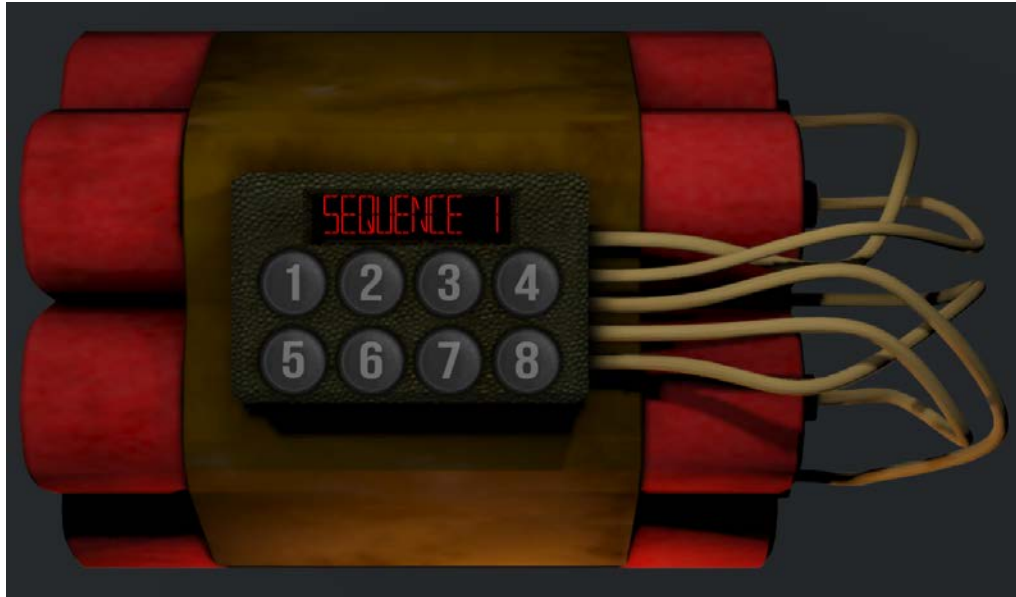


# Locks Plus: Dial Pad

Game documentation and HowTo guide.



## This document contains:

Package Description and features .....	2
PACKAGE INCLUDES.....	2
Update history .....	2
Overview of the project's library contents .....	3
Customization Guide.....	4
Getting started.....	4
LPActivator .....	4
LPDialPad.....	5
Integration with Realistic FPS Prefab .....	9
Integration with Ultimate FPS framework.....	12

## Package Description and features

LOCKS+ Dial Pad gives you a powerful locking system you can easily integrate into your FPS game. It can work as a standalone project, and can also be integrated into your current FPS project.

[Works with Ultimate FPS Framework](#)

[Works with Realistic FPS Prefab too](#)

## PACKAGE INCLUDES

- A bomb you need to defuse by repeating a button sequence correctly, or it blows up.
- A door with a dial pad, you must enter the correct numbers/letters in order to unlock it
- A standalone demo showing off many examples of the Dial Pad lock system, which also works on mobile, including bombs, doors, and a button that triggers a nuclear strike.
- Video guide to show you the basics of the system.
- Video guide to show you how to integrate the system into RFPS, with bomb damage.

**Current version 1.21**

## Update history

### **1.21 (09.09.2018)**

- Option to enter for sequence into the dial-pad before checking if it is correct or not.
- Improved input display on the screen, shows input from the first note.

### **1.20 (28.02.2018)**

- Updated integration and documentation for Realistic FPS (RFPS) package and Ultimate FPS framework (UFPS).

### **1.17 (20.12.2017)**

- Switched to Unity Events for Win/Lose actions, which allows for more parameter types to be used.
- Added support for Ultimate FPS (UFPS) package with quick integration guide.

### **1.15 (06.11.2017)**

- Added another implementation of the LOCKS+ Dial Pad system, this one using a world space UI canvas and a single button you can press to trigger a nuclear strike.
- Added Player Dummy which can be used to force the player to reset to a certain position and rotation when interacting with a lock. This is similar to how Skyrim/Fallout use dummies to position the player when interacting with workstations.
- Backwards compatibility with Unity 5.5. Rebuilt demo scene assets to work with this earlier version. Unfortunately it will not be possible to downgrade to a lower version without losing the assets. The scripts themselves can be downgraded to 5.4.

### 1.10 (15.10.2017)

- Added mobile controls to demo scene, automatically detected when playing on mobile device.
- Buttons on dial are disabled while listening to a sequence.
- You can use a DialPad lock without an activator, which makes it active from the start.
- Minor demo improvements, explosive pushes player, door animation toggles smoothly, etc

### 1.0 (21.09.2017)

- Initial version

## Overview of the project's library contents

Let's take a look inside the game files. Open the main LocksPlus folder using Unity3D 5.6 or newer. Take a look at the project library, usually placed on the right or bottom side of the screen. Here are the various folders inside:

- **DialPad:** Holds all the assets related to the Dial Pad lock system.
  - o **Animations:** Holds all the animations we use in the locks.
  - o **Fonts:** Holds the font used in the project.
  - o **Materials:** Holds some of the materials used in the demo.
  - o **Models:** Holds the 3D objects for the door and bomb.
  - o **Prefabs:** Holds all the lock examples we use in the demo, including 4 door examples and 3 bomb examples.
  - o **Scripts:** Holds all the scripts used in the project.
  - o **Sounds:** Holds all the sounds used in the demo.
  - o **Textures:** Holds all the textures used in the project.
- **DialPadDemo:** This is the main demo scene we can test.

