

Blog to Game Builder

Thanks for checking out Blog to Game Builder! This documentation will show you how to get the most out of this tool!

This is the system that I used to create [In My Mind](#), an autobiographical experience about living with bipolar. I used this system to pull down all my blog posts and plug them directly into the game, saving me a bunch of time!

So far this system supports blogs hosted by squarespace or tumblr, but depending on demand I may expand that.

Feel free to contact me with any issues/feature requests either via my twitter - [@charlietheGfish](#) or by email - charliefranciscassidy@charliethegoldfish.com

You can support me and my work either via [Patreon](#) or buying me a [coffee](#).

I'd also like to take this opportunity to give a **Trigger Warning for self harm, suicide, eating disorders and withdrawals** - as this documentation and the tool itself makes mention of them.

Table of Contents

[Blog to Game Builder](#)

[Table of Contents](#)

[Getting Started](#)

[Custom Trigger Warnings](#)

[Custom Fonts](#)

[Updating the UI](#)

[Updating Art Assets](#)

[Adding in Sound Effects](#)

Getting Started

Locate the **MainScene** and load it up. This is where everything is setup. If you hit play you'll be able to have a quick look at what the game runs like. To help with showing off how the game works I have pre-loaded it with an example blog.

The main things that we'll be using are:

- **TextLoader** which can be found under **GameControl** in the **MainScene**
- **DialogHolder** also under **GameControl**
- **GameUI**

This project already come setup for the following trigger warnings:

- Self harm
- Suicide
- Eating disorders
- Withdrawals

If you want to add some of your own the Custom Trigger Warnings [section](#).

To make use of the trigger warnings there is a little setting up you need to do on the blog posts you want tagged with them. Basically you just need the first line to say "*Trigger Warning:* " followed by which ever triggers (separated by commas) you want to tag it with. See below for an example.

Hello again

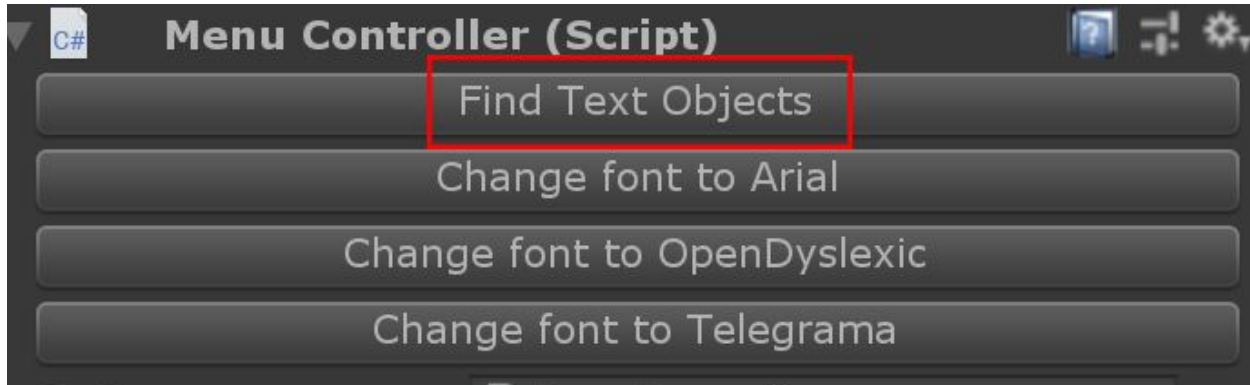
Trigger Warning: Self Harm

We're just testing the waters here a bit.

Yay!

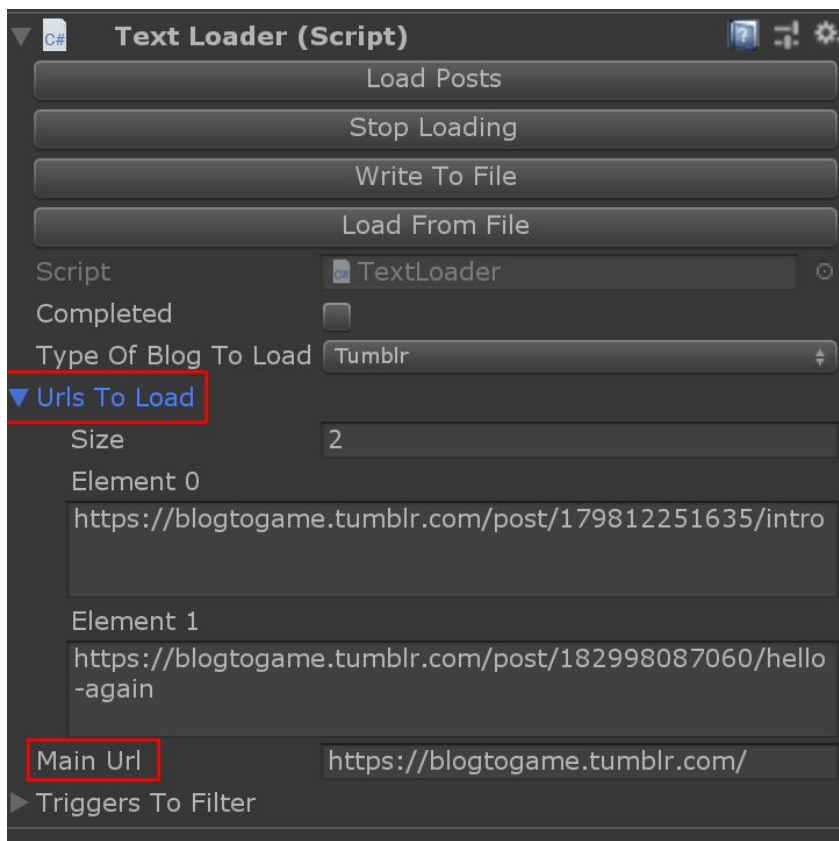
It also comes with the ability to change fonts and comes setup with two of them - Arial and Open Dyslexic. If you'd like to add more, please see the section on adding [custom fonts](#). If you happen to be playing with the UI and add more text elements or remove some there is an additional step you'll need to take so the game knows what elements to update.

On the **GameUI** prefab locate the button called “*Find Text Objects*” hit it, then hit apply. It should be all ready to go now!



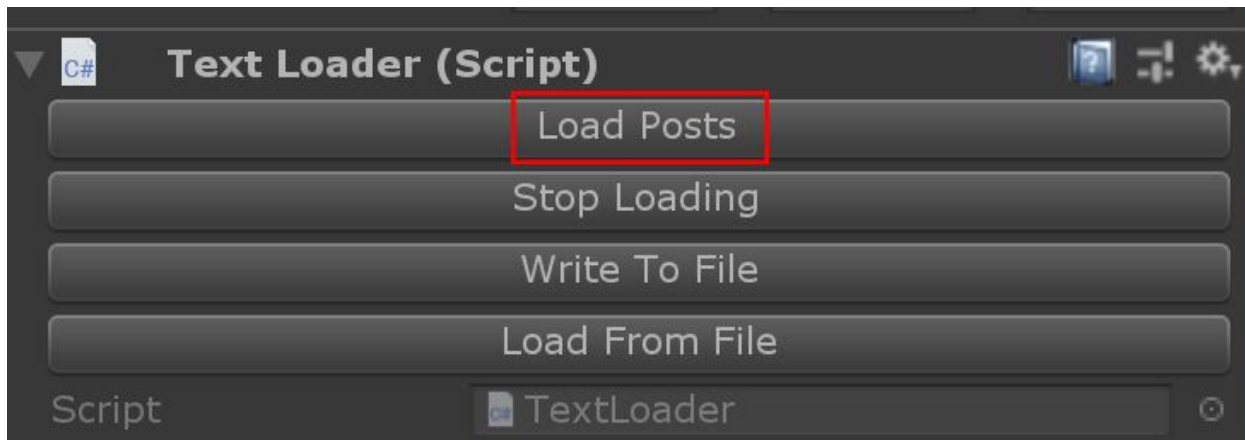
Alright, time to get your content into this game!

Head on over to the **TextLoader** prefab. There are two fields we want to use in it. “*Urls To Load*” and “*Main Url*”. You can see them highlighted in the picture below.

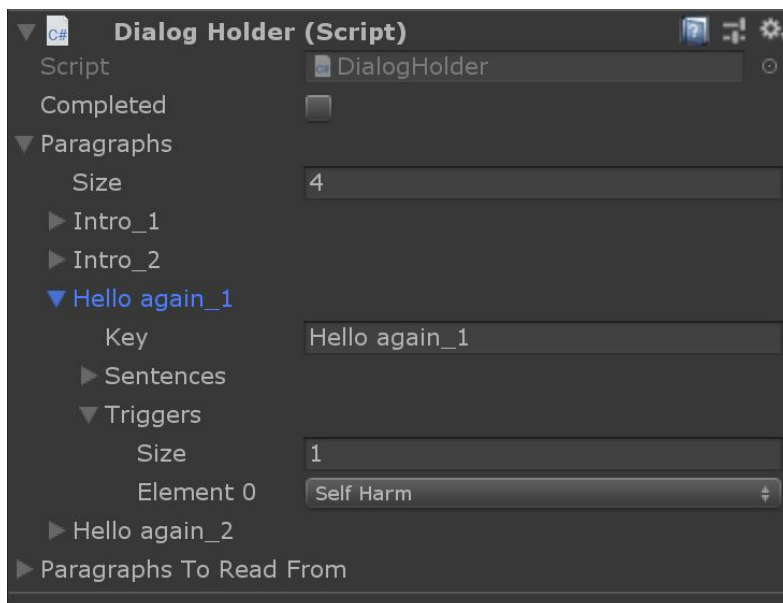


Put the home url of your blog into the “*Main Url*” section and then fill out the list “*Urls To Load*” with the individual blog posts you want included.

