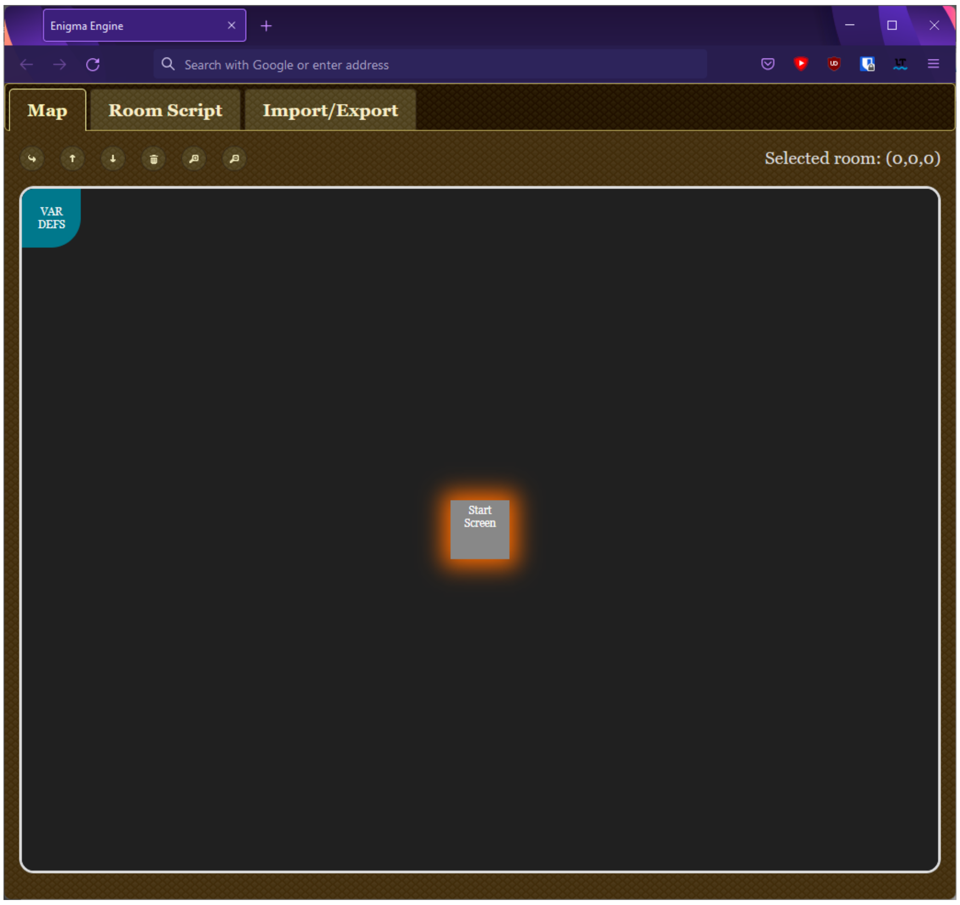
ENIGMA ENGINE MANUAL

DOWNLOADING THE ENGINE

Welcome! First of all, big thanks for checking out my engine! To get started, simply head over to my engine's GitHub page, then click the green "Code" button, and "Download ZIP". Then, extract the zip file wherever you like, and you're done! The engine is written in HTML and runs fully in the browser, so no installation is required. Now, you can simply open up the folder you just extracted, then go to the "Editor" folder, and open "Enigma Engine.html" in your favorite browser to start making games.

OVERVIEW

When you launch the engine, you are greeted with this:



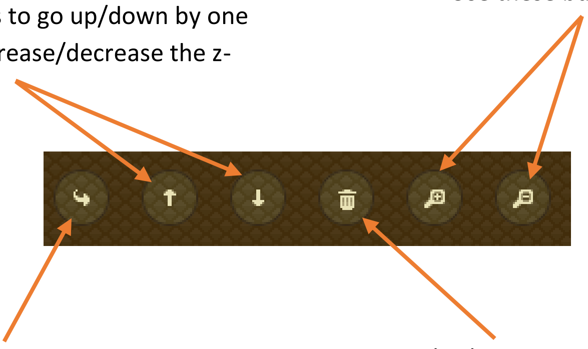
As you can see, we have three different tabs. Let's go over them all.

