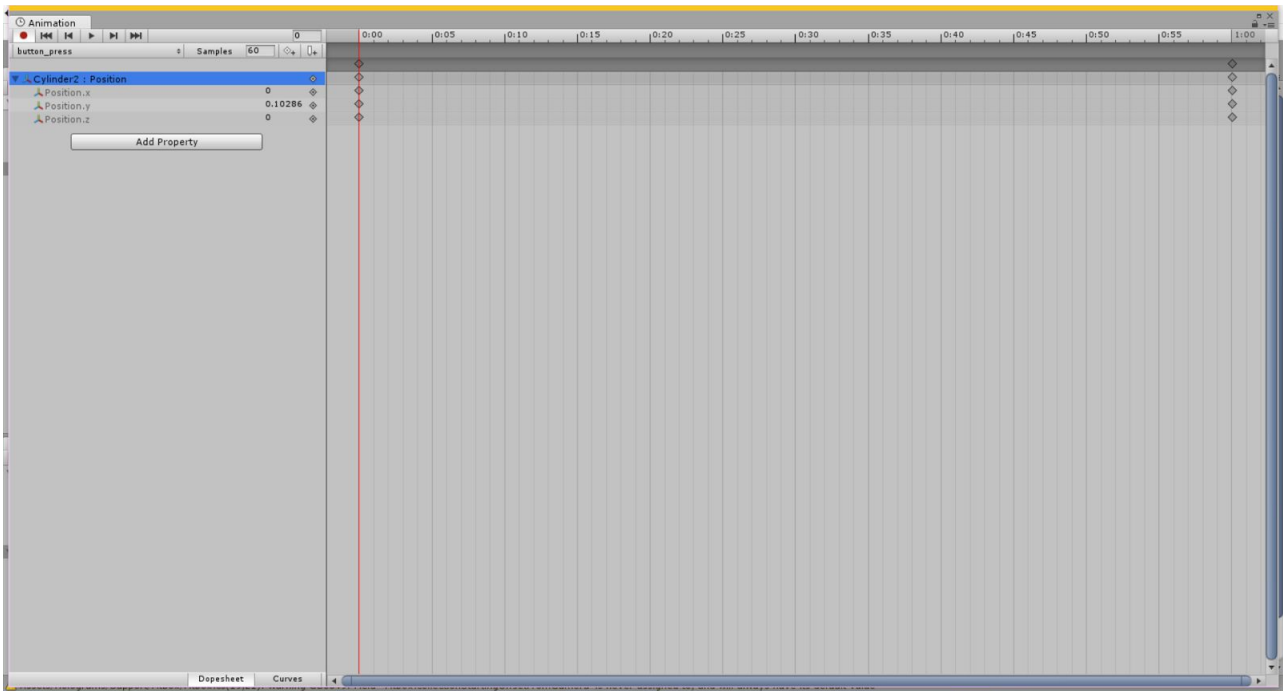
You will then be presented with the *Animation* window, prompting you to create a new animation for the object.



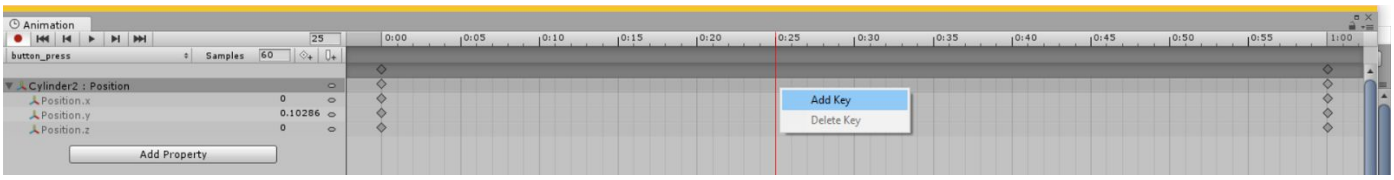
You should click on the button saying *Create*. It will then let you choose where to save the animation file, and give it a meaningful name.

You should then click the *AddProperty* button, and then there are a variety of options to choose from for animating the object, such as scale, position and rotation. For a simple movement

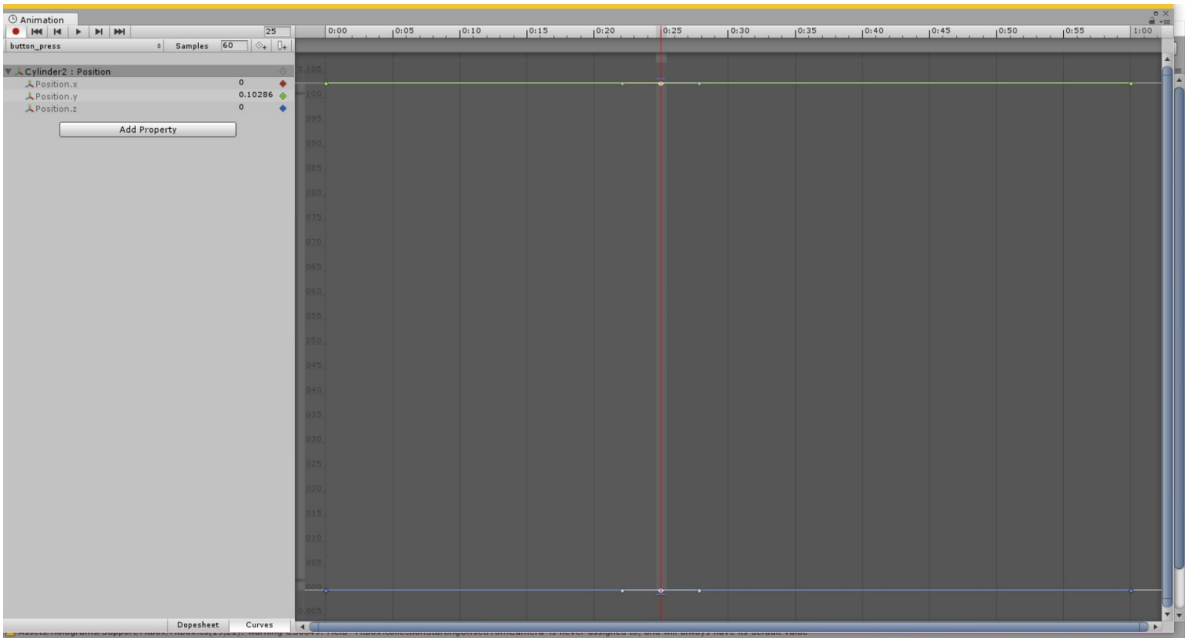
animation, you can select *Transform*, and then select the + next to *Position*. You will then be presented with a view similar to that shown below.