Then all you need to do is the the “*Load Posts*” button!



Head over to the **DialogHolder** prefab to check if your posts have been loaded correctly. If you're happy with it, be sure to hit apply! It should look something like the picture below, but with whatever content your blog had.



The game itself is all setup and ready to go now, so all you have to do is hit play and watch your blog come to life!

There is totally more customising you can do though with the [UI](#), the [models](#) in game, adding your own [sounds/music](#) (I had to remove my sounds for copyright reasons).

Custom Trigger Warnings

So you want to add some more trigger warnings, or just want to be able to filter more stuff out of your game? Brilliant! Let me show you how to do just that!

Let's pretend we want to add a new trigger warning for knives.

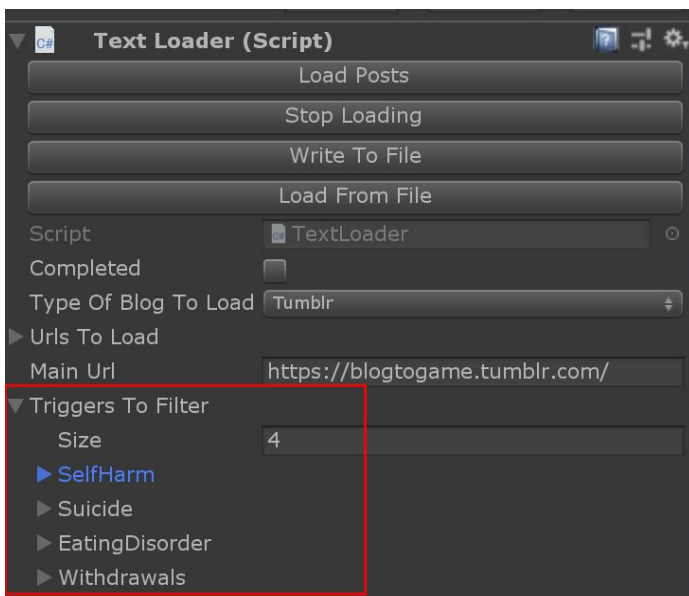
First we'll need to add our new trigger warning to the enum in the **TriggerEnumHolder** script. You'll notice the code looks like this:

```
public enum Triggers
{
    SelfHarm = 5,
    Suicide = 10,
    EatingDisorder = 15,
    Withdrawals = 20
}
```

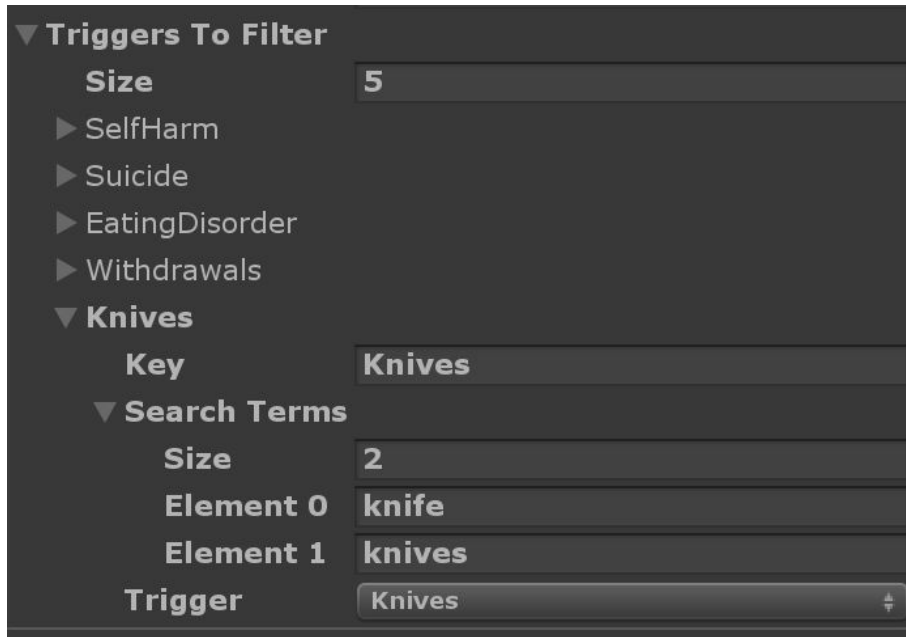
Let's add the new trigger warning to the end of that enum, so it now looks like this:

```
public enum Triggers
{
    SelfHarm = 5,
    Suicide = 10,
    EatingDisorder = 15,
    Withdrawals = 20,
    Knives = 25
}
```

Next go to the **TextLoader** prefab and locate the *"Triggers To Filter"* list.

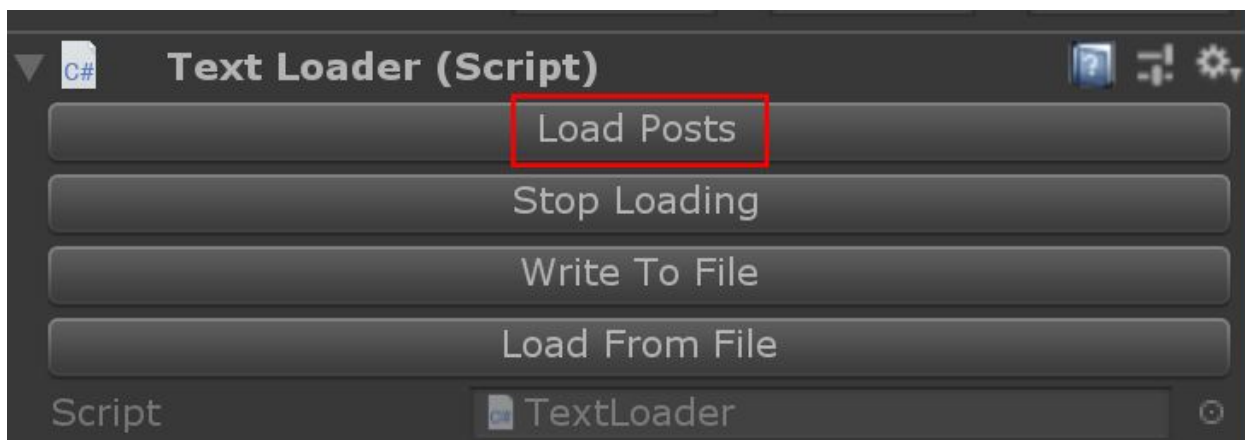


Time to add in the search parameters for the new trigger warning. We want to add all the words that we want the code to pickup and tag as that trigger/filter. So in this case we'll add knife and knives.



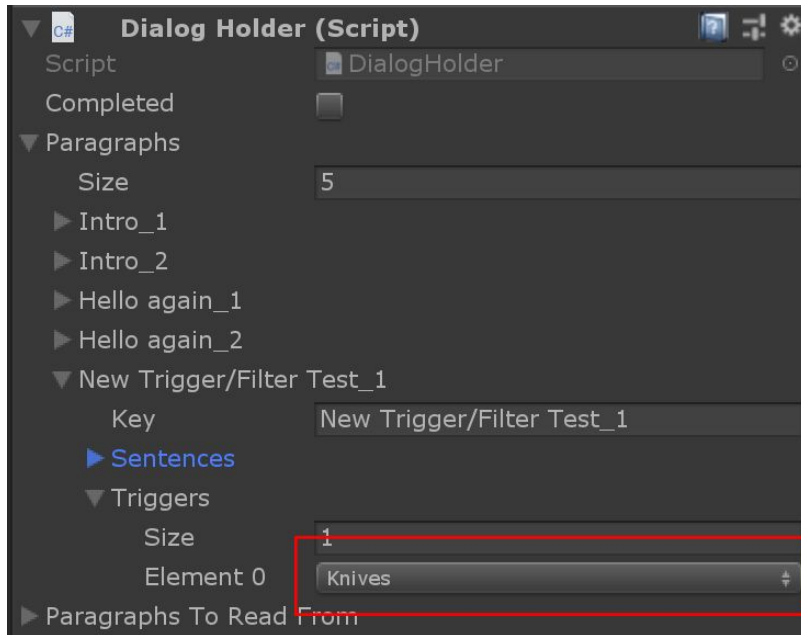
The screenshot shows a configuration panel titled "Triggers To Filter". It has a "Size" input field set to "5". Below this are four expandable sections: "SelfHarm", "Suicide", "EatingDisorder", and "Withdrawals". The "Knives" section is expanded, showing a "Key" input field set to "Knives". Below the "Knives" section is another expandable section titled "Search Terms". This section has a "Size" input field set to "2", and two "Element" input fields: "Element 0" set to "knife" and "Element 1" set to "knives". At the bottom of the "Search Terms" section is a "Trigger" input field set to "Knives".