MAP

Here, you can see the map view of your game. Click on a room, then head over to the Room Script tab to edit it. At the top, we have several buttons to help you navigate the map, and to the right you can see the (x,y,z) coordinates of the currently selected room. A blank game always contains one pre-made room – the Start Screen room – where the game starts, the coordinates of which are (0,0,0). You cannot delete this room. You also always have a "Variables" room – that's the blue one in the top-left corner. All code in this room is executed at the start of the game, and can be used to define the initial values for your game variables for instance, like the value of the player's health at the start of the game, hence the name. Now, let's go over all the buttons at the top.

Use these buttons to go up/down by one map layer (i.e. increase/decrease the z-coordinate by 1)

Use these buttons to zoom in/zoom out

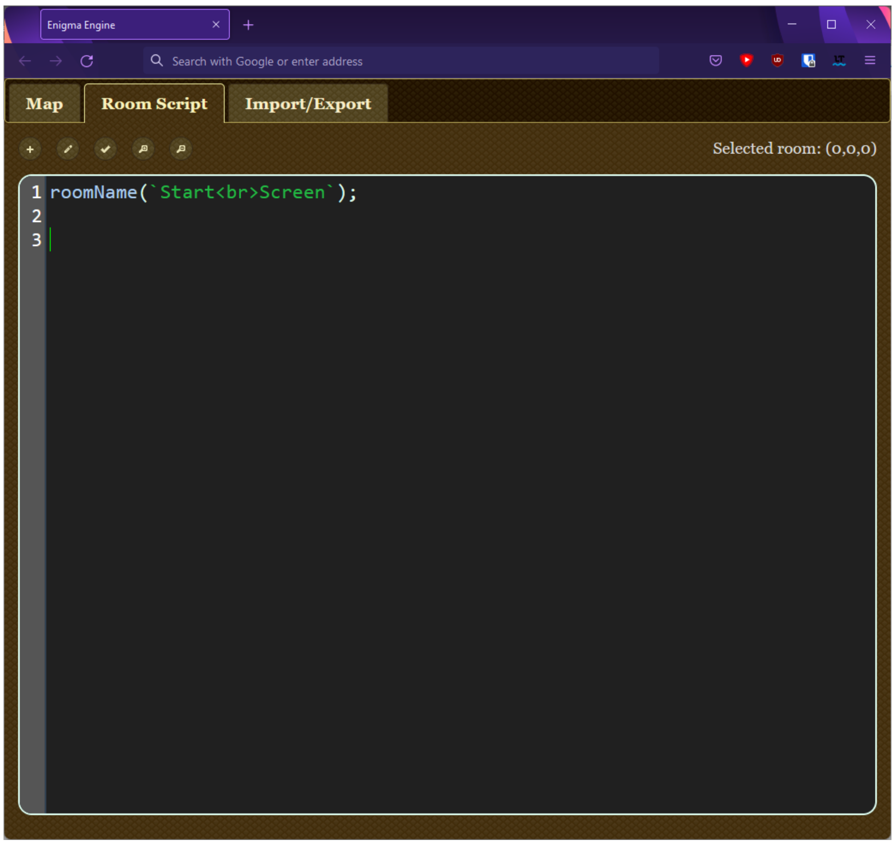


Click on this button to navigate to a specific set of (x,y,z) coordinates

Use this button to delete the currently selected room

The main question I usually get is "how do I create a new room?" The answer is: you do not. New rooms are created automatically as soon as they are referenced in the Room Script. So, let's head over and explore the Room Script tab!