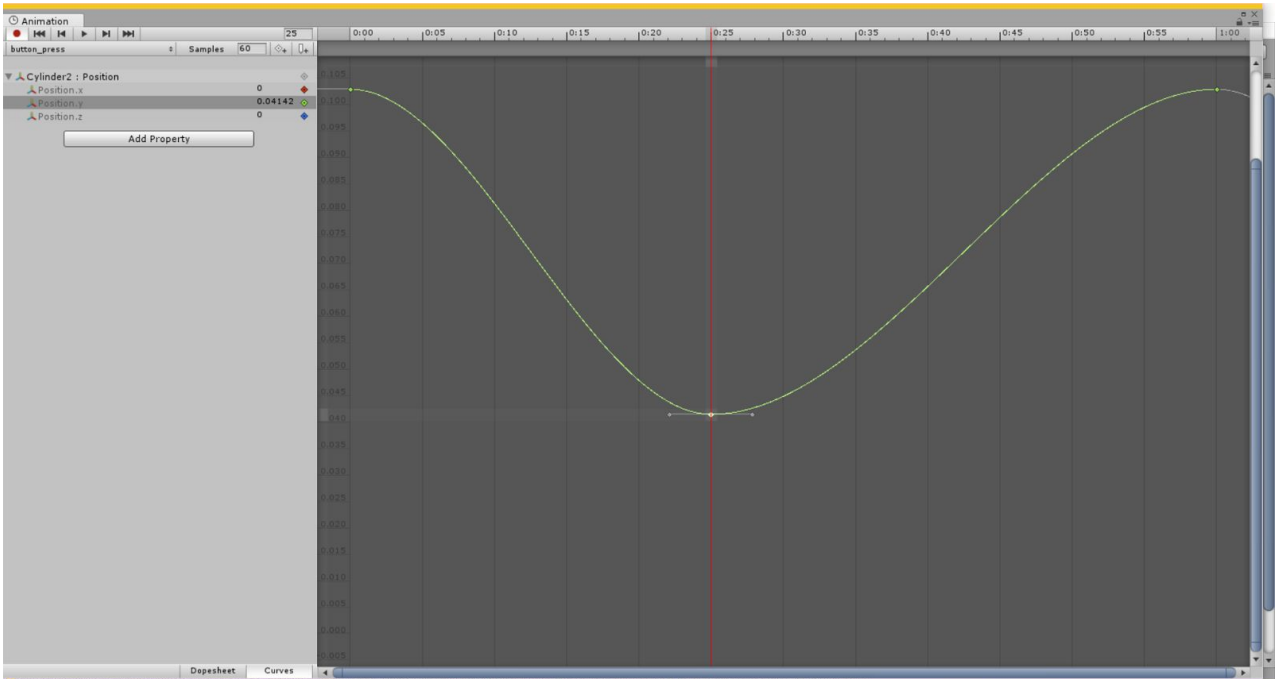
Initially there are nodes at the start and end of the animation, but more can be added at intermediate times by right-clicking and selecting *AddKey*.



This will add some nodes at the chosen time. To then edit the animation itself, it is often easier to go onto the *Curves* view, by selecting the button saying *Curves* to the right of *Dopesheet*. You will then be presented with a view containing flat lines representing each of the dimensions of the object.

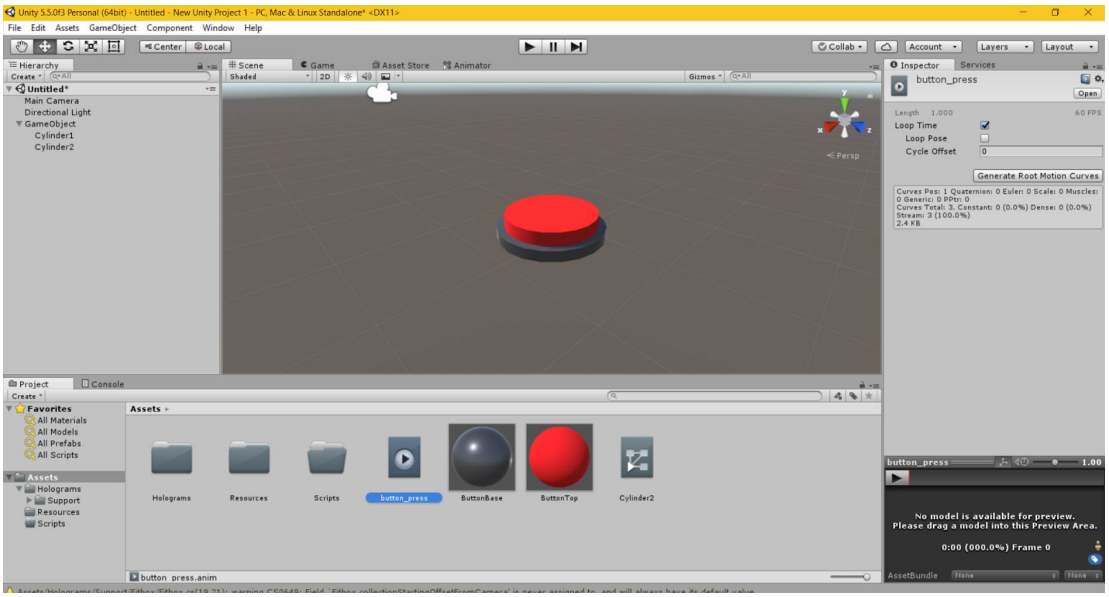


You can then click and drag the position of the object at a node, creating a smooth curve representing the animation. There are curves for x, y and z for position animations.

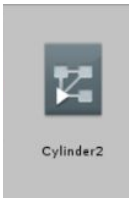


The above animation represents the object moving down and then back up, in the y direction. If the object needs to move along multiple axis, the curves for x and z can be edited too. Once

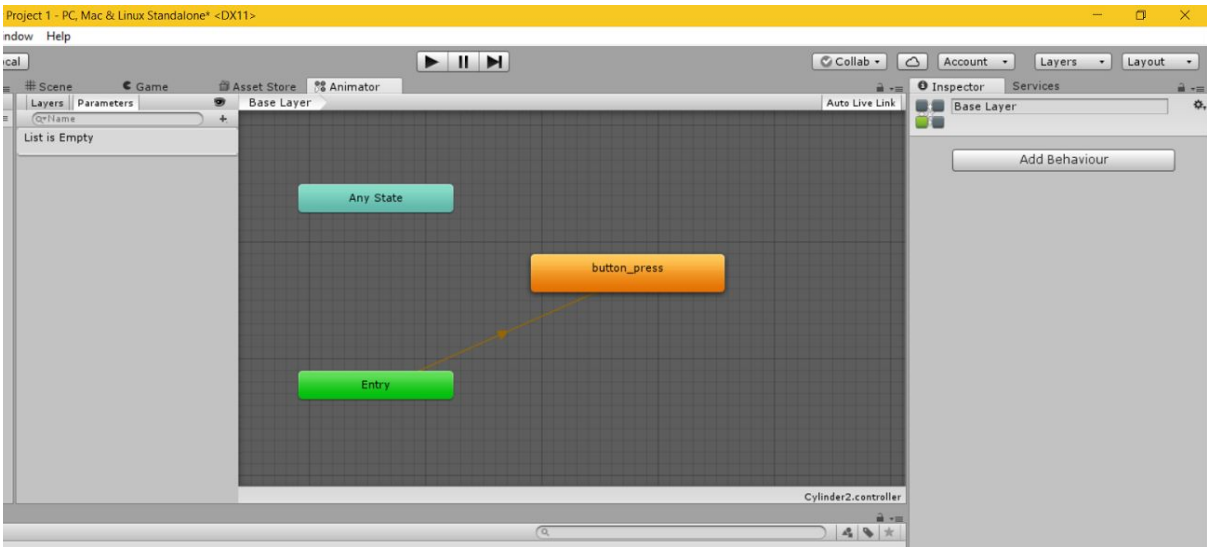
you are happy with your animation, you can close the *Animation* view. The animation file and an animation controller will now be within the *Project* panel at the bottom of Unity. By default, the animation will play on loop, so if you only want the animation to play once, you will want to disable looping--to do this, you need to click on the animation file in the *Project* panel, and then in the *Inspector* panel on the right, you need to untick the box that says *Loop Time*.