Now that we've defined what triggers to look for we need to load in all our blog data again so it can pickup the new triggers. So go ahead and hit the "Load Posts" button on the **TextLoader**.



The screenshot shows a panel titled "Text Loader (Script)". It has a "C#" icon on the left and a "?", "≡", and "⚙️" icons on the right. Below the title bar are four buttons: "Load Posts", "Stop Loading", "Write To File", and "Load From File". The "Load Posts" button is highlighted with a red rectangle. At the bottom of the panel is a "Script" input field set to "TextLoader".

Let's head on over to the **DialogHolder** prefab to see if our data has loaded in correctly. As we can see below the new trigger warning has been successfully added.



Let's hit apply on the **TextLoader** and **DialogHolder** prefabs and move on to assigning some toggles to actually filter them out in game!

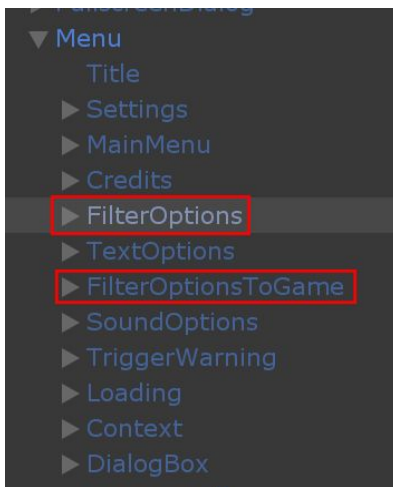
Firstly, we'll need to write a bit more code for this part. Locate the **MenuController** script and you should be able to find a section labelled "*Trigger Settings*". Let's take a look at the function we use for setting the withdrawals trigger.

```
public void setWithdrawalsTrigger(bool filterOut)
{
    SettingsController.instance.setTriggerSettings("Withdrawals", filterOut);
    if(GameController.instance.gameHasLoaded)
        SettingsController.instance.setTriggerFilters();
    loadSettings();
    SaveController.instance.saveSettings();
    RelayCentre.postMessage(Message.TriggerFilterUpdated);
}
```

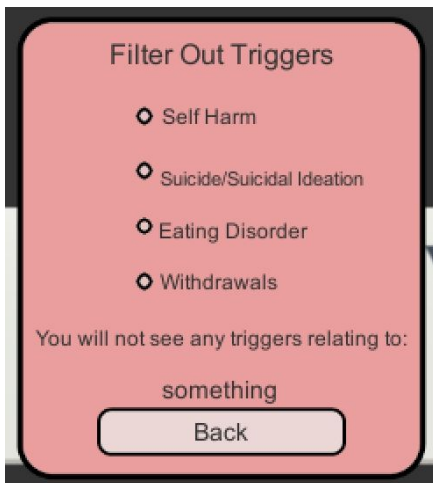

So let's write a new function for knives. It should look something like the code below, with "Knives" being whatever you've called your new trigger.

```
public void setKnivesTrigger(bool filterOut)
{
    SettingsController.instance.setTriggerSettings("Knives", filterOut);
    if(GameController.instance.gameHasLoaded)
        SettingsController.instance.setTriggerFilters();
    loadSettings();
    SaveController.instance.saveSettings();
    RelayCentre.postMessage(Message.TriggerFilterUpdated);
}
```

Now we want to add the physical toggle to the UI. There are two screens these toggles appear on, both under the **GameUI** prefab. They are "*FilterOptions*" and "*FilterOptionsToGame*".



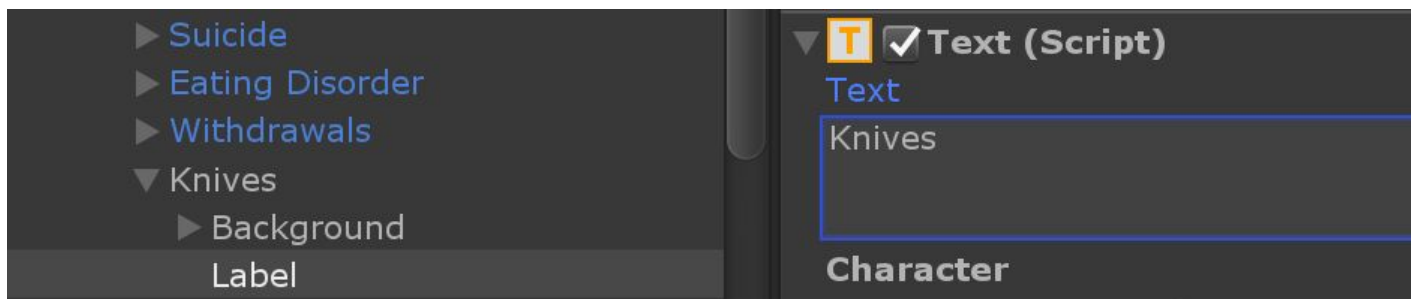
Turn one of them on and the picture below should appear in game.



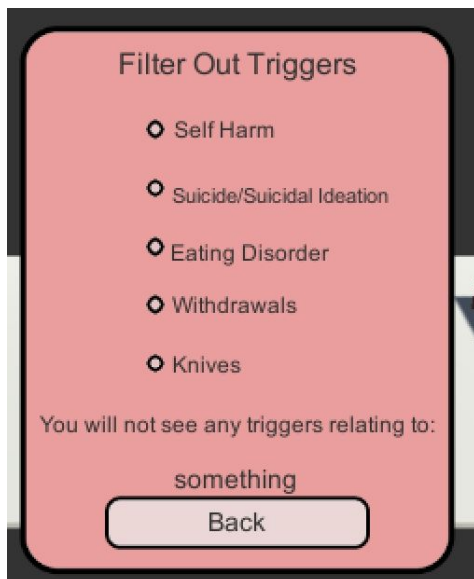
Now duplicate one of the trigger warning toggles and name it according to your new trigger. Be sure of the order it is in so that it appears in the correct place in the UI.



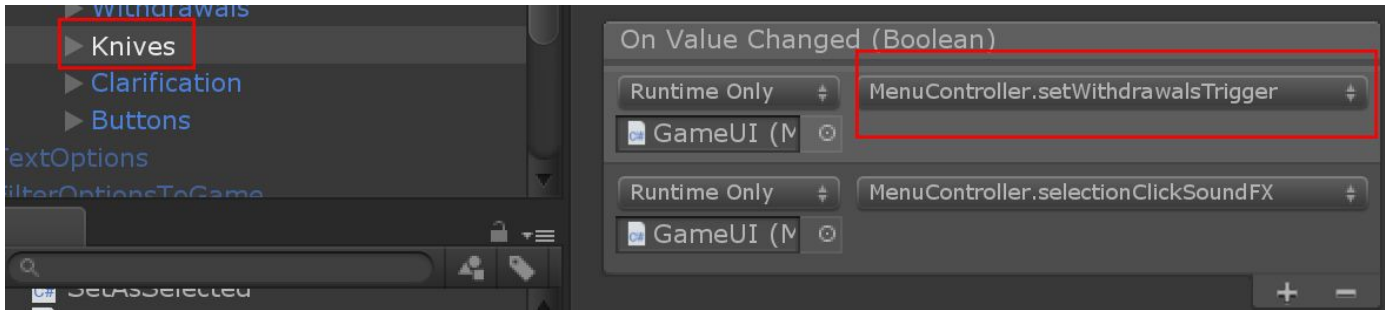
Now change the label under the toggle to reflect your new trigger warning as well.