ROOM SCRIPT

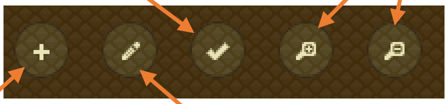


```
roomName(`Start<br>Screen`);


```

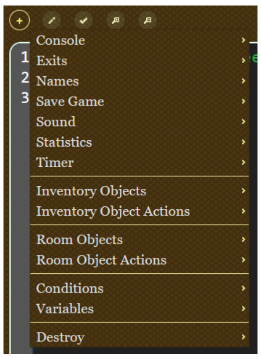
As you can see, the Start Screen room already has one line of code added to it, which sets the room's name to "Start Screen". First of all, let's go over the buttons at the top, like we did with the Map tab.

Use this button to check the syntax of the code below. If you use the code templates, you don't really need this, as the code is written *for* you

Use these buttons to increase/decrease the size of text in the text editor below

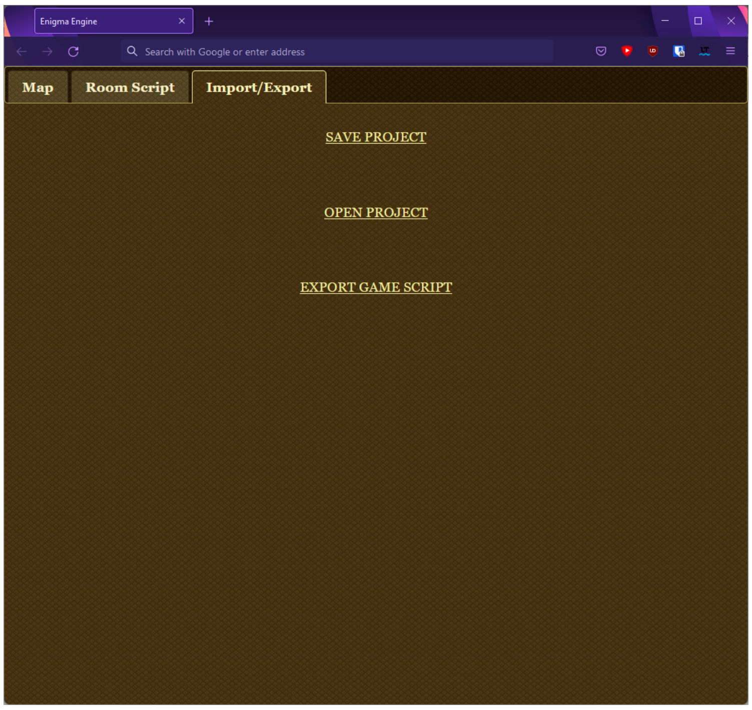


This is the most important button, and the one you will be using most often. This is the "Add" button, where all the code templates are stored:



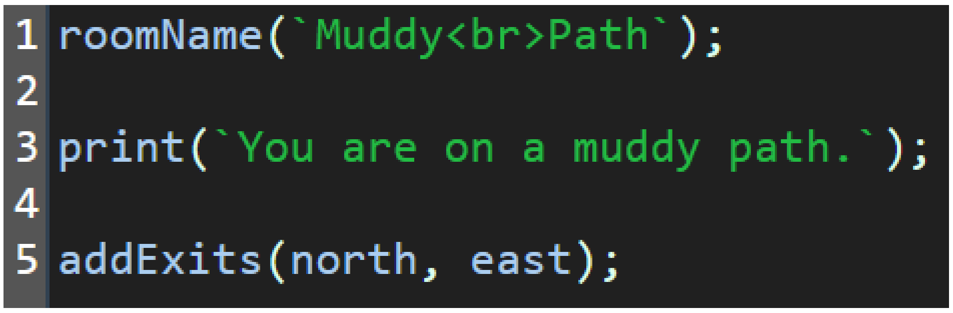
This is the "Edit" button. When you go to Add -> Console -> Print Text, you will be guided through creating a line of code that will display some text to the player. This is one of the functions you will be using most often, as we are making a text game! Now, the Edit button lets you edit that line of code after it's been created, in case you change your mind about what text you want to display to the player!