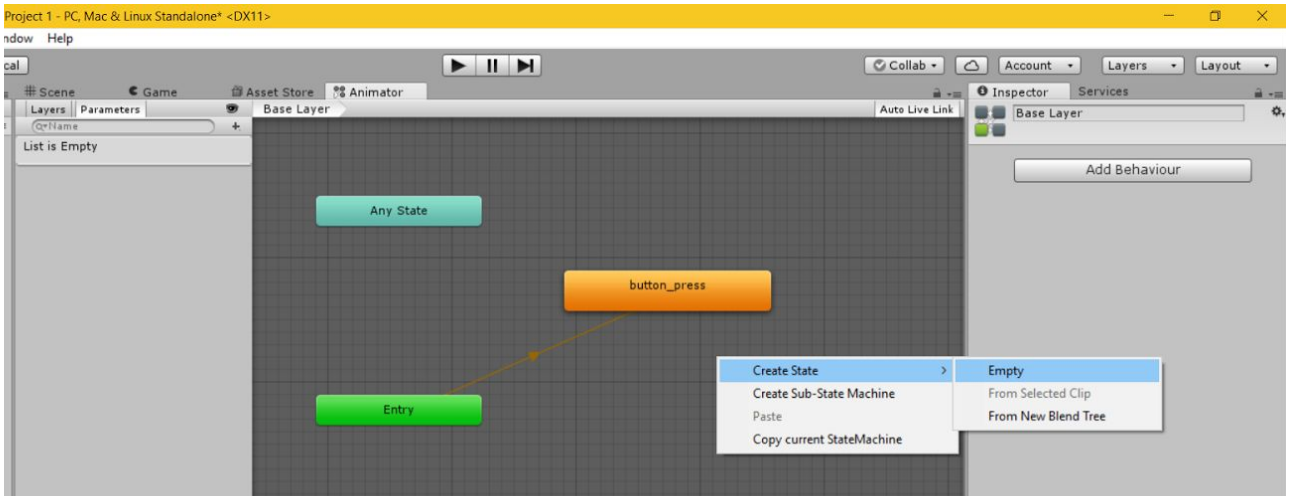
You then need to adjust the animation controller so that the animation doesn't instantly play on startup, and only plays when a chosen trigger occurs. You should click on the animation controller, which will, by default, have the name of the object you decided to animate.

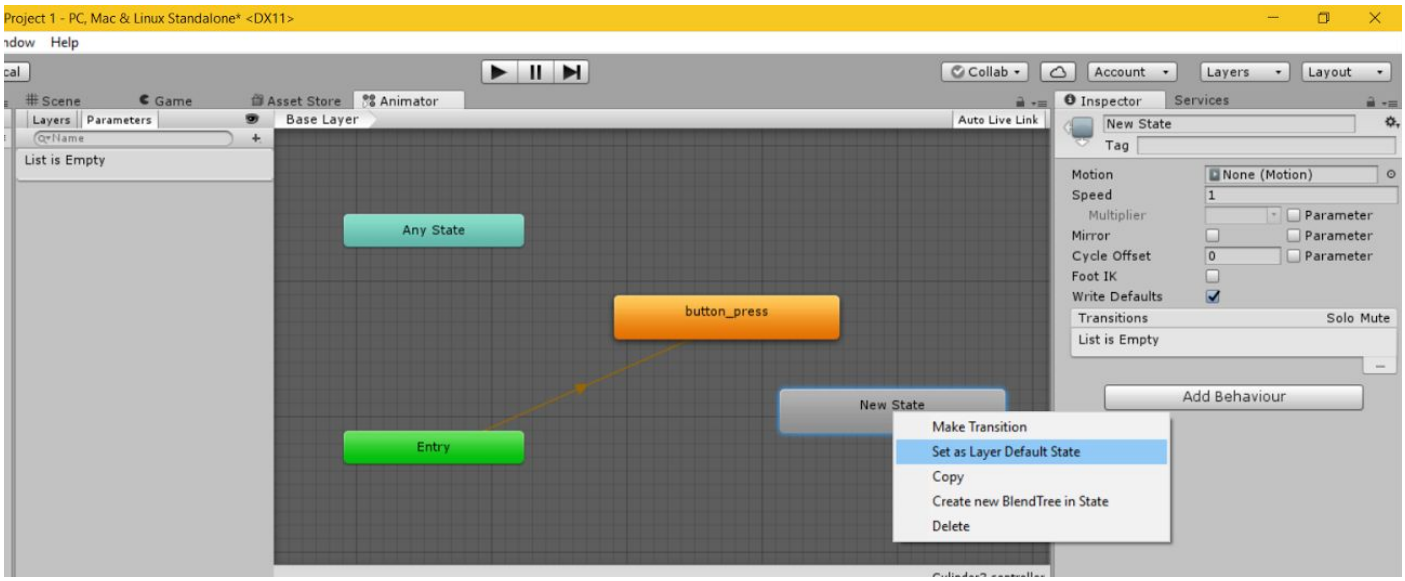


You will then enter the *Animator* view, which will contain a state machine for the animations of that object.

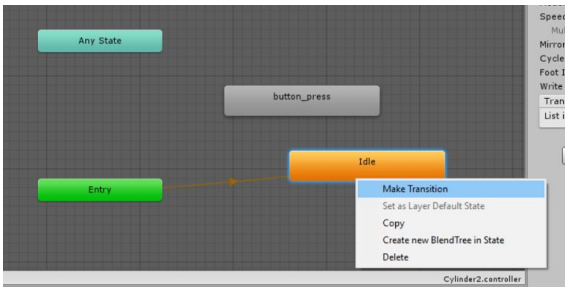


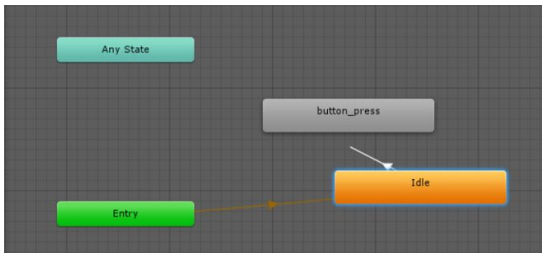
The orange box will show the animation you just created, as it will be the default state. However, since the object instantly transitions from the *Entry* state to the animated state, the animation plays straight away. If this is not the behaviour you want, you need to add a new state by right-clicking anywhere on the *Animator* view and selecting *Create State*, and then clicking *Empty*.

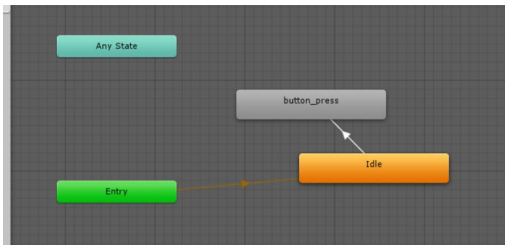




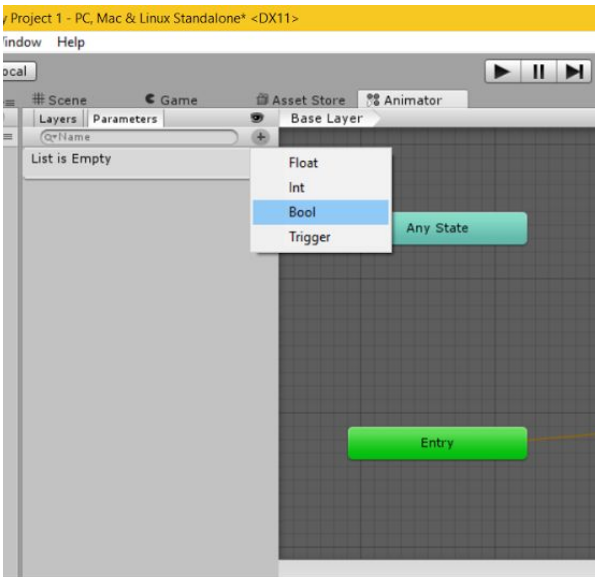
Then, once the new state has been created, you can right-click on it and *Set as Layer Default State*, which will change the new state orange, and the animation state will become grey. The new state can be renamed, preferably to something meaningful such as “Idle”, as this will be the state where the object is unmoving. Transitions then need to be added between the new state and the animation state, and these can be added by right-clicking the current state and selecting *Make Transition* and then dragging the arrow that appears to the animation state you wish to transition to.



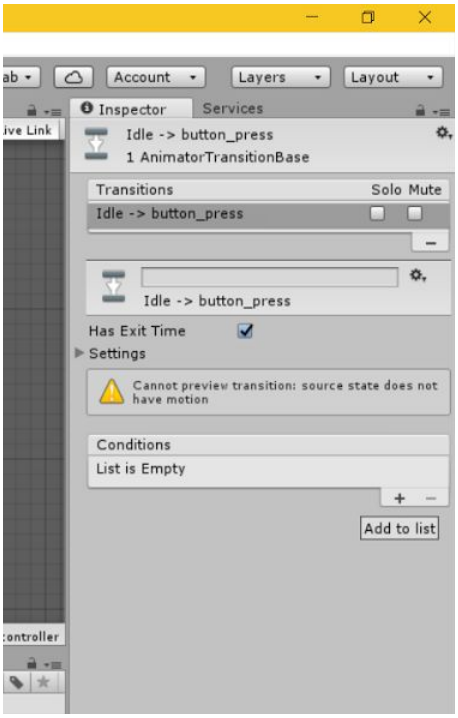




For the transition to only occur on specific events, a parameter needs to be added. This can be done in the *Parameters* tab, and clicking on the + and then your desired parameter type--bool is a good choice.

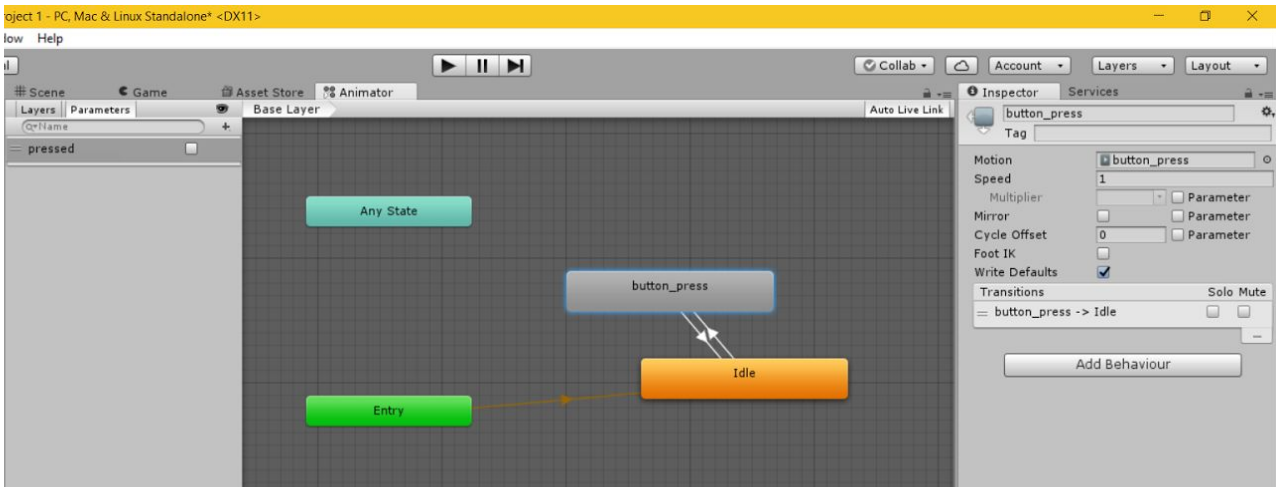


To make the transition only happen when that parameter is true, you need to select the transition in question and then in the *Inspector* panel, and then click the + on the *Conditions* list at the bottom.



You can then choose whether you want the transition to happen when the parameter becomes true, or when it becomes false. In the object script itself at the moment you wish the animation to occur, you need to then add “GetComponent<Animator>().SetTrigger(“pressed”);”, where