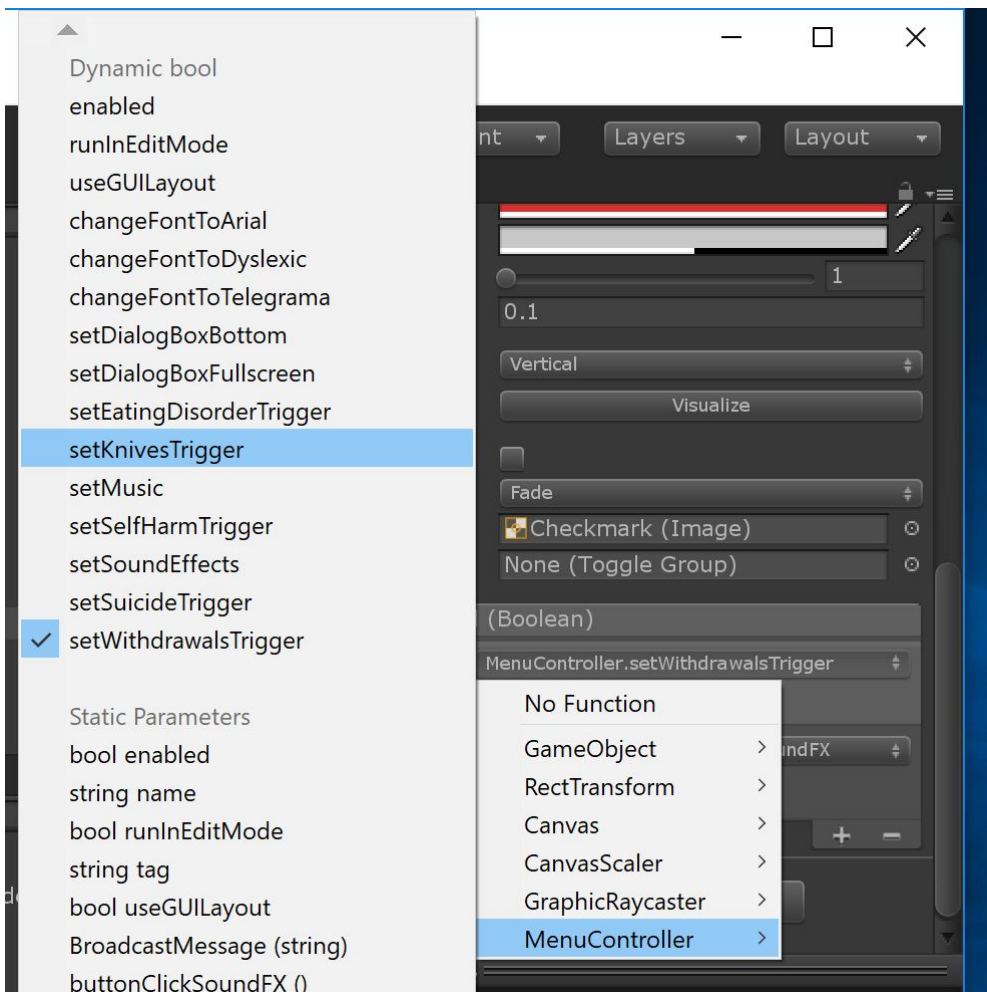
It should now look like the image below:



Finally for this toggle we want to hook up the function that will get called when we click on it. Locate the toggle itself and the section where we can pick the function to call.

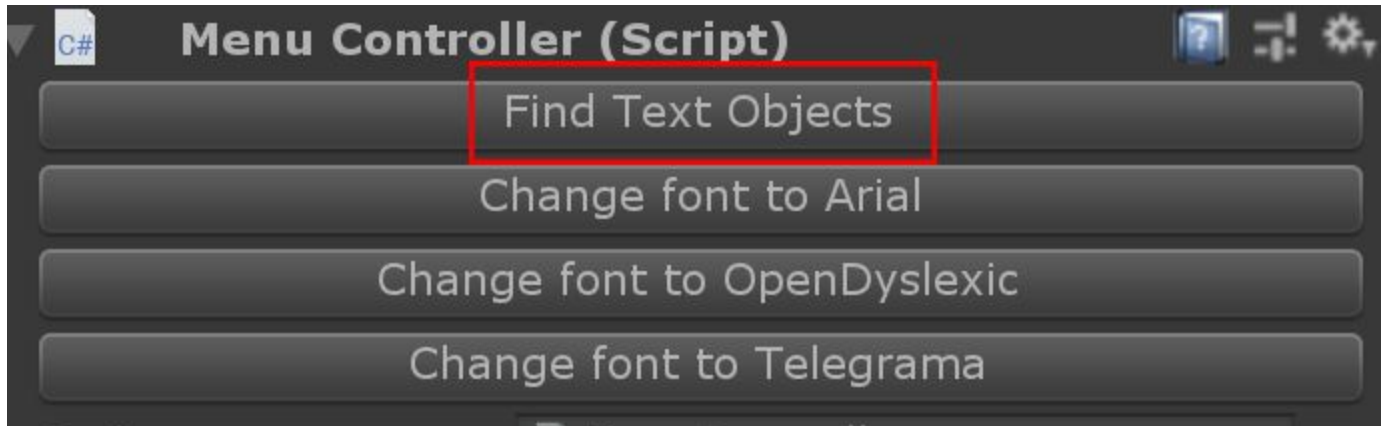


Then pick the function we just wrote, making sure we pick it from the “*Dynamic bool*” section.

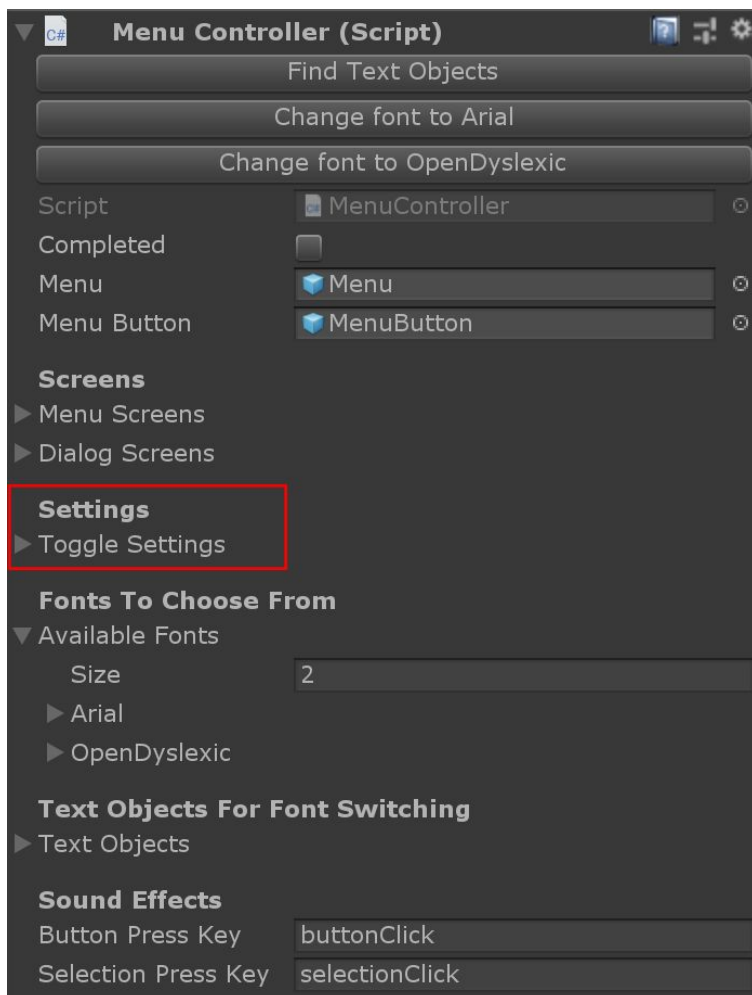


That's it for this toggle, hit apply and then follow the same steps for the next screen to add the new toggle to it.

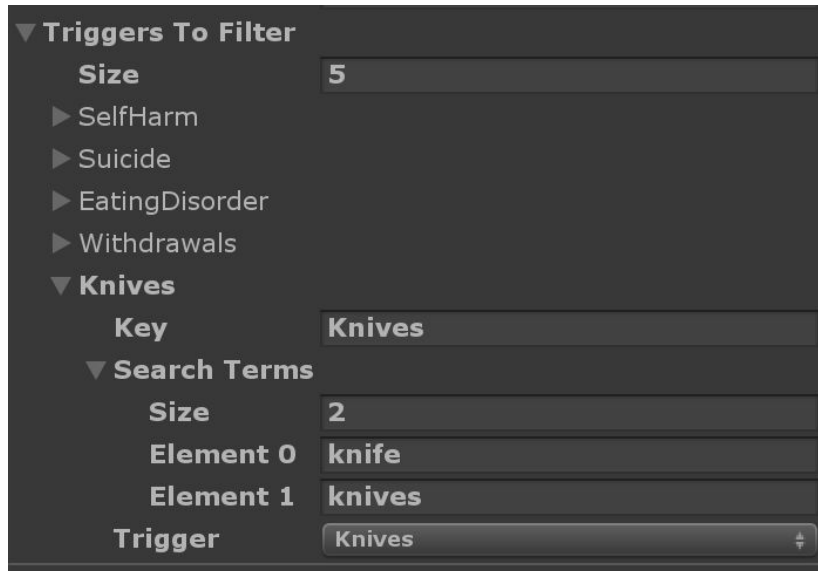
Before we go any further, we need to update our list of text objects so that the game can still accurately change fonts. Since we've added two new toggles, that list is a bit out of date. So locate the *"Find Text Objects"* button on the **GameUI** prefab and go ahead and hit it.



Alright, now let's add some toggle settings so that it can keep track of which toggles have been activated between game loads. Locate the *"Toggle Settings"* under the **GameUI** prefab.



Add in a new toggle setting and call it “*Knives*” or whatever you’ve called you new trigger warning. Make sure you hook up both toggles from the two screens into it like below.



The image shows a settings menu with a dark background and light text. It is organized into sections with expandable/collapsible headers. The first section is 'Triggers To Filter', which is expanded. Inside this section, there is a 'Size' field set to '5'. Below that are four items with right-pointing triangle icons: 'SelfHarm', 'Suicide', 'EatingDisorder', and 'Withdrawals'. The second section is 'Knives', which is also expanded. It contains a 'Key' field set to 'Knives'. Below that is a 'Search Terms' section, which is expanded. Inside 'Search Terms', there are three fields: 'Size' set to '2', 'Element 0' set to 'knife', and 'Element 1' set to 'knives'. At the bottom of the 'Knives' section is a 'Trigger' field set to 'Knives', which has a small icon on its right side.