IMPORT/EXPORT



This tab is quite self-explanatory; here, you can save your project to disk, open a project you have already started, or "Export Game Script", which you use to export the actual game in a playable format – more on that later. This concludes our overview, so let's move onto some examples!

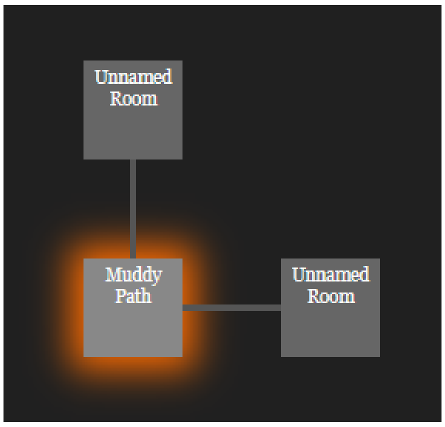
EXAMPLE – ROOM SCRIPT



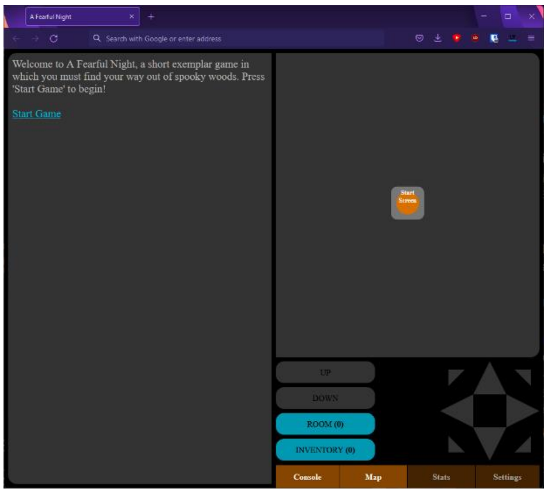
```
roomName(`Muddy<br>Path`);

print(`You are on a muddy path.`);

addExits(north, east);
```

This is what the most basic room in your game will look like – the Room Name function sets the name of the room, which is visible on the map in the Editor as well as to the player. The Print function displays some text to the player. And the Add Exits function, in this case, allows the player to either move North or East from here. These three functions can be accessed by pressing on the Add button, and then Names -> Room Name (to create the Room Name function), Console -> Print Text (to create the Print function), and Exits -> Add Exits (to add the exits).

If you now head over to the Map tab, you will notice that new rooms to the North and to the East of the currently selected room have been automatically created *for* you!



A LOOK AT AN EXPORTED GAME



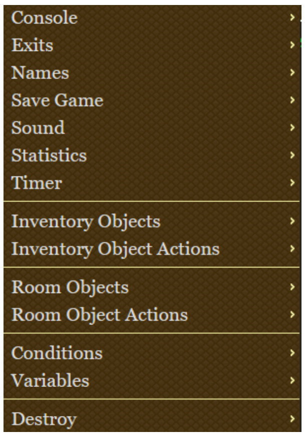
Welcome to A Fearful Night, a short exemplar game in which you must find your way out of spooky woods. Press 'Start Game' to begin!

Start Game

Before we go over all the currently-available code templates and how to use them and what they do, let's see what a game created with this engine looks like, so you fully understand what is happening. Go to the "Example Game" folder, then "Exported Game", and launch "A Fearful Night.html". The game will look something like the picture to the left.

Play around with it, and I have included the project file as well so you can open it with the Editor and have a look at the source code. Of course, to understand the source code, you will need to know all the functions available in this engine, so let's move onto creating actual code!

CODE TEMPLATES



First of all, an important note about how the engine works: the game is never aware of what is happening in other rooms. When the player is in a room, the script attached to this room is executed, and that's all the game is aware of. So, for example, if there is a button in Room A, and you want something in Room B to be affected when you press it, you must store this data in a variable (e.g. create a variable "pl.buttonPressed" and set its initial value to "false" (without the quotes), then set it to "true" (also without the quotes) when the player presses the button; then, make different things happen when the player enters Room B depending on the value of this variable).