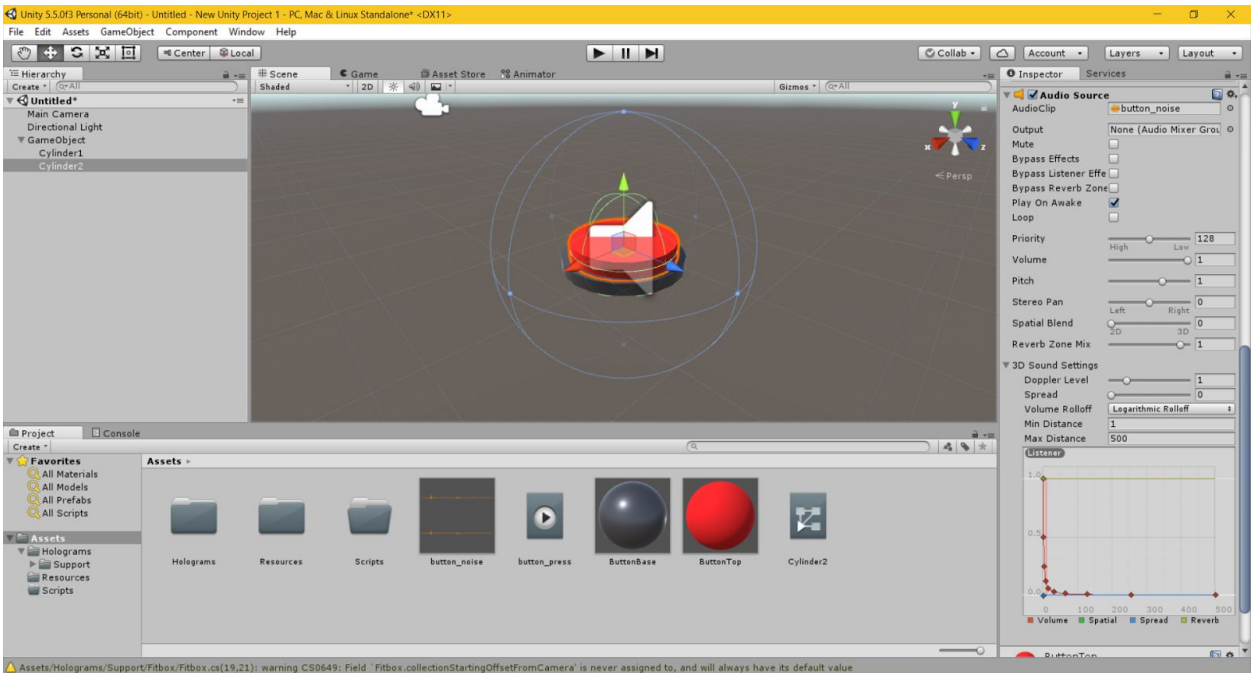
“pressed” can be replaced with your chosen parameter name. ResetTrigger can be used for resetting the parameter’s value. You can also set transitions to happen when the previous animation finishes (on exit time), so once the animation happens it will then transition to the idle state.



Eventually you will have a complete state machine, potentially containing multiple animations and transitions between states.

Adding Sound Effects to a Clue Object

For some clues, you may wish for a sound to play upon an event--for instance, we may want a button to make a clicking noise when pressed, or we may want the the drawers to sound like they are actually being opened. First, the sound effect file (.wav) needs to be imported, and this can be done in the same way other assets are imported--by clicking the *Assets* tab on the menu and then selecting the *Import New Asset…* option from the drop-down menu, and then selecting your own sound file. You then need to drag the sound effect onto the object that you wish to play the noise.



You can then scroll down in the *Inspector* tab to the *Audio Source* section, and you can choose whether you want the sound effect to loop, or play on awake and so on. To make the noise play only when you want it to, within the script for the object, you need to write “AudioSource audio_name”, and then set “audio_name = GetComponent<AudioSource>();”, and then at the moment you want the sound to play, do “audioname.Play();”.

Adding Physics to a Clue Object

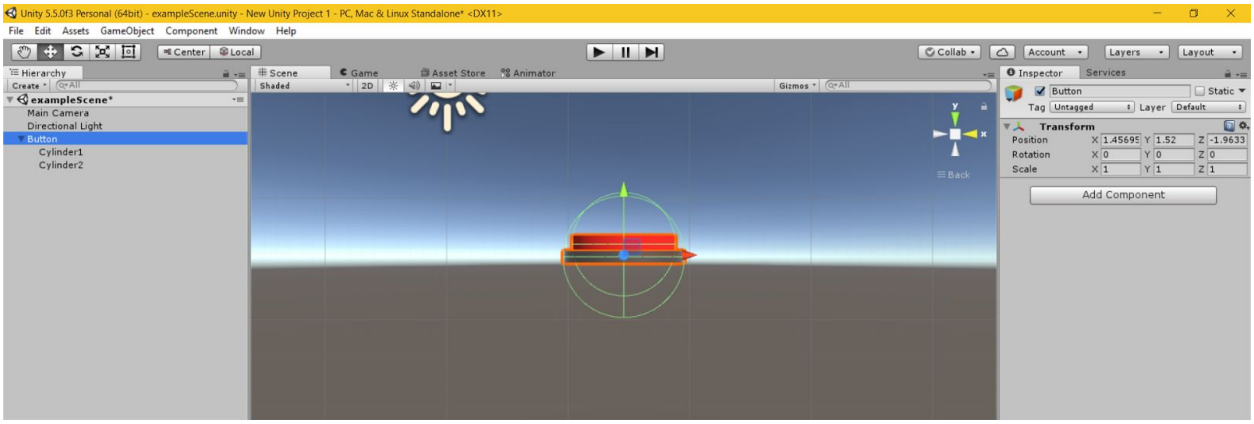
Some clues may require physics for them to work, and this can allow for some more creative clues--for instance, the ball-drop game uses physics on the ball itself, so it can fall realistically and interact with all the platforms within the game. Something like a tiltable-maze containing a ball could also work in a similar way. The BallDrop script shows one way of using the physics engine--the rigidbody component needs to be enabled, and this allows the ball to fall. It also needs to detect collisions with other objects so it doesn’t just fall through them. For more information, take a look at Unity’s documentation on the physics engine: https://docs.unity3d.com/Manual/PhysicsSection.html

Adding Voice Commands to a Clue Object

You may also want some clues to respond to voice input. This can be done using keywords, and then the object will listen out for those specific keywords, and upon hearing a specific keyword, will trigger an action. The VoicePad script shows how a keyword (in this case, a password) can be used to trigger something else when the correct word is heard. Multiple keywords can be added so clues can respond in different ways to different words.

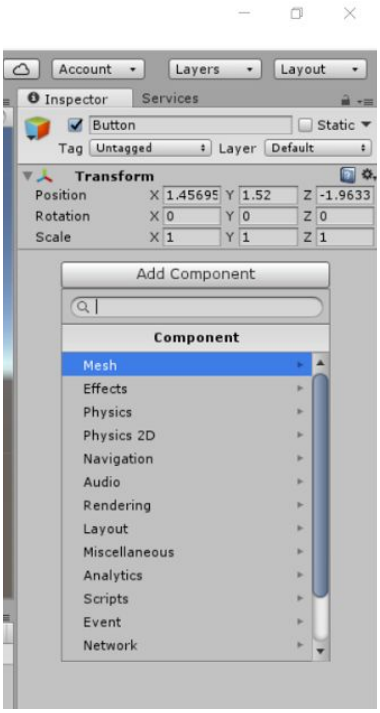
Adding a Box Collider to a Clue Object

To be able to be placed correctly by the clue placement program, each clue object needs to have a box collider added. Some Unity shapes, such as the cube, already have a box collider, but for others these will have to be specifically added.



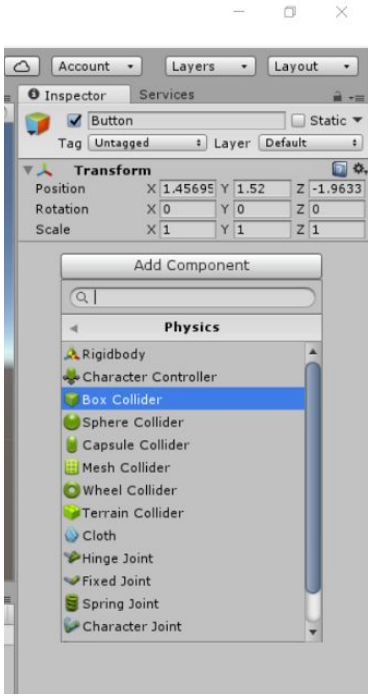
To do this, you need to make sure you select the overall clue object (rather than a specific part of it), and then in the *Inspector* panel, you need to click on the button that says *Add Component*.

This will then give you a menu containing many different components.

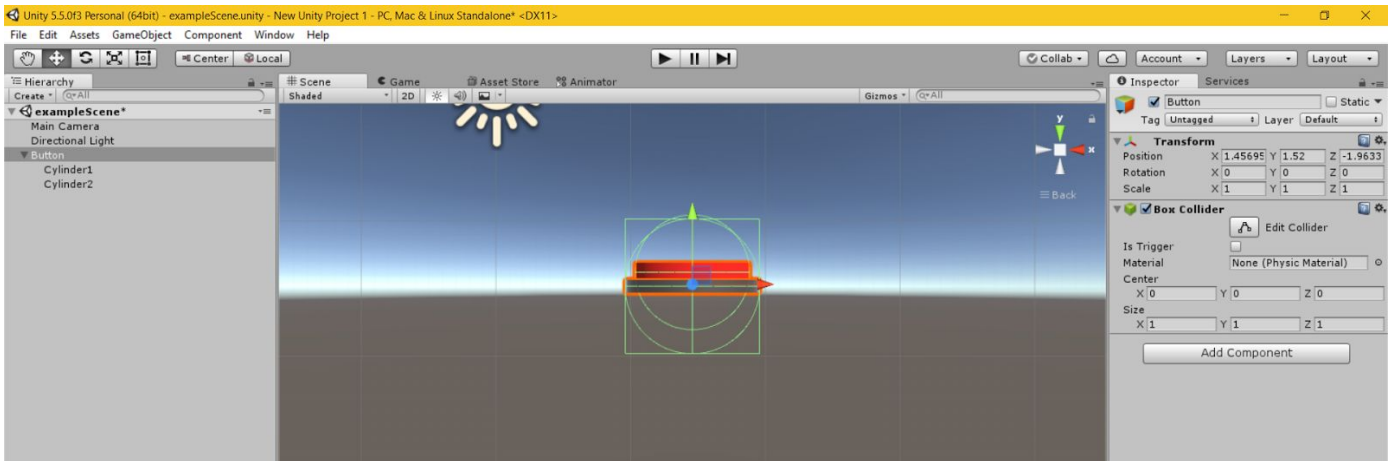


To find the box collider, you can type in “box” in the search bar at the top, and then select the option *Box Collider*.

Alternatively you can select the *Physics* option from the initial menu, and then select the option *Box Collider* from there.

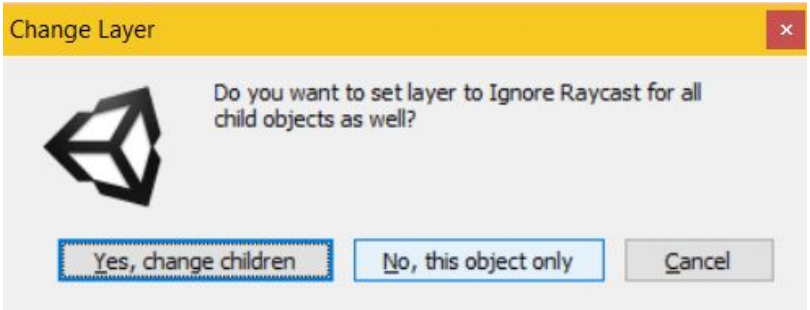


This will then add a box collider to the object. You should make sure that the box collider is centred at x, y and z all being 0, and then you should adjust the scale of the box collider so that it closely bounds the clue object. This is necessary because the clue placement program places the box collider, so if there is a lot of space between the back of the clue object and the box collider, if the box collider is placed on the wall, it will appear that the object itself is actually floating in front of the wall, rather than on it. To adjust the size of the box collider, simply edit the x, y and z values under the *Size* heading in the *Box Collider* options in the *Inspector View*. You should be able to see how big your box collider is by looking in the *Scene* view. If changing the size of the collider still leaves a gap between the edge of the object and the box collider, you can just move the individual child parts of the object closer to the box collider's edge, as moving the child objects will not move the parent containing the box collider.



However, once the box collider has been added, it can interfere with gesture commands directed at the individual parts of the object. This means that the box collider needs to be ignored during raycasts, so to fix this, you need to click on the *Layer* option in the *Inspector* view and then select *Ignore Raycast* from the options given.

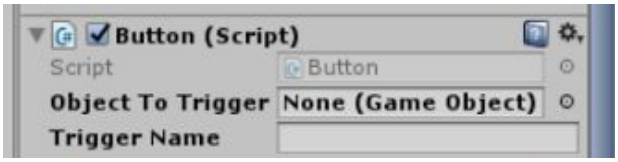
You will then be presented with the option to set all the child parts to also be in the *Ignore Raycast* layer. However, you will want the individual parts of the clue to respond to gestures and so on, so the second option, *No, this object only*, should be selected.



If you need the whole general clue object to react to something rather than just an individual part of the object, you can skip the step that sets the box collider to the ignore raycast layer--instead you can leave it in the default layer.

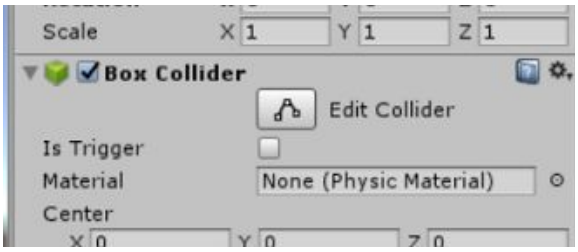
Testing a Clue Object

The clue object needs to be tested to see that it works before it is uploaded to the server. You can test clues by creating a build and running it in the HoloLens Emulator. To test if clues trigger each other correctly, you can set their properties and what object you want them to trigger in Unity.



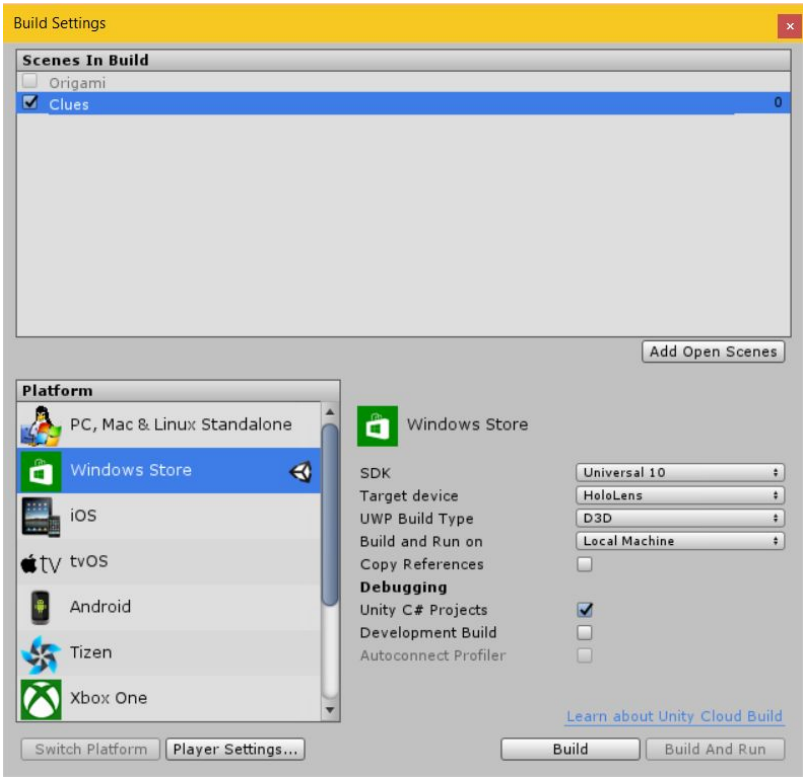
You should find the object where you attached the script in question, and then if there are any objects to trigger, you should click on the circle next to the option and select another clue you have made from within the same scene. You should then name the trigger appropriately so that the next object is triggered.

If the script contained any string properties that could be set, such as text to be displayed on a screen, or the correct password for a voice pad, these will be in the options and can be typed in the appropriate box. Any boolean values such as visibility can be set using the tick box.



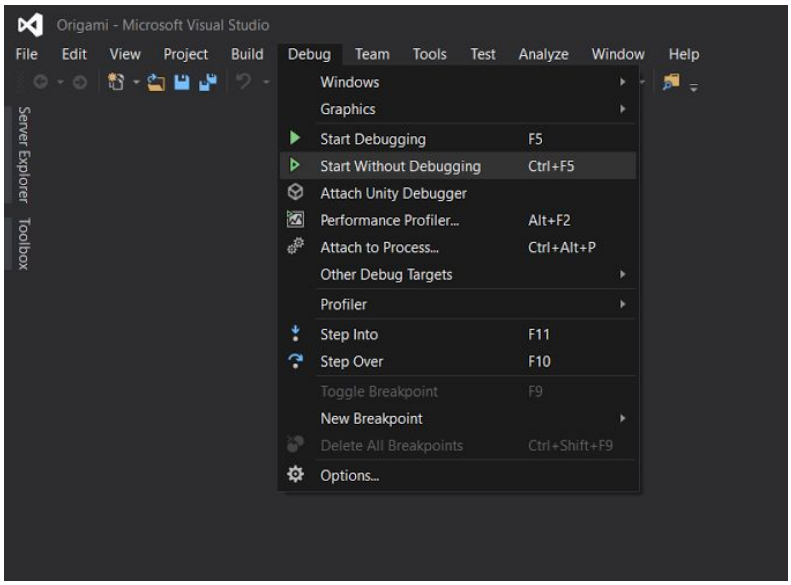
Once you are satisfied you have linked up all of the clues correctly so that you can test that they trigger each other properly, play their animations and sound effects and so on, you can then click on *File* on the menu bar at the top of Unity, and then click on *Build Settings…* in the drop-down menu.

Then you should make sure that the scene containing the clues you have made is in the build (you can use the *Add Open Scenes* button), and make sure all the settings are as shown below--the *SDK* should be Universal 10, and the *Target device* should be the HoloLens. You should then click on the *Build* button.



You should then open the newly created file in visual studio, and make sure that it is set to *Release*, *x86* and *HoloLens Emulator* at the top, underneath the main menu bar.

You can then click on *Debug* option in the top menu bar, and then select *Start Without Debugging* from the drop-down menu. The HoloLens emulator will then start running your build, and you can then try out interacting with your clues and seeing if they work as expected. If you want to test out voice activated clues, you will need to ensure that your microphone is enabled, otherwise these will not work as the keyword recogniser will never hear the word.

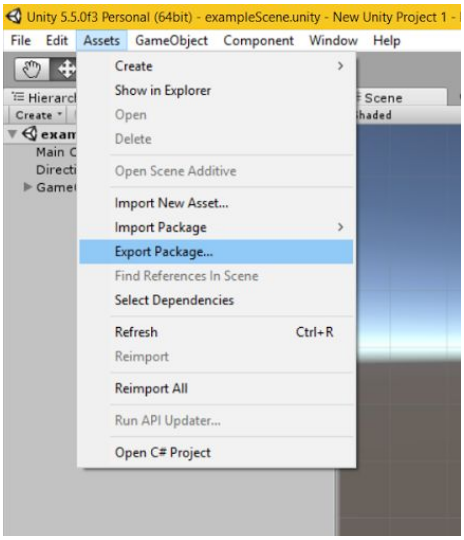


Creating a JSON File for a Clue Object

Once the clue has been created, a JSON file for that specific clue needs to be made as this is what the editor uses for creating new levels. The JSON file needs to include any properties or actions that can change--such as passwords that can be set, the clue's visibility, what available actions the clue object has for triggering other objects, and any triggers the clue object itself has that it can be triggered with. The JSON file should also include a helpful description of what the clue is and what it will do. The editor will then be able to decide what to set these properties as, and will link up the clue to other clues so it can trigger them. Some example JSON files have been provided as a guideline, and these correspond to the example clues that were created. Default values should be given for each property to be used if the value is not set. All clues will have a visible property, and this will have a default value of true--clues should only initially be invisible if specifically decided in the level editor. The default values should be given as a string. The available types for use in the JSON are boolean, string and number. The clue should also be given a unique identifier so it can be told apart from other clues with similar names--for instance, there may be multiple padlock clues that all look similar, so each one needs its own unique identifier.

Exporting a Unity Package

Once the clue is finished, all of the relevant files will need uploading to the server, so a Unity package needs to be created containing the clue. This can be done by clicking on *Assets* on the menu bar, and then selecting *Export Package…* from the drop-down menu.

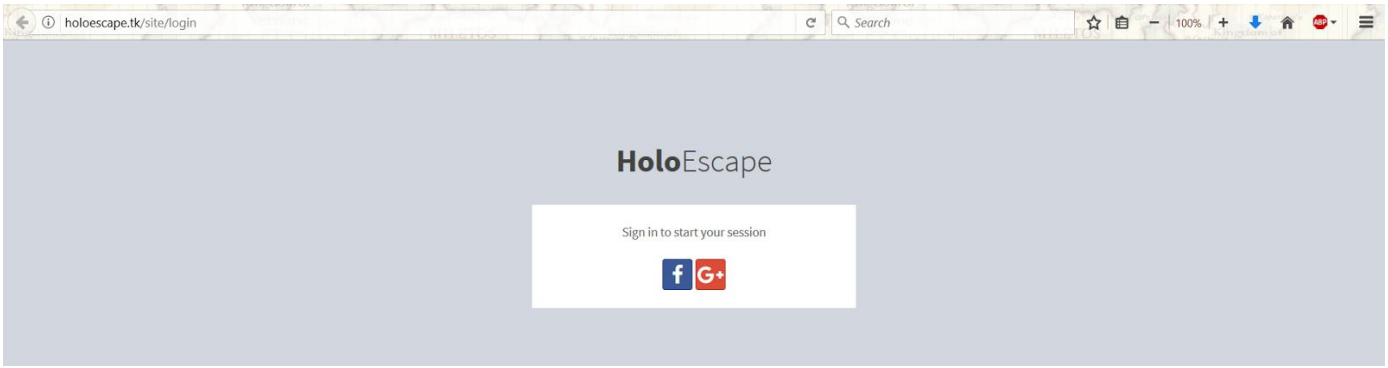


The *Exporting package* window will then open, and you can select all the files that need exporting--make sure that the boxes next to those are ticked.

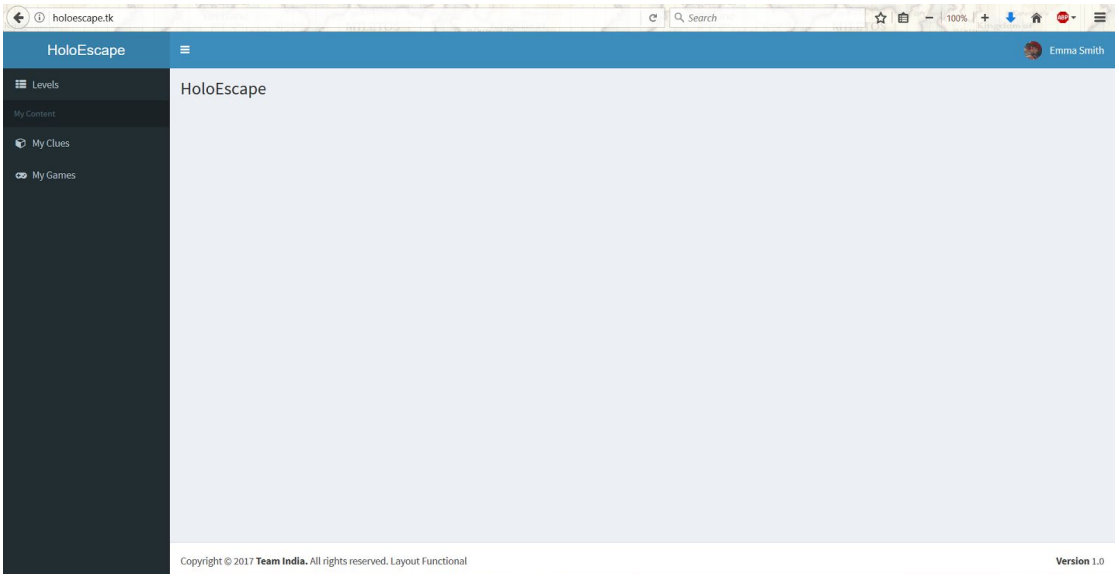
Once you are satisfied that the package will contain everything it needs, you can click on the *Export…* button to export the package.

Uploading a Clue to the Server

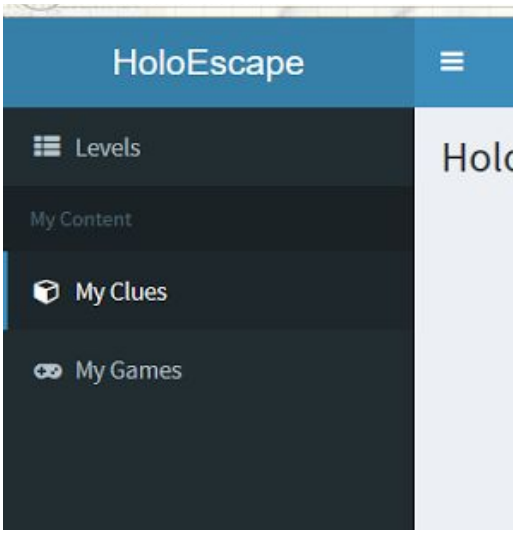
Once the clue has been completed, it needs to be uploaded to the server so that levels can be created using it. You need to go to holoescape.tk and then login using either facebook or a google+ account.



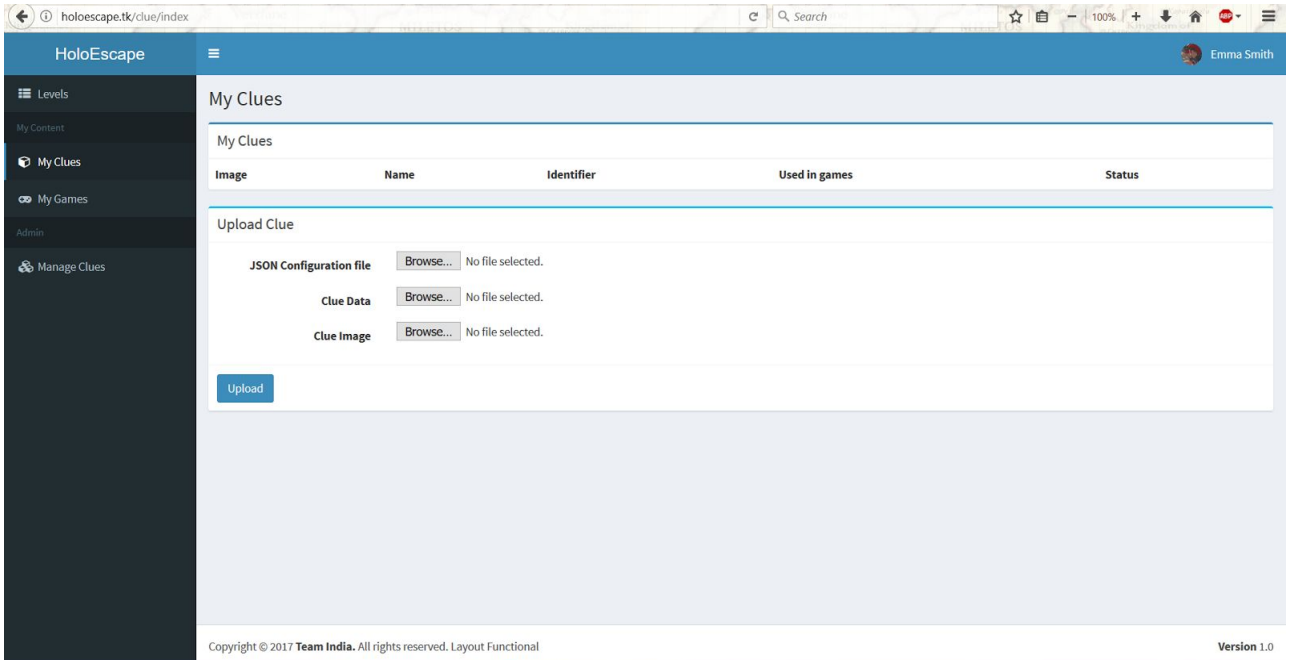
Once you have logged in, you will be presented with the home screen.



To upload your clue, you need to click on the *My Clues* tab on the left-hand side of the page.



You then need to upload the JSON file for the clue where it says *JSON Configuration file*, and upload a Unity package containing all the relevant files for the clue where it says *Clue Data*. You should also upload a picture of the clue where it says *Clue Image*.



Once both have been added, you then need to click on *Upload* to upload both files.