▼ Triggers To Filter	
Size	5
▶ SelfHarm	
▶ Suicide	
▶ EatingDisorder	
▶ Withdrawals	
▼ Knives	
Key	Knives
▼ Search Terms	
Size	2
Element 0	knife
Element 1	knives
Trigger	Knives

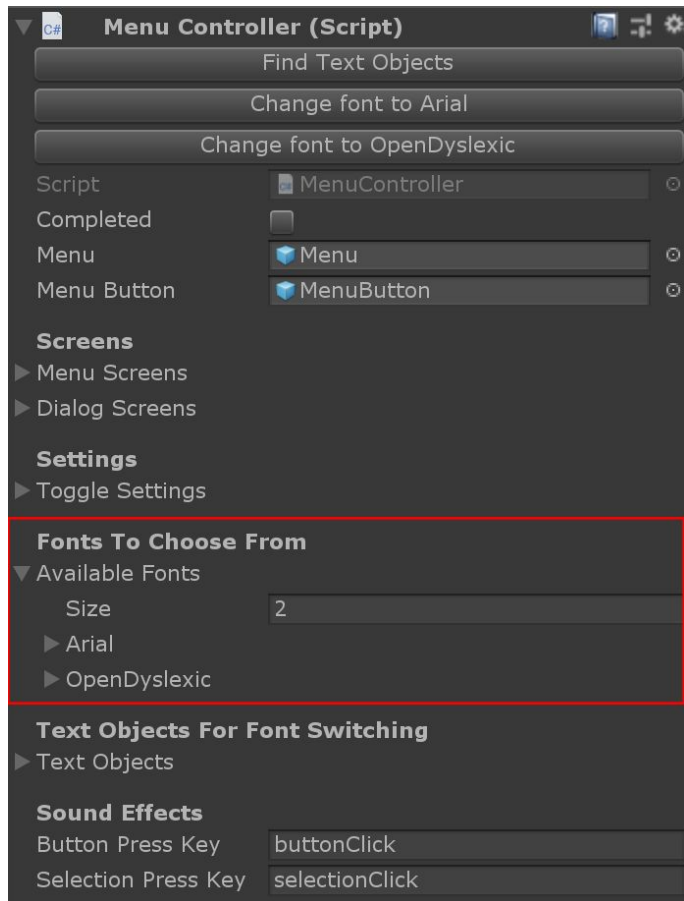
Finally hit apply on our **GameUI** prefab and we should be good to go!

Custom Fonts

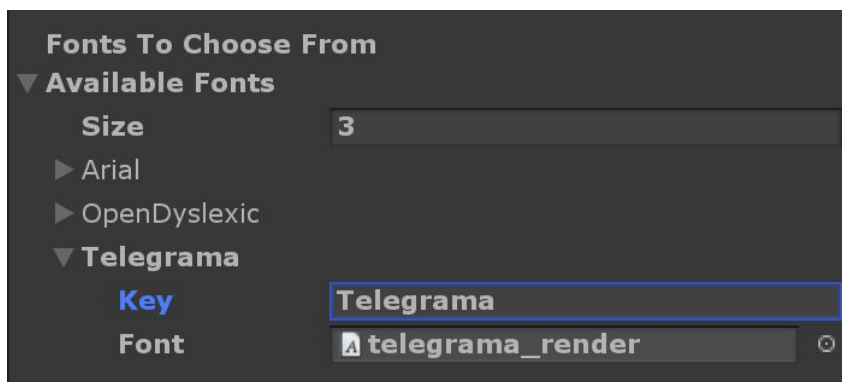
Want to add some of your own fonts? Well, you've come to the right section!

Let's pretend we want to add the new font Telegrama.

First of all we go to the **MenuController** script on the **GameUI** prefab. Then we want the section *"Fonts To Choose From"*



Then we go ahead and add in Telegrama as a new font and link up the font we want to use.



Next we need to write a bit of code so that we can change the fonts in game. Let's take a look at what the code looks like for changing to Open Dyslexic. This can be found in the **MenuController** script.

```
public void changeFontToDyslexic(bool selected)
{
    if(!selected) return;

    changeFont("OpenDyslexic");
    SaveController.instance.settings.setBool("Font_Arial", false);
    SaveController.instance.settings.setBool("Font_OpenDyslexic", true);
    RelayCentre.postMessage(Message.SaveGame);
}
```

So we need to write something fairly similar to that:

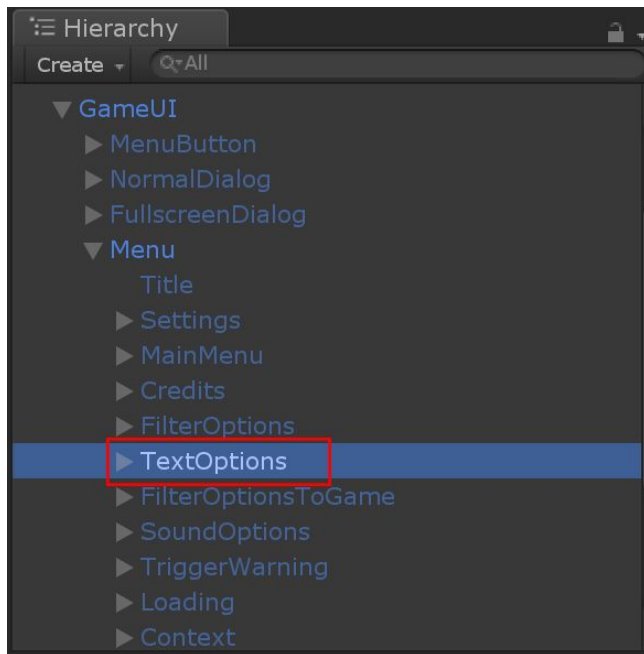
```
public void changeFontToTelegrama(bool selected)
{
    if(!selected) return;

    changeFont("Telegrama");
    SaveController.instance.settings.setBool("Font_Arial", false);
    SaveController.instance.settings.setBool("Font_OpenDyslexic", false);
    SaveController.instance.settings.setBool("Font_Telegrama", true);
    RelayCentre.postMessage(Message.SaveGame);
}
```

Then we need to just add this line below, to all the other functions used to change fonts.

```
SaveController.instance.settings.setBool("Font_Telegrama", false);
```

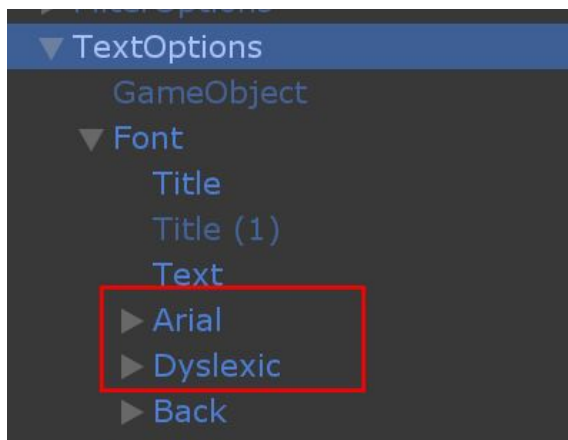
Now we want to add an actual toggle in the game UI so we can change the fonts! Locate the **TextOptions** object under the **GameUI** prefab.



Turn on the **TextOptions** object and you should see the below screen in the game view.

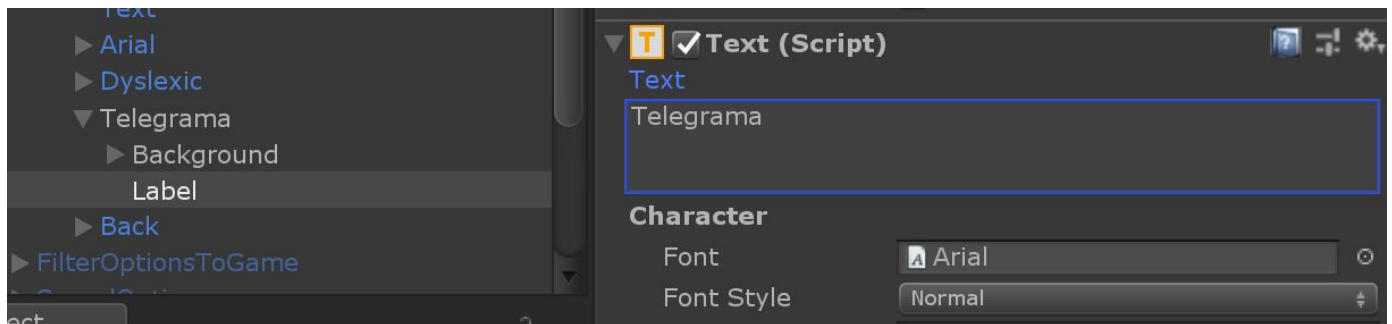


Take a look inside the **TextOptions** object and you'll find toggles for the two fonts we already have.



Now duplicate one of the and change its name to the font your adding. In this case we'll be naming it *"Telegrama"*. Be sure of the order it appears here, you'll want it above the back button so that it appears before it.