Another important thing to note is that all variable names must be preceded by "pl." For example, if you want to store the health of the player, you can make a variable called "pl.hp". Or, if you want to store the player's name, "pl.name".

Now – when you go to the Room Script tab and press the Add button, you will be presented with a list of various things you can add to the room script, as seen in the picture to the left. Let's go over them all.

CONSOLE -> PRINT TEXT

Allows you to display text to the player. It is recommended to display some text after every player interaction, to keep the player informed of what's happening. Once this code is generated, its content can be edited later on by clicking on it, and then – now with the cursor on the line that contains (only) the Print function – clicking the Edit button.

The code this template will generate for you looks like this:

```
print(`YOUR TEXT`);
```

CONSOLE -> PRINT HYPERLINK

This allows you to print a clickable link to the console. You can assign any actions you like to this link. Links are disabled whenever the player moves to a different room (i.e. the player can no longer click on them). HOWEVER – if you have complex interactions in a single room with many different links, you must remember to disable a link yourself once the player clicks on it, unless you *want* the player to be able to click on it more than once, or go back and change his mind. The function to disable all currently printed links can be found in Destroy -> Destroy All Hyperlinks

The code this template will generate for you looks like this:

```
link(`TITLE OF HYPERLINK`, function() {
        |HERE GO ALL THE ACTIONS YOU WANT TO HAPPEN WHEN THE PLAYER CLICKS ON THE LINK
});
```

CONSOLE -> PRINT IMAGE

This allows you to print an image to the console. Make sure to place the image in the same folder as the exported game, otherwise nothing will be displayed!

The code this template will generate for you looks like this:

```
image(`FILENAME`);
```

CONSOLE -> CLEAR CONSOLE

This allows you to clear the console of all content. You can use this to tidy things up for the player – e.g. when the player moves onto a new chapter, you may wish to clear the console so the player starts fresh. Note that, if there are currently any hyperlinks in the console that the player needs to click on to progress in the game, they will be deleted as well!

The code this template will generate for you looks like this:

```
clearConsole();
```

EXITS -> ADD EXITS

This allows you to add exits, i.e. tell the game which directions the player can move in. Available directions are north, south, east, west, northeast, northwest, southeast, southwest, up, and down.

The code this template will generate for you looks like this:

```
addExits(LIST OF EXITS);
```

EXITS -> REMOVE EXITS

This allows you to remove exits, i.e. if you have previously allowed the player to move into a certain direction, but they've interacted with something in the room and now you want them to no longer be able to do so, you can use this function to disable that exit.

The code this template will generate for you looks like this:

```
removeExits(LIST OF EXITS);
```

EXITS -> TELEPORT PLAYER

This allows you to instantly teleport the player to a different room. Note that this should always be the very last action executed in a room!

The code this template will generate for you looks like this:

```
teleport(`COORDINATES`);
```

NAMES -> GAME TITLE

This allows you to change the title of the game, i.e. what the browser tab that the game is open in will be called.

The code this template will generate for you looks like this:

```
gameTitle(`TITLE`);
```

NAMES -> ROOM NAME

This allows you to change the name of the current room. This shows on the map in the Editor as well as the exported game. You will see that all spaces you enter will be replaced by <br>, which stands for "break" and makes each word appear on a separate line, so you don't end up with a long name that goes outside of the box on the map. It's for aesthetics, but you are welcome to replace the <br>s with spaces again in the generated code if you prefer.

The code this template will generate for you looks like this:

```
roomName(`ROOM NAME`);
```

SAVE GAME – ENABLE ABILITY TO SAVE

This enables saving the game for the player. Game-saving is enabled by default.

The code this template will generate for you looks like this:

```
enableSaving();
```

SAVE GAME – DISABLE ABILITY TO SAVE

This disables the ability to save the game for the player. Game-saving is enabled by default.