## Customization Guide

### Getting started

Locks Plus is meant to be used as part of a game project such as Realistic FPS Prefab, or any other game that allows you to interact with objects. You can import it on its own and test the demo scene, or import it into another project and integrate it into the demo scenes of that project.

**(When importing into an existing UFPS, RFPS, or other complete packages, don't import Project Settings!)**

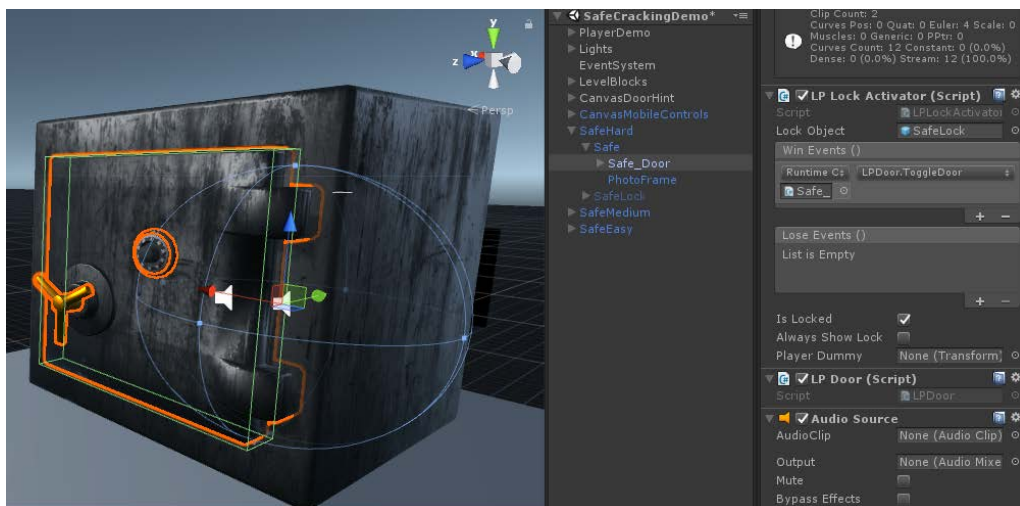
To start testing the locks, simply import the package into an empty project and open the demo scene **DialPadDemo**. Here you can see several examples of the dial pad lock you can interact with.

There are two scripts that allow you to use the Dial Pad lock system, **LPActivator**, and **LPDialPad**. Read more about them below.

### LPActivator

The Activator script allows the player to interact with objects in the scene and activate the relevant lock, then apply either a Win or Lose command which can trigger any custom function you want.

Select one of the bomb locks and take a look at the **LPActivator** component assigned to it.



**Lock Object** – The lock object that will be activated when we interact with this activator.

**Win Actions** – A list of actions that will be triggered when we successfully unlock the lock.

**Action Target** – The object that will be targeted by the action.

**Action Function** – The name of the function that will be triggered.

**Action Parameter** – An optional parameter that can be passed along to the triggered function.

**Lose Actions** – A list of actions that will be triggered when we fail unlocking the lock. It uses the same elements as Win Actions.

Winning in the bomb lock above we call a function called **RemoveObject** which will simply destroy the bomb object. If we lose we call a function **Detonate** which triggers an explosion effect from the script **LPExplosive** attached to it.

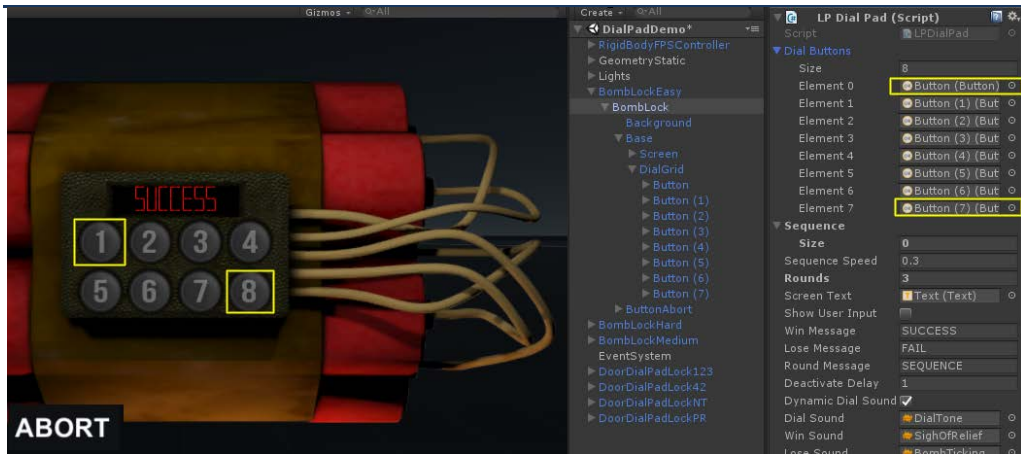
**Is Locked** – If the lock is locked, we interact with it and try to unlock it. If we succeed it becomes unlocked. If it is unlocked the Win Actions are triggered when we interact with it.

**Always Show Lock** – Always show the lock object (Don't hide it when at the start of the game or when we close it). This is good if you want to put the lock in world space, like the Big Red Button example.

**Player Dummy** – Reposition the player and rotate it to face the lock object. This is useful if you want the player to look in a certain direction when interacting with a lock in the world space, like the Big Red Button.

## LPDialPad

This is the component that holds the actual lock game with all its custom gameplay. Click on the object **BombLock** inside and unhide it.



**Dial Buttons** – This is a list of the buttons on the dial pad. Notice that the button at Element 0 refers to the first button in the pad, so 0 is actually the first button and 7 is the 8th button.

**Sequence** – The sequence of buttons that need to be pressed in order to win. If you enter these manually, the game will always start with them. If you leave them empty, the game will be entirely random.

**Sequence Speed** – The delay between each two tones in seconds. Lower number means faster dialing and harder game.

**Screen Text** – The text object on the dial pad that displays current sequence, win/lose status.

**Show User Input** – Show the user input when pressing buttons. For example we will see 123 on the screen when the player presses the 123 buttons. In the bomb example above we turned this off, but in the door examples we turned this on and you can see the input as you press the buttons.

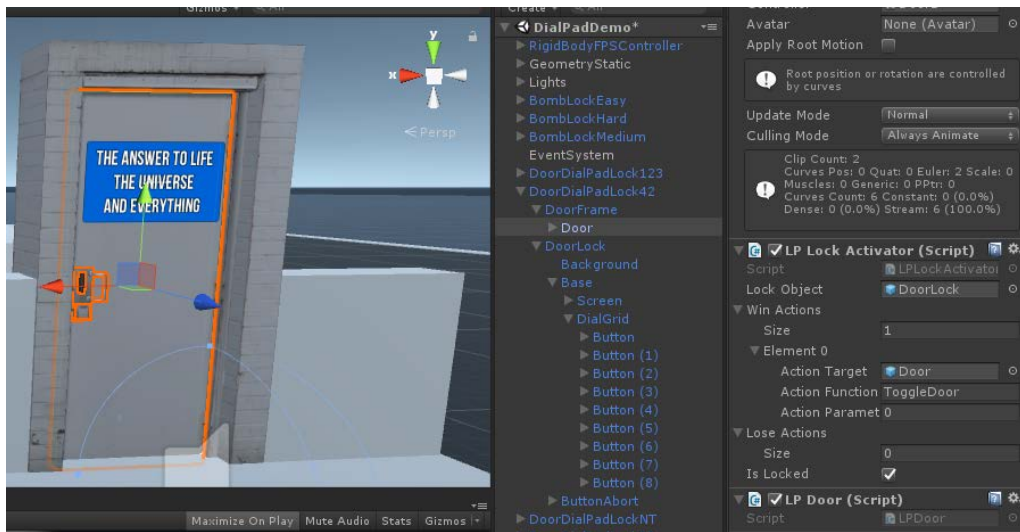
**Win/Lose/Round Messages** – The message for winning/losing, and the current sequence round. For example when showing the first round you will see the text "Sequence 1".

**Deactivate Delay** – How long to wait before deactivating the lock object when we exit the game, either after winning/losing.

**Dynamic Dial Sound** – Change the dial sound pitch based on the index of the button we press. First button pitch is lowest, last button pitch is highest.

**Sounds** – Sounds for dialing, winning and losing.

Now that we know what each of these components does, let's take a look at a different example of how they are used. Select one of the doors in the scene.

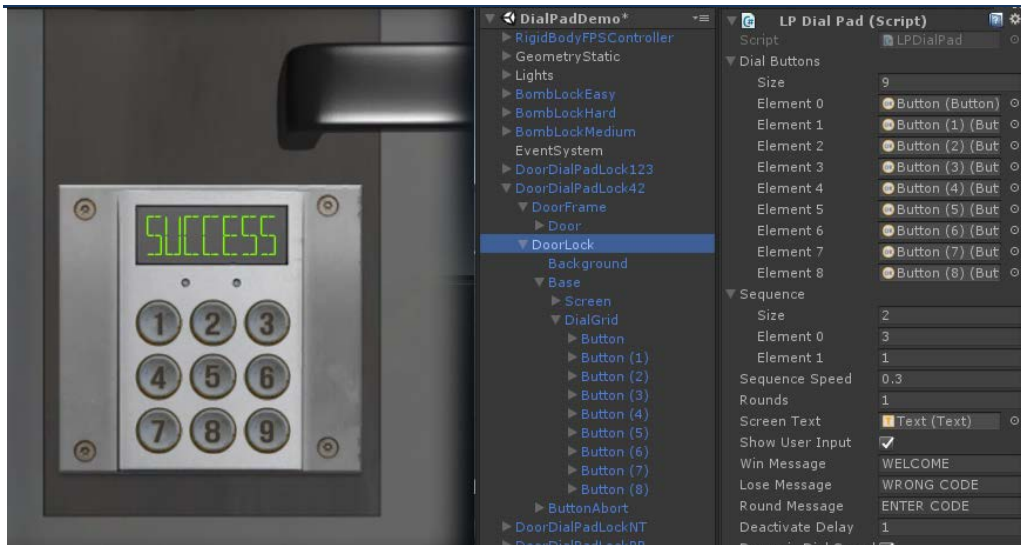


In this door, the actual activator component is assigned to the **Door** object. Click on it to see what the activator does. You'll notice that a Win action calls a function **ToggleDoor**, which is in the attached script **LPDoor** and simply toggles between two animations between opened and closed for the current door. There is also a box collider attached to the door to block the path of the player when closed. On the Lose Actions we didn't put anything, so nothing happens when we fail.

Now select the lock object **DoorLock** to see what it looks like. In this lock we have 9 buttons, and we made a pre-set sequence that the player must enter. We also set the game to 1 round so once the player enters the correct sequence (42) we unlock the lock. We also placed a text hint on the door to help the player.

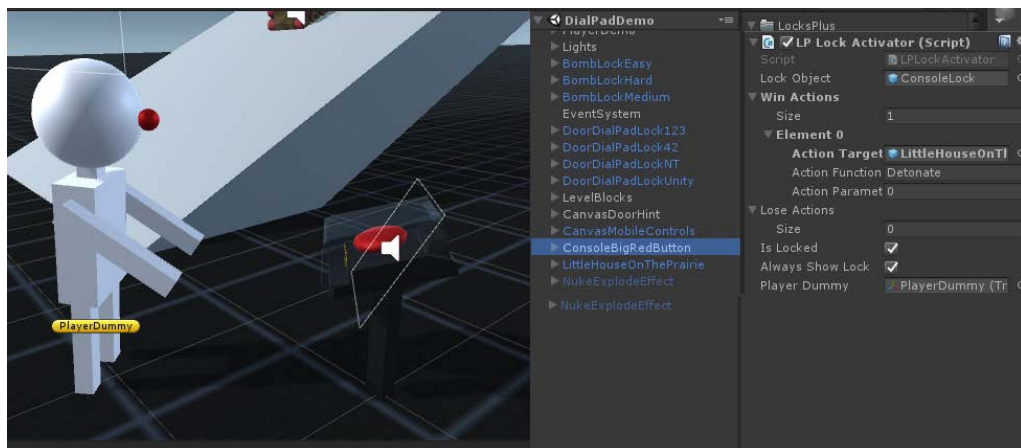
There are also examples of doors that use letter sequences instead of numbers, it doesn't matter, you can even use images as long as you assign the sequence correctly ( 0 refers to the first button, 1 is the second button, etc ).





One more example of the Dial Pad lock system is the Big Red Button, which is an example of a button placed in the world which you can interact with to trigger a nuclear strike.

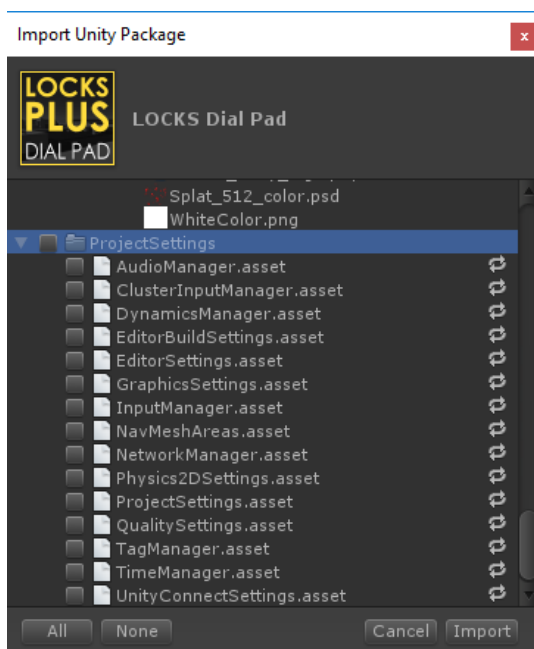
Select the object **ConsoleBigRedButton**, here you'll see two unique things, first we set **IsLocked** to false which makes the lock object appear in the scene at all times even when we are not interacting with it. The second thing is the **PlayerDummy** which is an object that will help you reposition the player and rotate it in the direction you want when interacting with the lock. This is similar to how Skyrim/Fallout use these dummies to help place the player at the correct orientation in front of a workshop for example.





## Integration with Realistic FPS Prefab

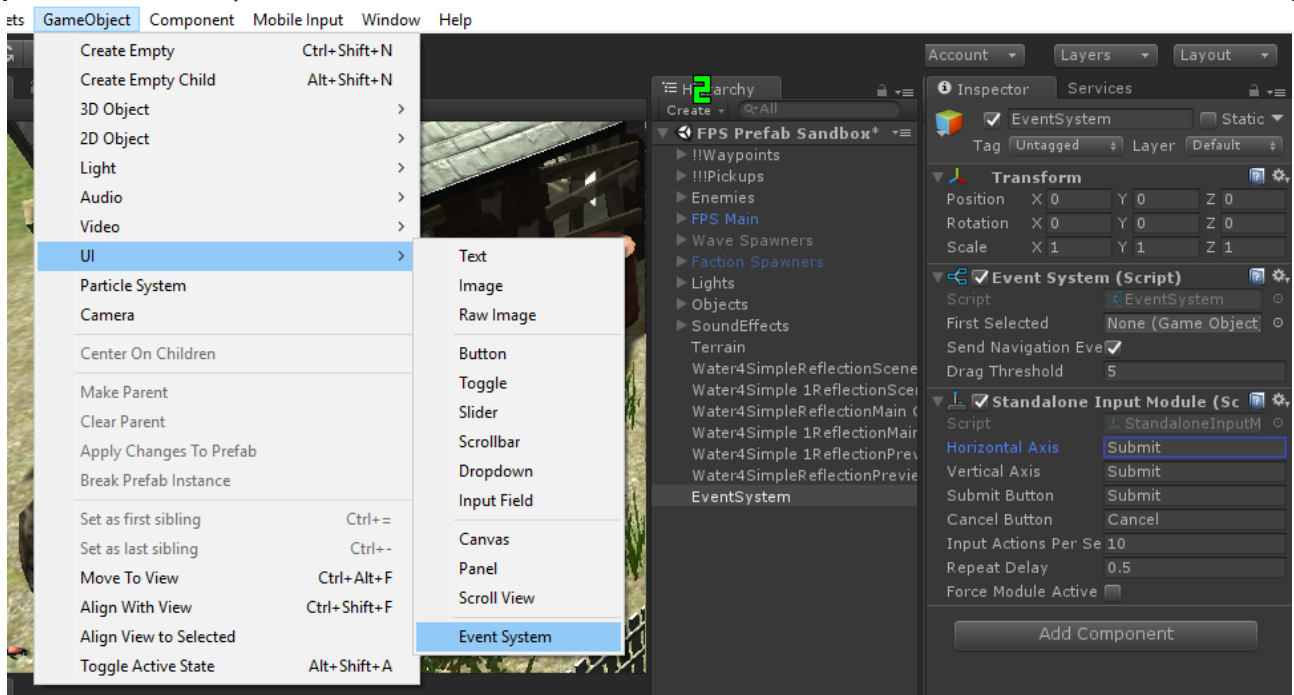
(When importing into an existing UFPS, RFPS, or other complete packages, don't import Project Settings!)



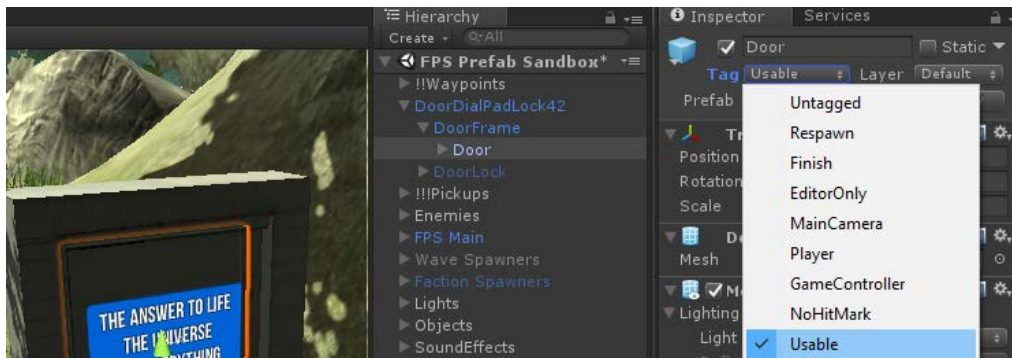
Import into your project the Realistic FPS Prefab project (tested with 1.44), open the Sandbox scene from the RFPS project, and drag one of the lock examples to the scene. In order to make it work with RFPS you need to set a few basic things.

(This guide refers to the LOCKS+ Dial Pad package, but it works the same way in all LOCKS+ packages)

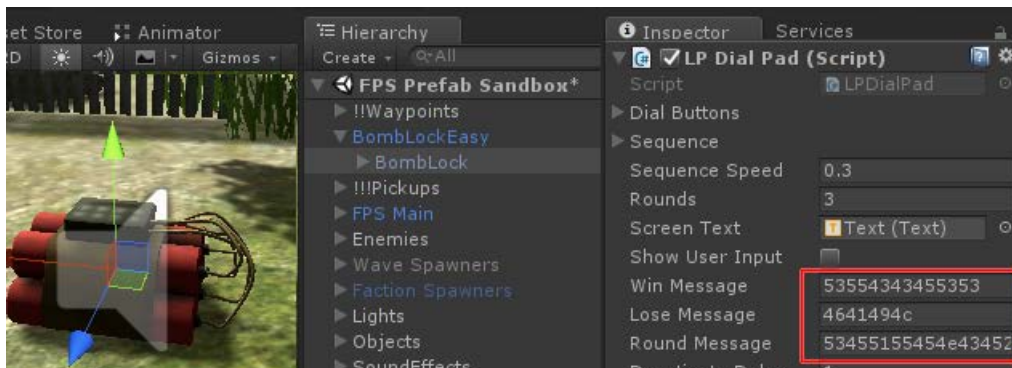
First add an **EventSystem** object, and change the "Horizontal" and "Vertical" to local inputs that RFPS can understand like for example "Submit". The lock in RFPS doesn't use that input anyway, so we just want to avoid an error.



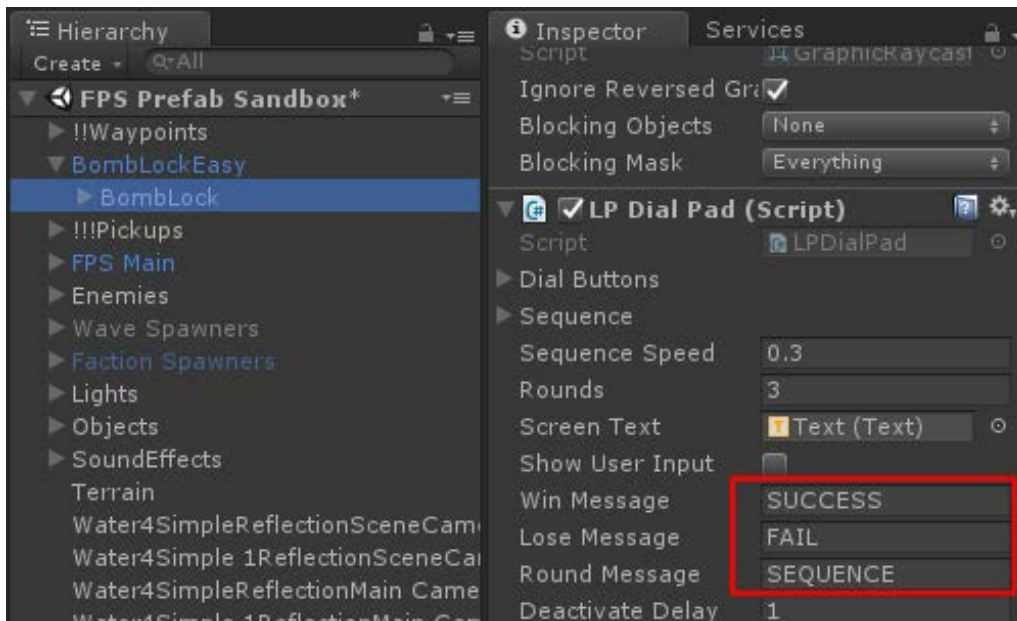
Next thing is to set the tag of your lock Activator to "Useable" which allows RFPS to interact with objects.



One last thing to check is that sometimes when importing a LOCKS+ package into RFPS, some of the text fields in the components are jumbled into random number/letter combinations, like this:



To solve this simply import the LOCKS+ package again, and it will return to normal, like this:



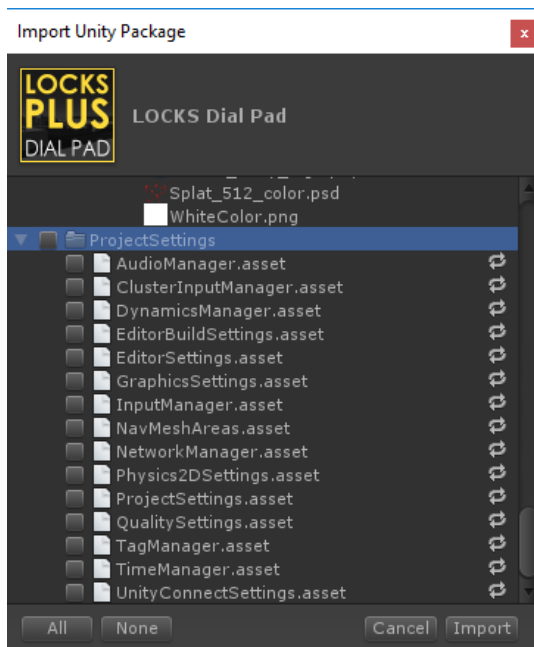
If you test the game now you'll be able to interact with the door, play the dial pad lock, unlock it and then open and close the door freely.

Check out this video to see how it's done:

<https://www.youtube.com/watch?v=Qn5V6uDUrWE>

## Integration with Ultimate FPS framework

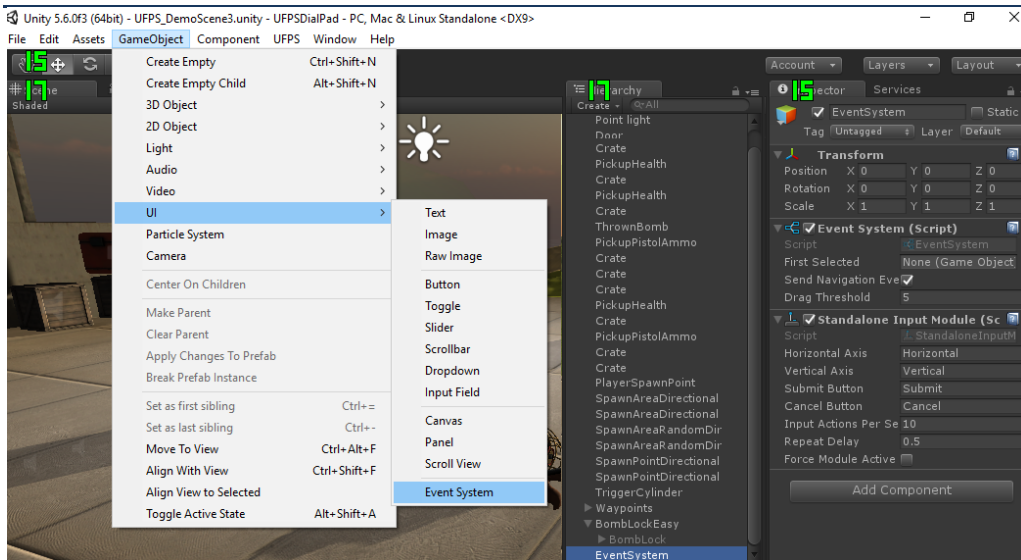
(When importing into an existing UFPS, RFPS, or other complete packages, don't import Project Settings!)



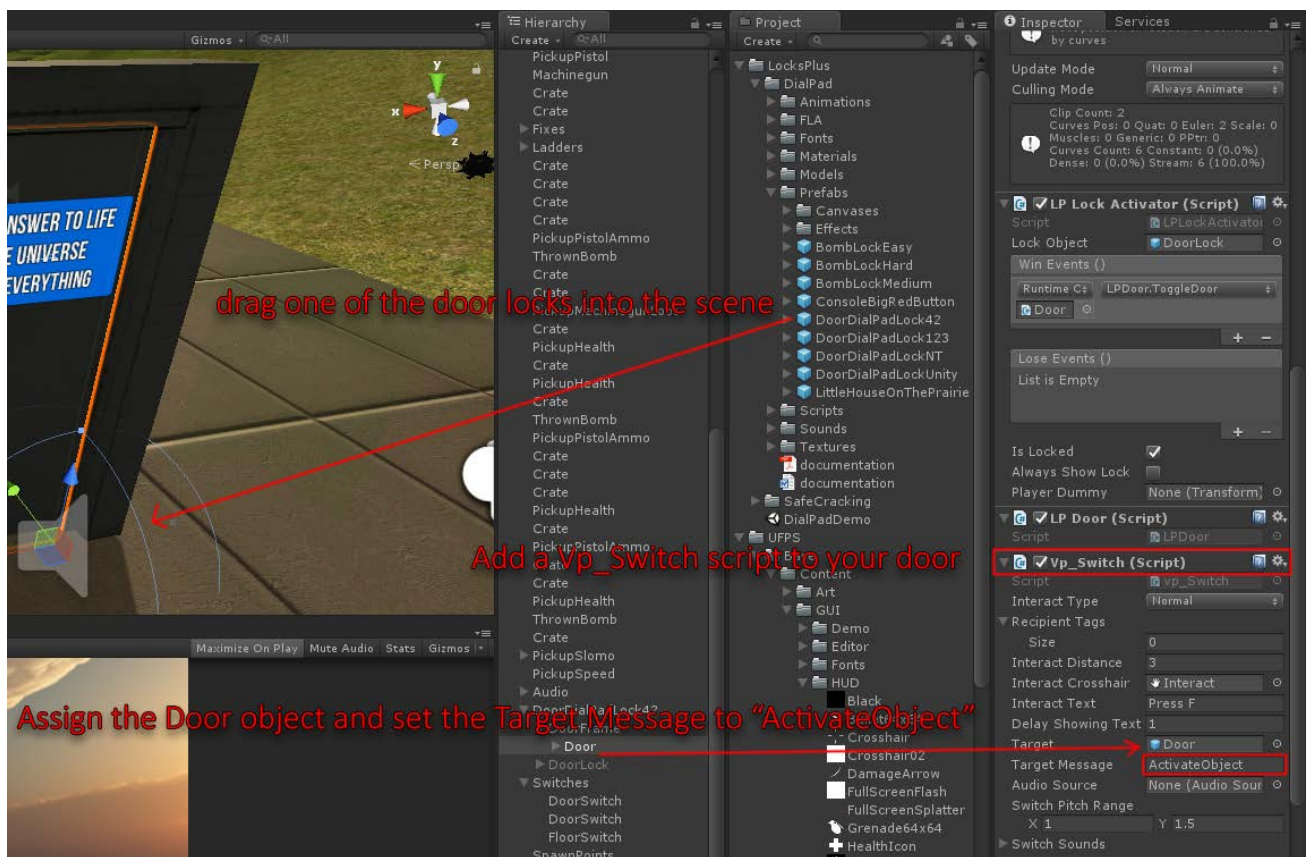
Import into your project the Ultimate FPS package (tested with 1.7.2), open the UFPS\_DemoScene3 scene from the UFPS project, and drag one of the lock examples to the scene. In order to make it work with UFPS you need to set a few basic things.

(This guide refers to the LOCKS+ Dial Pad package, but it works the same way in all LOCKS+ packages)

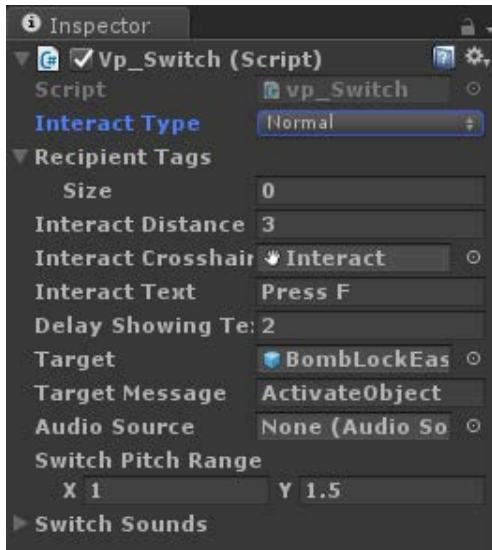
Add an EventSystem object to the scene. LOCKS+ is based on Unity UI and event system so we need that to make the buttons clickable.



Now drag one of the LOCKS+ objects into the scene and let's define it.



We need to add a Vp\_Switch script component to the object, and then define it by assigning the object that has a **LockActivator** component on it, and set the Target Message to "**ActivateObject**".



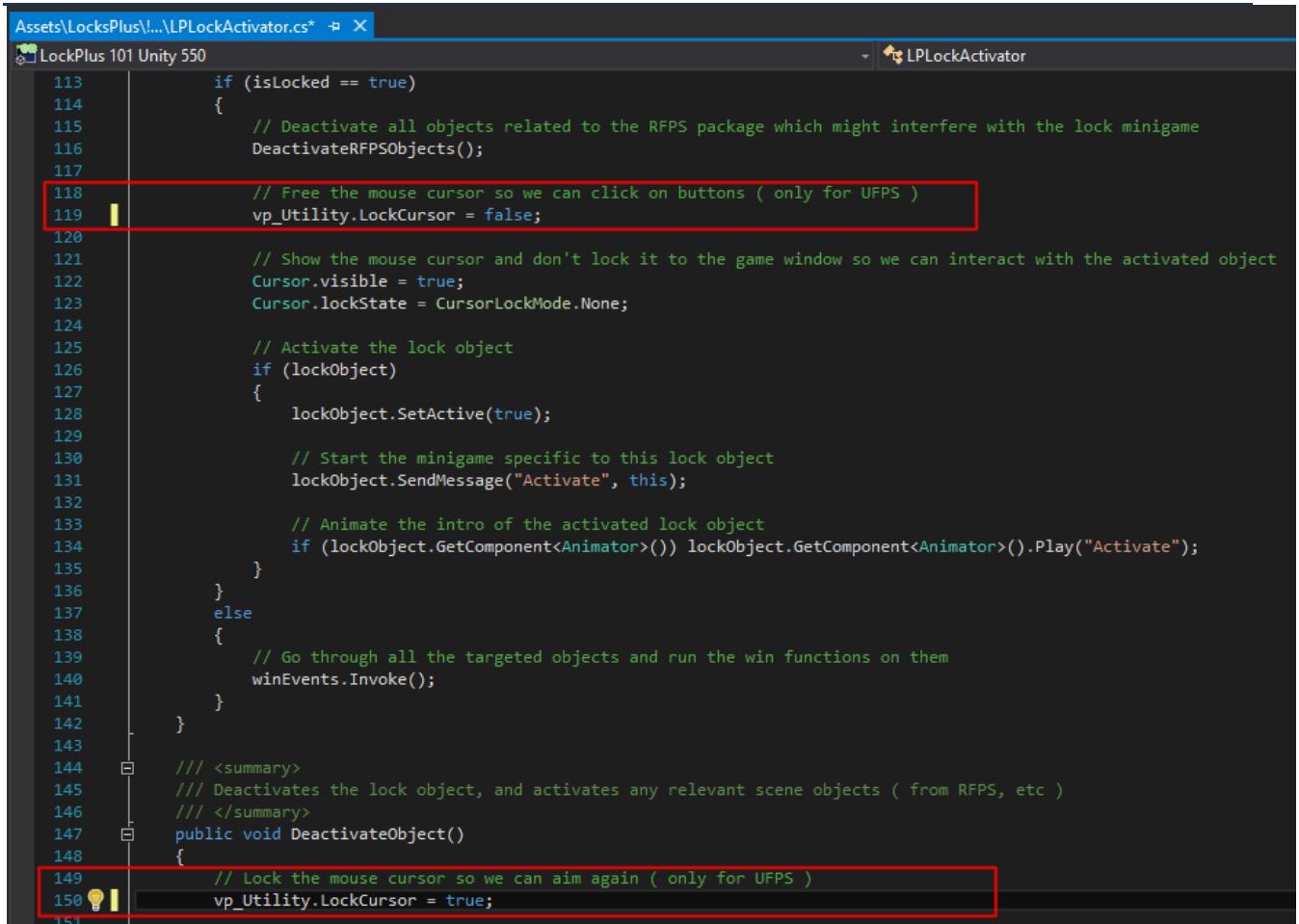
Here's a closer look at the component setup. Make sure interact type is **Normal**, and interact distance is larger than 0 so you can activate it from afar.

You can also set a **Crosshair** and **Text**.

Don't forget to target the lock object itself so we can activate it.

Finally, we need to allow the player to interact with locks by enabling mouse controls in UFPS. To do that, open the script **LP\_LockActivator** and uncomment these lines:





```
Assets\LocksPlus\!...\LPLockActivator.cs* X
LockPlus 101 Unity 550 LPLockActivator

113     if (isLocked == true)
114     {
115         // Deactivate all objects related to the RFPS package which might interfere with the lock minigame
116         DeactivateRFPSObjects();
117
118         // Free the mouse cursor so we can click on buttons ( only for UFPS )
119         vp_Utility.LockCursor = false;
120
121         // Show the mouse cursor and don't lock it to the game window so we can interact with the activated object
122         Cursor.visible = true;
123         Cursor.lockState = CursorLockMode.None;
124
125         // Activate the lock object
126         if (lockObject)
127         {
128             lockObject.SetActive(true);
129
130             // Start the minigame specific to this lock object
131             lockObject.SendMessage("Activate", this);
132
133             // Animate the intro of the activated lock object
134             if (lockObject.GetComponent<Animator>()) lockObject.GetComponent<Animator>().Play("Activate");
135         }
136     }
137     else
138     {
139         // Go through all the targeted objects and run the win functions on them
140         winEvents.Invoke();
141     }
142 }
143
144 /// <summary>
145 /// Deactivates the lock object, and activates any relevant scene objects ( from RFPS, etc )
146 /// </summary>
147 public void DeactivateObject()
148 {
149     // Lock the mouse cursor so we can aim again ( only for UFPS )
150     vp_Utility.LockCursor = true;
151 }
```

That should be all, test it out!