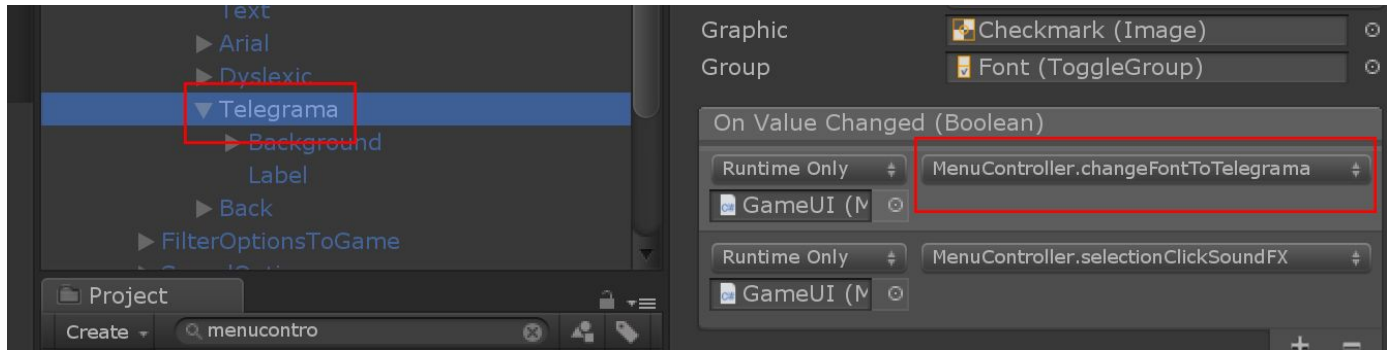
We also want to change the label on the toggle so that it says *"Telegrama"* in game.



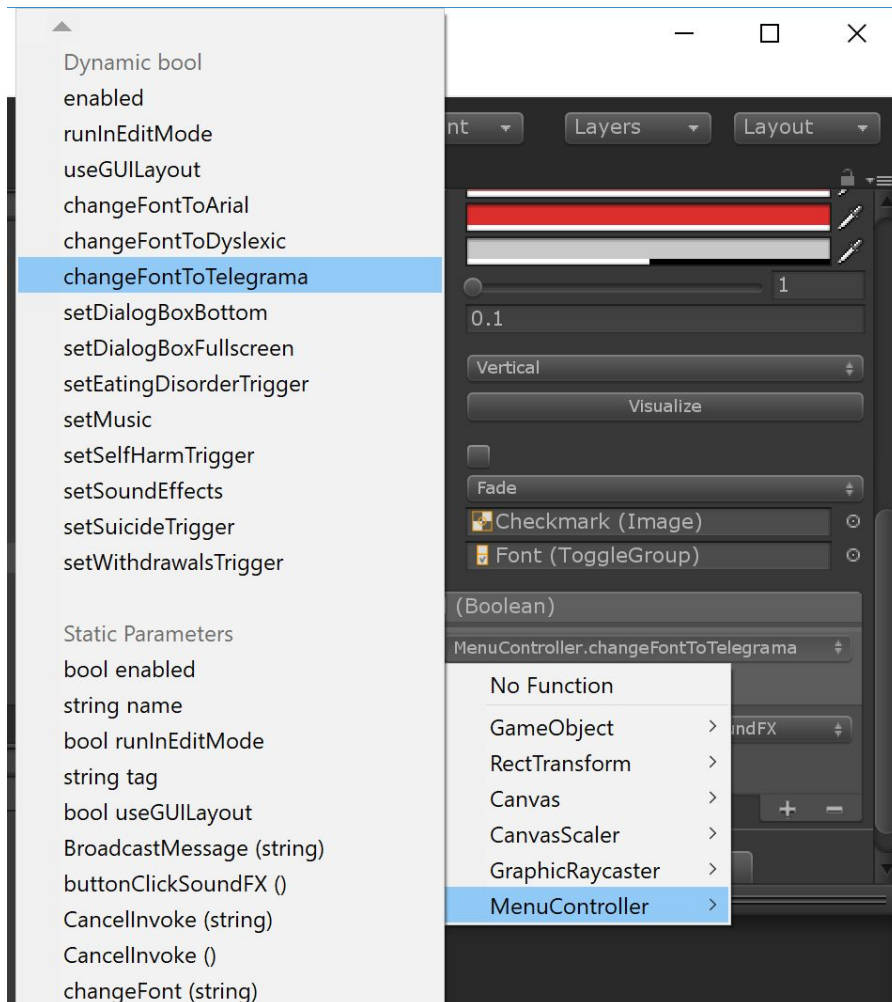
The UI should now be looking something like below:



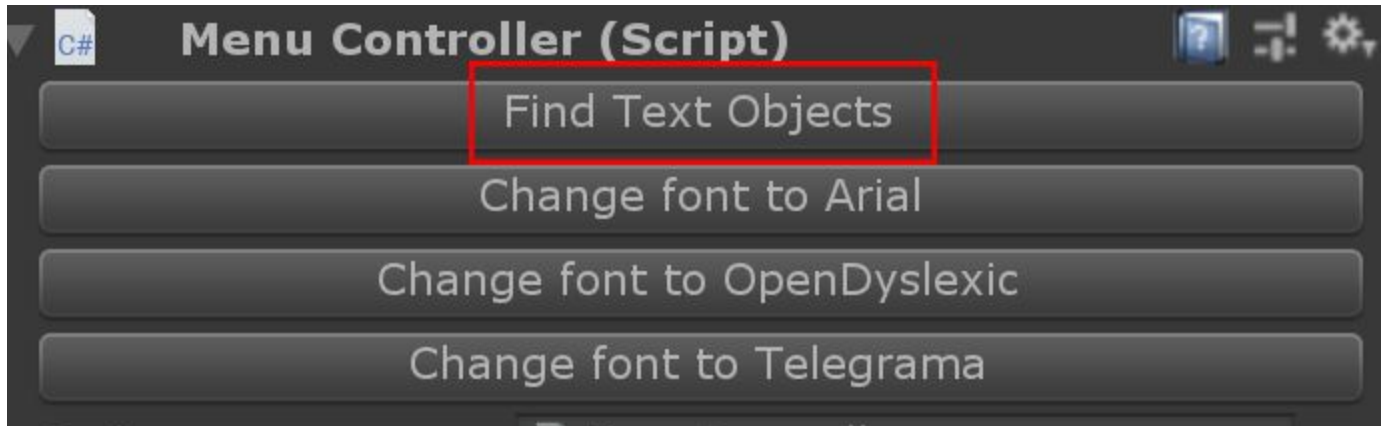
Lastly for this toggle setup is making sure we have the right function hooked up got when you click on it. Click on the Telegrama (or whatever you named your font) object and then find the “On Value Changed” section.



Click on the above highlighted area to select what action we want to occur when the toggle is clicked on. We want to call that function we wrote earlier! Make sure you are picking from the *Dynamic Bool* section, as these are the ones that will pass in the toggle value we want.



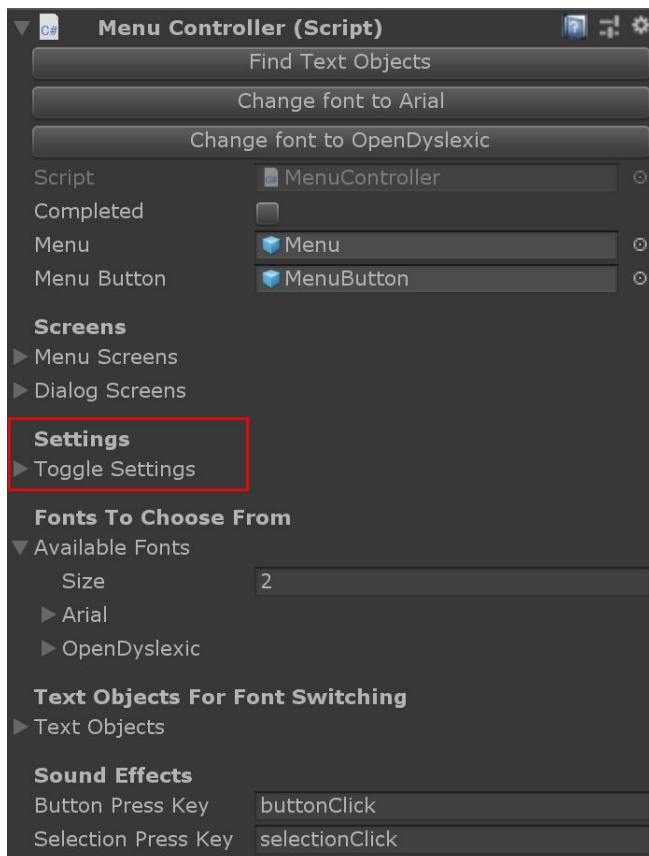
Now because we've added more text objects to the UI there is an extra step we need to take to make sure they will get updated when we change fonts. So locate the button labelled *"Find Text Object"* - which can be found on in the **MenuController** which is on the **GameUI** prefab.