The code this template will generate for you looks like this:

```
disableSaving();
```

SOUND -> PLAY SOUND

Play a short sound clip. Make sure the sound file is in the same folder as the exported game, otherwise nothing will be played!

The code this template will generate for you looks like this:

```
playSound(`FILENAME`);
```

SOUND -> PLAY MUSIC

Use this to play music. If this function is executed when music is already playing, one of two things will happen:

- If the music this function wants to play is the *same* as the music already playing, nothing will happen, the music will simply continue playing
- If the music this function wants to play is *different* to the music already playing, the music currently playing will be stopped and the new music file will be played instead

This function is incorporated into the Save Game mechanism. I.e. if the player saves the game while music is playing, it will resume when the player loads the game.

The code this template will generate for you looks like this:

```
playMusic(`FILENAME`);
```

SOUND -> PAUSE MUSIC

Stops the currently playing music. Only use this if you want silence – otherwise, just use the Play Music function to switch tracks, no need to stop the currently playing track.

The code this template will generate for you looks like this:

```
pauseMusic();
```

STATISTICS -> ADD PLAYER STATISTIC

This allows you to make the value of a variable visible to the player in the "Stats" tab of the exported game. The engine will ask you for the variable name as well as the unit you want to attach to it. For example, let's say you have a variable called pl.hp that stores the player's health, and you want to make it visible to the player. You give the engine "pl.hp" as the variable name, and perhaps "%" as the unit. It will then show A: BC in the Stats tab, where A is the variable name without the "pl." part and with all underscores changed to spaces, B is the value of the variable, and C is the unit. In this case it would be HP: (current health)%

The code this template will generate for you looks like this:

```
addStat(`VAR NAME WITHOUT THE 'PL.' PART`, `UNIT`);
```

STATISTICS -> REMOVE PLAYER STATISTIC

This will remove a previously added statistic from the Stats tab, so that the player can no longer see it.

The code this template will generate for you looks like this:

```
removeStat(`VAR NAME WITHOUT THE 'PL.' PART `);
```

TIMER -> SET TIMER

This allows you to set a timer, and execute actions of your choice after the set amount of time has passed. Please note that this is a new addition to Enigma Engine, and is still rather basic. Here are some important things to note:

- As the timer is not yet integrated into the Save Game mechanism, you must either restrict the timer to a single room (because progress in the current room is never saved, so the timer will still work as intended when the player loads the game, as they will have to progress through this room from scratch), or disable saving once the timed event starts, and then re-enable it once the timed event finishes
- You should only ever have one timer running at a time. You can have a few timers running at once, however, you will not be able to stop these timers, as the Destroy Timer command will only stop the *newest* timer you have created

The code this template will generate for you looks like this:

```
setTimer(`TIME IN SECONDS`, function() {
        |HERE GO ALL THE ACTIONS YOU WANT TO HAPPEN WHEN THE TIMER RUNS OUT
});
```

TIMER -> DESTROY TIMER

This stops the currently running timer; the actions it is attached to will never be executed. You should NOT have more than one timer running at the same time. However, if you do, this will only disable the most recently created one.

The code this template will generate for you looks like this:

```
clearTimer();
```

INVENTORY OBJECTS -> ADD OBJECT

Adds an object to the player's inventory.

The code this template will generate for you looks like this:

```
addInventoryObject(`NAME OF OBJECT`);
```

INVENTORY OBJECTS -> REMOVE OBJECT

Removes an object from the player's inventory.

The code this template will generate for you looks like this:

```
removeInventoryObject(`NAME OF OBJECT`);
```

INVENTORY OBJECT ACTIONS -> ADD ACTION TO OBJECT

Allows you to attach an action to an inventory object. It is recommended that all inventory objects have at least one action attached to them (e.g. 'inspect'). If you add an action to an object that does not exist, the object will be created and then the action will be added to it. In other words, you should never really need to use the "Add Inventory Object" function.

The code this template will generate for you looks like this:

```
addInventoryObjectAction(`OBJECT NAME`, `ACTION NAME`, function() {
        |HERE GO ALL THE ACTIONS YOU WANT TO HAPPEN WHEN THE PLAYER CLICKS ON THIS INV OBJ ACTION
});
```

INVENTORY OBJECT ACTIONS -> REMOVE ACTION FROM OBJECT

Allows you to remove an action from an inventory object.

The code this template will generate for you looks like this:

```
removeInventoryObjectAction(`OBJECT NAME`, `ACTION NAME`);
```

ROOM OBJECTS / ROOM OBJECT ACTIONS

Exactly the same as Inventory Objects and Inventory Object Actions, but instead lets you manipulate objects in the current room.

CONDITIONS

In here, we have three choices:

- If…then…
- Else if…then…
- Else…

These are very important, as they let you change what happens depending on variable values. Note that you currently cannot explicitly check if a player has something in their inventory. Instead, you must store this in a variable. For example, if you add an apple to the player's inventory, also create a variable called, for example, "pl.appleTaken" and set its value to "true" (without the quotes). Then you can use this to make different things happen depending on whether the player has taken the apple or not.

- "If…then…" – lets you execute some actions only if certain conditions are met
- "Else if…then…" – must only be used after an "if…then…" statement. It works in the same way as an "if…then…" statement, but is only executed if the above "if…then…" statement has FAILED
- "Else…" – must only be used after either an "if…then…" statement or an "else if…then…" statement. This will only be executed if all the above "if…then…" and "else if…then…" statements have failed.

The code this template will generate for you looks something like this:

```
if (CONDITIONS) {
        |HERE GO ALL THE ACTIONS YOU WANT TO HAPPEN IF THE ABOVE CONSITIONS ARE MET
}
```

VARIABLES -> SET/MODIFY A VARIABLE

Allows you to set or modify a variable. Remember to set the initial values for your variables in the Variables room. All variables must start with "pl."

For example, the code this template will generate for you might look something like this:

```
pl.health = 100;
```

DESTROY -> DESTROY ALL HYPERLINKS

Disables all currently-printed hyperlinks in the console. You will quite often need to use this as one of the actions that are executed when a player clicks on a link, otherwise the player can go back in a conversation and change their mind, or click the same link over and over again. This function is automatically executed when a player moves to a different room.

The code this template will generate for you looks like this:

```
nukeHyperlinks();
```

DESTROY -> DESTROY ALL EXITS

Disables all currently enabled exits, so that the player cannot move in any direction.

The code this template will generate for you looks like this:

```
clearExits();
```

DESTROY -> DESTROY ALL INVENTORY OBJ

Deletes all objects from the player's inventory.

The code this template will generate for you looks like this:

```
clearInventory();
```

DESTROY -> DESTROY ALL ROOM OBJ

Deletes all objects from the current room.

The code this template will generate for you looks like this:

```
clearRoom();
```

DESTROY -> DESTROY EVERYTHING

Executes all four of the above (i.e. disables all hyperlinks, disables all exits, deletes all inventory objects, and deletes all room objects).

The code this template will generate for you looks like this:

```
clearRoom();
```

DESTROY -> END GAME

Does the same as "Destroy Everything", but also stops all timers, and prints a "Game Over" message.

The code this template will generate for you looks like this:

```
killGame();
```

EXPORTING THE GAME

When you are ready to test your game, or publish it altogether, do the following:

1) Create a new folder somewhere, and name it whatever you like (e.g. the title of your game). We will name our folder “Fun Game” for this tutorial.
2) Click on the “Export Game Script” option under the “Import/Export Tab” in the Editor. This will generate a “gamescript.js” file. Put it into the Fun Game folder.
3) In the Enigma Engine folder you downloaded from GitHub, there is a folder called “Runtime Engine”, inside of which are two files – “index.html” and “runtime_engine.js”. Copy and paste both files to the Fun Game folder.
4) It’s ready! Play your game by opening the “index.html” file in your favourite browser!