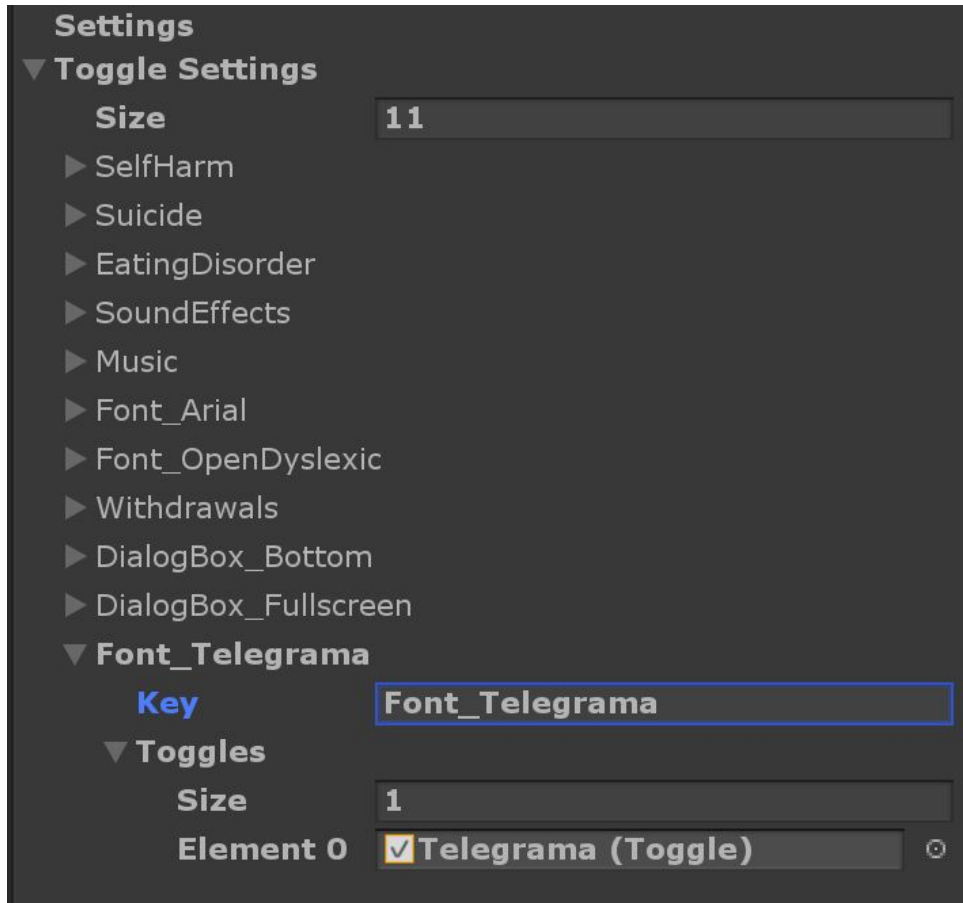
Go ahead and hit that button! You'll notice the *"Text Objects"* list should have grown in size. You can go ahead and hit apply on the **GameUI** prefab.

Now there is one last step we need to go through. While you can totally load up the game and watch the fonts change as you switch between the toggles, it won't yet save any settings for the new one. So we need to add that now.

Go ahead and locate the *"Toggle Settings"* under the **MenuController**.



We want to add a new toggle setting like below and call it “*Font_Telegrama*”. Then hookup that toggle we created earlier.



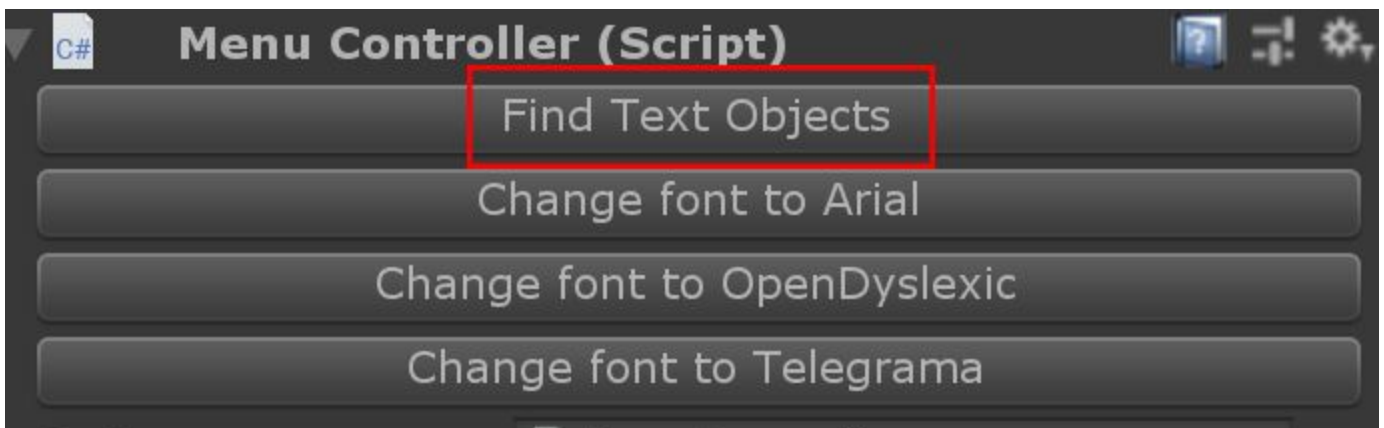
Lastly hit apply on the **GameUI** prefab and we're done!

Updating the UI

I'm not going to go in depth on the ways you could tailor the UI to suit your needs. You can swap out sprites and add new menus etc. I'd recommend checking out Unity's documentation on UI - there is lots you can do!

One thing I will go into is what you need to do if you add more text to your UI, or remove some. Because this game has a system that allows the user to change fonts as they see fit, I need to keep track of all the bits and pieces that need replacing.

Good news is it's super easy to do! Locate the button labelled "*Find Text Objects*" - which can be found in the **MenuController** which is on the **GameUI** prefab.



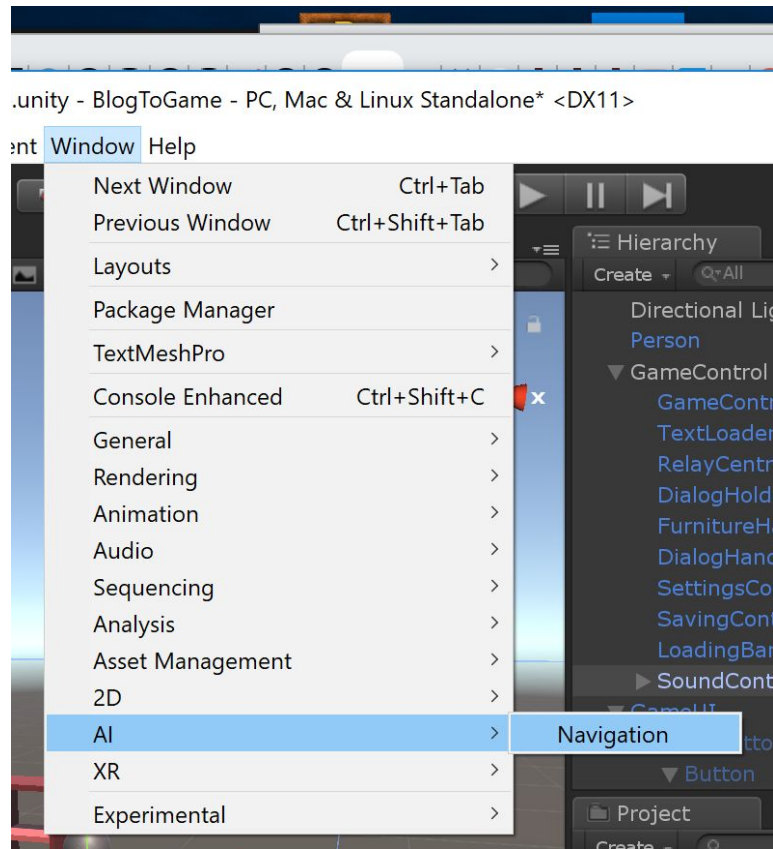
Hit that button, then you can hit apply to the **GameUI** prefab and you're done!

It's good to get in the habit of doing a quick scan of the "*Text Objects*" list to see if it's lost any references to text objects if you've been playing with the UI at all. Which can again be fixed by hitting that handy button above.

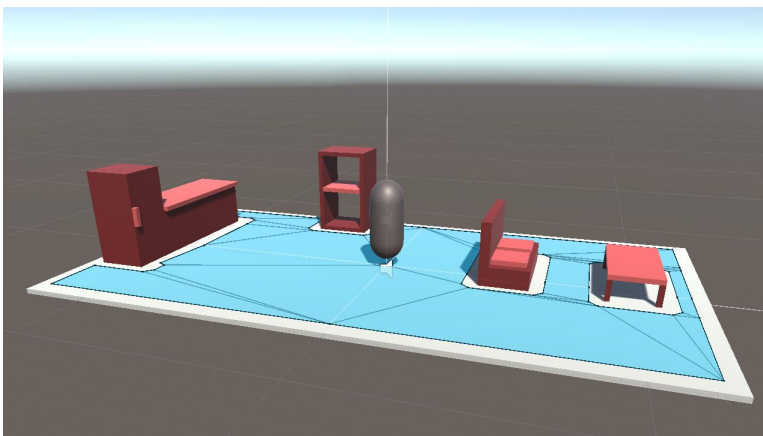
Updating Art Assets

While the this game builder totally comes with art assets already setup, you are more than welcome to setup your game world however you like! It's super easy to quickly rearrange the room or swap in some of your own art. It's also not overly difficult to really overhaul the world, provided you're still thinking 3d. I am sure 2d is possible too but I won't be covering how to go about that.

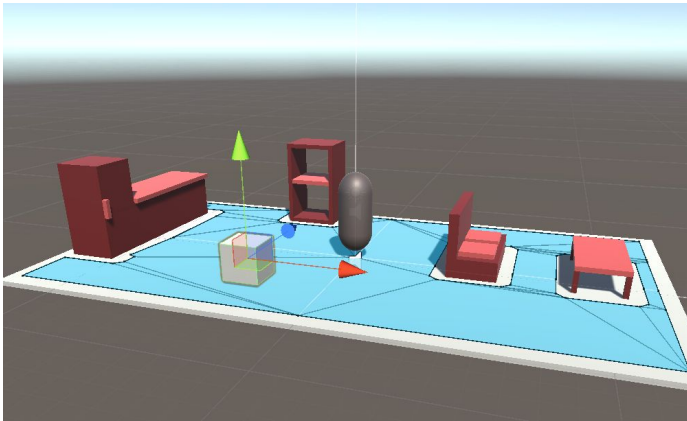
So for this I have been using Unity's Nav Mesh system, which is a great way to get quick navigation going! So you can find it under Window/AI/Navigation.



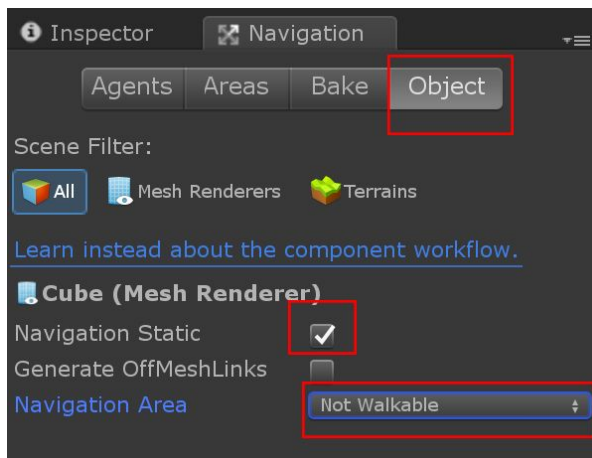
This is what our game area should look like now with the Navigation tab open:



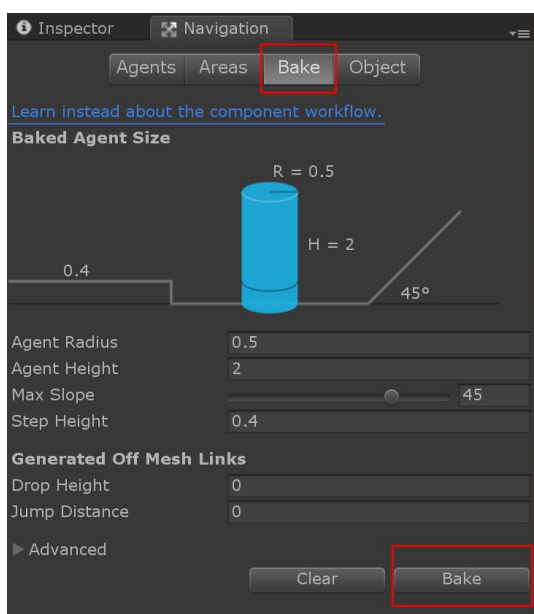
Let's pretend I want to add a cube! Totally new piece of furniture in the world. Now it should look like this:



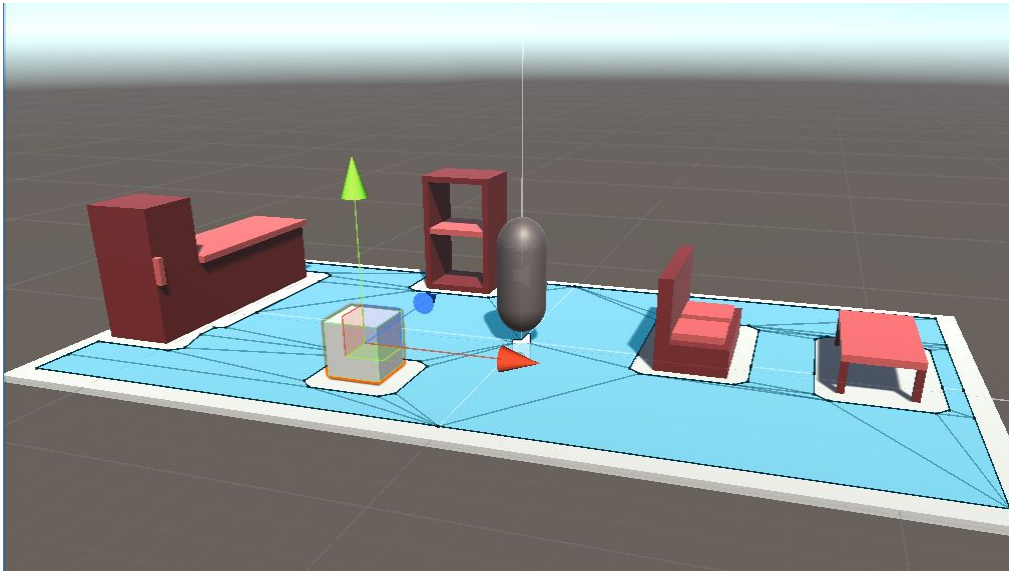
Now we need to tell the navigation system that there is something there. So under the navigation tab go to the object tab. Here we want to tick yes to Navigation Static and tag the area as Not Walkable.



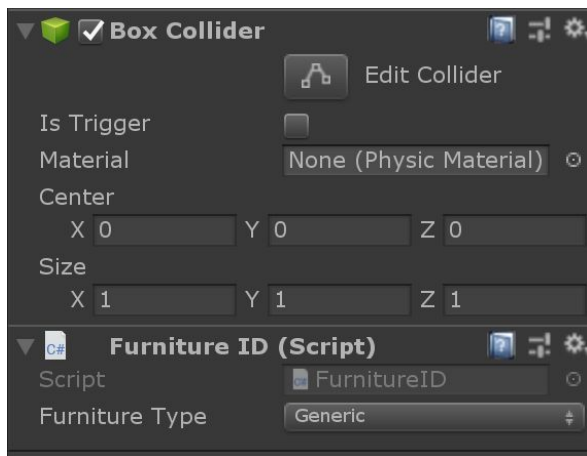
Now head back to the Bake tab and hit the bake button.



The world should now look more like this!



The last thing you need to do is ensure your new piece of furniture, or in this case cube, has the “*FurnitureId*” script on it, so that it’s interactable in the game. If you want it to be. It will also need a collider on it.



Adding in Sound Effects

The game comes with some sound hooks already in place, just no sounds in the library. This was due to me using sounds that I purchased for my use, so I can't provide them for public use unfortunately. But I can show you how you can quickly get your own sounds working in game!

We'll be working in the **SoundController** prefab, which can be found under the **GameControl** object. In there you'll notice a list called "*Sound Library*" - this is where you'll store your sounds!

This whole system is built on keys, so you can call a sound anywhere by using the **SoundController** and calling the function "*PlayAudioEntry*" and then inputting the key of the sound effect you want! The other cool thing is you can put multiple clips in an entry and it will pick one of them to play at random, so you can get a bit of variety happening. Helps the sounds to not sound too repeated like in the case of the typewriter sounds I was using for the text appearing on screen.

Let's take a look at the code that runs the sounds that play while the text appears on screen.

```
//we need it to always play on the first word and last, then a chance of playing
if(i == 1)
{
    SoundController.instance.playAudioEntry("TypeWriterClick");
} else if (i == text.Length - 1)
{
    SoundController.instance.playAudioEntry("TypeWriterClick");
} else if(value < 0.6)
{
    SoundController.instance.playAudioEntry("TypeWriterClick");
} else if(value < 0.7)
{
    SoundController.instance.playAudioEntry("TypeWriterClickFlavour");
}
```

Here we make sure that there is always a sound on the first and last character typed, then a 60% chance of a normal click and then a 10% chance of a flavour sound. That leaves a 30% chance that no sound will be played, this way we're not creating an overbearing sound scape. For my game my flavour sound was a carriage return which worked well. It's up to you how you'd like it to sound and please feel free to customise the code as you see fit!

The "*buttonClick*" and "*selectionClick*" entries are for the UI. They'll fire off when hitting a button or selecting a toggle.

The "*footsteps*" entry is for the characters footsteps as they move about the room. I'd advise putting in a few difference sounding singular steps to get the best effect.

Feel free to add anymore you might want or need!

The current sound library entries:

