Lockpicking & Safecracking Toolkit

Game documentation and HowTo guide.

This document contains:

Package Description and features

Lockpicking & Safecracking Toolkit (LSToolkit) gives developers a quick and easy way to implement lock picking mechanics such as those found in some of the top AAA titles.

This project is a collaboration between Puppeteer & Jordan Swapp

Available in JS and C#

Thanks to judah4 for the C# port of LHcontainer/LHlock/LHsafeLock

Showcase and tutorial videos

LST demonstration video
LST AngryBots implementation video

Features

LST provides you with several generic scripts:

- LHcontainer component allows you to set any object such as a door, a safe, a chest etc with a lock and allow it to be unlocked and activated. Once activated you can call any function in any object, and you can even set several functions to run as a result of that container being unlocked. This allows for a lot of versatility in a game environment.
- LHlock component brings the classic game mechanic from popular games into Unity. It works exactly the same as those games, with a sweetspot range, falloff, and lockpick attributes.
- LHsafeLock component give an original and unique way to unlock safes not found in any other game. This lock type requires the player to listen carefully to get the correct sequence.
- LHBombLock component gives an new original game mechanic where you must race against time in order to defuse a bomb. The wires are color coded and randomized. There are 3 types of wires: win, lose, and timer (which changes the countdown speed when cut). You can set the colors, count of each wire type, the number of cuts required to win/lose, as well as the change of timer speed when cutting a timer wire. This is version A of the bomb game. Version B with additional functionality will be available in a future update.
- LHPanelLock component gives you a new way to challenge your players. Based on a popular horror game, in this puzzle type you must place the plugs in the correct slots within a set number of moves.

Current version 1.26

Update history

(1.26) 11.08.2015
- Updated integration guide for RFPS (Only relevant to Unity 5).

(1.25) 25.05.2015
- Updated project to Unity 4.6.5 and Unity 5.

(1.23) 15.02.2015
- Changed some of the code and Layer/Input settings to allow better integration into RFPS. Full implementation guide included in documentation. There is also a video guide.
- You can assign icons instead lock/unlock captions.
- You can set several activation animations, for example a door opens when you activate it, and then closes when you activate it again.
- Added a door prefab which you can open and pass through, as an implementation example for RFPS.

(1.21) 12.04.2014
- You can now leave the Required Tool field empty, which allows you to unlock a container without need for a certain tool in the inventory (Example is in the Panel Lock).
- When entering a lock game, the container is deactivated. It is reactivated after finishing the lock game.
- All cameras set to Clear Flags Solid with a black background. This overrides any sky boxes you may have in your game.
- Added FAQ: Importing LSToolkit into an existing project, locking cursor on exit.

(1.2) 16.03.2014
- Added a new panel lock game type. Place the plugs into the correct slots within a set number of moves, and make sure no sires intersect.
- Added the basis for the next game type: Combination Lock (still a work in progress).

(1.15) 19.11.2013
- Added a new bomb defusing game. Cut random color-coded wires and try to defuse the bomb before time runs out. Be careful not to cut the wrong wires or the whole thing will go boom!
- Added misc scripts used mainly in the preview: LHEffect creates an effect at a target position, LHDestroyObject destroys a target object with a delay, LHSplat creates a splat effect on-screen, LHLookAt is used to make the icons look at the camera at all times.
- Added the basis for the next game type: Panel Lock (still a work in progress).

(1.09) 25.9.2013
- Replaced the static LHLockpicks script with a more advanced LHinventory component. Attach this to the player and you can give him several tools to carry. When the player reaches a container, the

lock belonging to that container checks if we have the correct tool for the job before allowing us to commence unlocking.
- Code ported to C# thanks to judah4 and Jordan Swapp. Pakcage now contains both versions.

(1.03) 20.9.2013
- Added a new container and lock model, the chest and chestLockpick.
- Improved hotspot random positioning in safes.

This package is a work in progress, and will receive enhancements as they are made ready. More LST components will be created such as **Combination Lock**, **Hacking a terminal**, and more lockpicking mechanics from popular games, as well as more original mechanics like the safecracking component. If you have an idea for a lockpicking/hacking mechanic you'd like to see created, please tell me.

Credits

The textures are courtesy of CGtextures.com.

The sounds are courtesy of the free sound project.

Credits go to these authors for their great sound samples: **cognito, gelo-papas, zimbot, soundmary, vrodge, cemagar, thecluegeek, d-w, mrauralization, tdude9000, adam_n, smcameron, hardance, klarpen, georgeantoniv, kalisemorrison, and various others**

Please rate my file, I'd appreciate it 🙂

Overview of the game's library contents

Let's take a look inside the game files. Open the main LH Assets folder using Unity3D 4.1 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:

- **Fonts:** Holds the fonts used in the game as well as the main GUISkin used. The fonts are ARIAL16, and ARIAL 24.
- **Materials:** Holds all materials used in the game.
- **Models:** Holds all FBX models imported over from 3DS Max..
- **Prefabs:** Holds all the prefabs used in the game. These are distributed to various folders for easier access:
    - Generic – contains the basic prefabs that make up the package, these are LHdoor/LHsafe, LHlock, and LHsafeLock
    - Presets – readymade variations on the generic prefabs. For example you can find locks with varying difficulty configurations.
    - Misc – Holds some prefabs that are not necessary for the package, used mainly to demonstrate some functionality.
- **Scenes:** There is one scene in the package, Gallery, which showcases the available containers and lock types.
- **Scripts:** Holds all the scripts used in the game. Each prefab contains one or more of these scripts. To learn more about the editable variables in the scripts, check out the component sheet.
- **Sounds:** Holds all the sounds used in the game.
- **Textures:** Holds all the textures used in the game.

Container and lock types guide

What is a container?

In LSToolkit a container is any object that can have a lock attached to it, which can then be unlocked and activated, resulting in a meaningful consequence in-game.

It can be a locked door that opens to the next game area, or it can be a safe that holds valuables. It all depends on what you want the container to perform.

Setting up a container

A container requires the following to function:

- First of all you must attach an LHcontainer component. This defines all the variables needed to make a container work.

- You also need a collider set to trigger, which is responsible for detecting the activator (ex: a player standing near a door)
- You will also need animation and audio source in your prefab. If none are added, the script will not have any animation or sound.

Now that we have all the parts, let's build a container. For the purpose of this guide we will use the ready graphics of the door model. Drag the door model from the Models folder into the scene view. Note that the door has no materials, but that is not important for our guide.

The model already has an animation component. Click on the Add Component button under it and choose Scripts > LHcontainer. Also add a box collider component and an audio source component.

Let's define the variables in the LHcontainer component:

- Lock: While the container is still selected, go to Prefabs > Presets and drag a LHlock easy prefab to this variable.
- Locked: Let's keep this set to true, locked.
- GuiSkin: Is not important, you can set your own gui later.
- Locked Caption: Write anything you want to be displayed when this container is triggered and is locked.
- Locked Caption: Write anything you want to be displayed when this container is triggered and is unlocked.
- Caption Width/Height: The size of the text box in which the above captions are written.
- Sound Locked/Activated: Put any sounds you want to play when this container is activated while locked/unlocked.
- Activator: This is the name of the tag of the prefab that will trigger this container. It's very important to write the name correctly, or the container will not be detected. By default we set it to "Player" as that is usually the prefab that will activate the container. Don't forget to set the tag of the activator prefab accordingly.
- Activate Button: The name of the button that will activate this container. By default it is set to "Fire1" which is the Left Mouse Button as well as the Left Ctrl key. Same as above, you must write the name correctly and make sure you defined the button in the Input Manager. If no button is assigned, the container will be activated automatically on collision.
- Activate Functions: Here you can set any number of actions to happen as a result of this container being unlocked and activated. First increase the size of this array to 1 so we can see what variables we have. You'll

see we can set a receiver which is the prefab that will receive the action, and we can also set a function name which is the name of the function in the receiver prefab that will be run.

For example you can set the receiver as this door which we are working with, and also give the function name "AnimateContainer". This AnimateContainer function is actually a function we have in the LHcontainer component and it makes the door animation open and close.

You can set other functions to run as well. For example if you have prefab that is called Levels and in it there is a function called GoToNextLevel() that goes to the next level. You can set the receiver in this component to Levels and give the function name "GoToNextLevel". The result will be a door that when unlocked gets you to the next level.

- Activate Animation: This variable holds the name of the animation you wish to run on this container when it's unlocked. If you don't set anything it will simply run the default animation.

Take a look at the following image for reference on how a container might be configured.

Before we are ready to test this out, we must setup the animation and collider components. Uncheck the Play Automatically box in the animation component, and check the trigger box in the collider component. Also enlarge the collision size of the collider so that it's easy for the player to collide with it. Finally make sure the door is in the path of the limited player prefab as it can only move sideways.

Press the play button and try it out. The player should be able to move to the door, see a caption, and press a button to start the lockpicking game. After it is unlocked, the door will play an animation which we triggered using the ActivateFunctions variable.

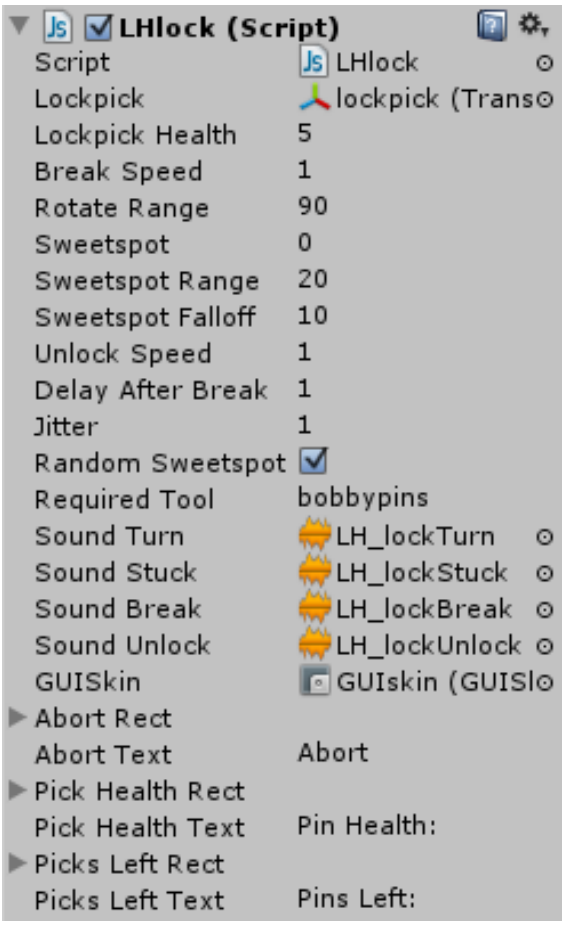
Setting up a lock

Now that we have our door ready, we want to give it a unique type of lock other than the readymade ones. Go to Prefabs > Generic and select LHlock. Press Ctrl + D to duplicate it and let's get to work:

- Lockpick: This is the lockpick object that is part of the Lock mesh. Expand your LHlock prefab hierarchy and drag the lockpick object to this variable slot.
- Lockpick: You can set the health of an individual lockpick here.
- Break Speed: The higher this number the faster the lockpick breaks. So for a hard difficulty lock you would set this to something like 6-10.
- Rotate Range: The overall range within which the lockpick can rotate. It's better to leave this at 90 as it works well gameplay-wise and logic-wise that way, and it might cause problems otherwise.
- Sweetspot: This is the center angle in which the sweetspot range starts. You can set it to something specific or you can leave it as it is, because the Random Sweetspot variable can make it random every time we play it. This way players won't be able to game the system by memorizing the positions they tried out and restarting the lock.
- Sweespot Range: An angle range within which the lock will be opened without breaking. The smaller this range the harder it is to open the lock.
- Sweetspot Falloff: Is the falloff around the sweetspot range which will let you only open the lock to an extent before starting to break. A larger falloff will make it easier for the player to slowly detect if he is getting closer to the position of the sweetspot.
- Unlock Speed: How quickly the unlock animation runs. 1 is default normal speed.
- Delay After Break: How many seconds to wait after breaking the lockpick before replacing it with another.
- Jitter: How far the angle of the lockpicks jitters when it's stuck. If this is set to a large number it might cause the player to get in the sweetspot range by luck. Keep it small, we don't want the game to be too easy.
- Random Sweetspot: If set to true it resets the lockpicking mechanic

with a new sweetspot after aborting and reactivating a container.

- Required Tool: This variable allows you to assign the name of a tool from the player's inventory which will be required in order to unlock this lock.
- The rest of the variables are fairly generic, including sound for turning the cylinder, getting stuck, breaking a lockpick, and unlocking the lock. There's also an abort button, and a text box for the lockpick health and lockpicks left.
- Mobile Icons: This last variable holds the icons that are used in Android/iOS.



Take a look at the following image for reference on how a lock might be configured.

What about lockpicks?

As you may have noticed, when starting the lockpickng game mechanic, the number of lockpicks that the player has appears on-screen. You can also see that the number of lockpicks is global and not related to one lock or the other.

This is done with a simple script; Go to Scripts > Abstract and select LHlockpicks. This is actually a static class which holds the number of lockpicks we have. It is not attached to any prefab and is accessible from anywhere. As long as you have it in your project it will work.

How can I increase the number of lockpicks I have?

When you run out of lockpicks a button appears which when pressed gives you more lockpicks. This is achieved with a simple script attached to the Player prefab that updates the number of lockpicks in the LHlockpicks class. You can find it in Scripts > Misc. For the purpose of this demo we have a button to give you the lockpicks. In a real game you will not want to give the player lockpicks with a simple button, but by picking them up or buying them.

Since the methods by which you can acquire lockpicks vary greatly and can even be considered part of a more generic inventory system, we will not make any changes to the script.

You can, however dabble with the script, remove the button and replace it with your own functions.

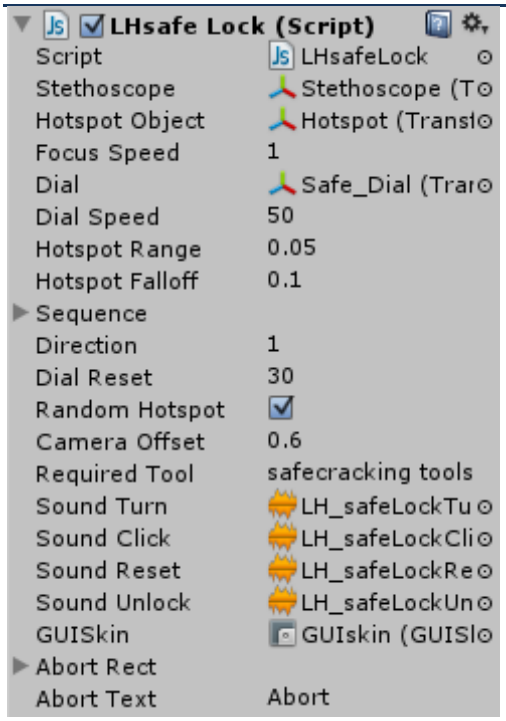
The safe lock

Another type of lock with a slightly different set of values is the LHsafeLock. Let's customize it. Select it from Prefabs > Generic and duplicate it. Now onto the configuration:

- Stethoscope: Open the hierarchy of this safeLock and drag the stethoscope to its slot.
- Hotspot Object: The object that is used as a hotspot for the stethoscope. This is the point in which the volume of the dial turning is highest.
- Focus Speed: This defines how quickly the scope settles back down after being moved. A higher value will make focus come faster and as a result make sound loud faster, which makes the game easier.
- Dial: Same as above, drag the Dial object to its slot.
- Dial Speed: How quickly the dial turns. A value too high will make it more difficult to set the dial at the number we want, and as a result cause frustration.
- Hotspot: You can set the center of the hotspot where the sound is loudest. If you put Random Hotspot on true you'll get a new hotspot with every play.
- Hotspot Range: The radius of the hotspot. In this area the sound is

loudest. A smaller value makes the game harder because it's more difficult to find the sound hotspot.

- Hotspot Falloff: A falloff around the hotspot where the sound gets fainter and fainter. A larger value will help players find the hotspot more easily.
- Sequence: This is the sequence which needs to be dialed to get the safe unlocked. You can put as many numbers as you want, which will of course make the lock much harder to solve.
- Direction: 1 is right, and -1 is left. If you start a lock with 1 you must move to the right until you reach the first number in the sequence, and then the direction flips and you must go left until you reach the next number and so on until you complete the entire sequence.
- Dial Reset: This is how much you can turn the dial in the wrong direction before the entire sequence is reset and you must do it all over again. Giving a larger value here will give players a bigger error margin and make the lock easier to solve.
- Random Hotspot: Keep it on true, better gameplay experience.
- Required Tool: This variable allows you to assign the name of a tool from the player's inventory which will be required in order to unlock this lock.
- Mobile Icons: This last variable holds the icons that are used in Android/iOS.
- The rest of the values are the same as in the regular lock.



The Bomb Defuse Lock

A new type of lock with a completely different set of values is the LHBombLock. Let's customize it. Select it from Prefabs > Generic and duplicate it. Now onto the configuration:

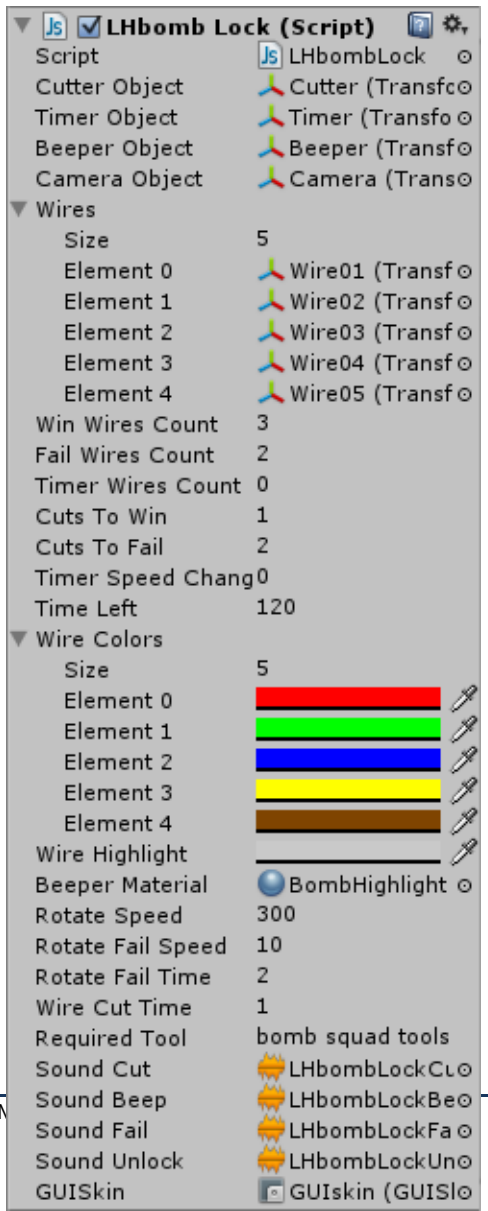
- Cutter Object: Open the hierarchy of this bombLock and drag the cutter object to its slot. This is the object that shows when you cut a wire.
- Timer Object: This is the object that has the timer box and timer text mesh that displays seconds to detonation.
- Beeper Object: This is the object that lights up when you near detonation. It lights up when the timer is less than 10 seconds left, and when you rotate the bomb too quickly.
- Camera Object: This is the camera of the bomb lock. We assign it so that it stays in place when we rotate the bomb.
- Wires: Like the above, go through your bomb lock hierarchy and add each wire to this list. These wires will be assigned Win/Loss/Timer attributes and will be color coded. Note that each wire has a base model as well as a cut model (which will replace the base model after we cut this specific wire). A wire can also have an icon model that always looks at the camera. Also note that the wire has a mesh collider.
- Win Wires Count: The number of good wires which we must cut. Set

this to 3.

- Fail Wires Count: The number of bad wires which we must not cut. Set this to 1.
- Timer Wires Count: The number of timer wires which we should also avoid cutting. These change the speed of the bomb timer and get us closer to losing. Set this to 1.

  Note: The total number of wires (Win + Fail + Timer) must not exceed the number of available wires. We have 5 total wires so we can at most assign 3 Win + 1 Fail + 1 Timer = 5 total.

- Cuts To Win: How many Win wires do we need to cut in order to disarm this bomb, and activate the Unlock functions. Setting this number to 1 should make the bomb game very easy, as it takes only one good cut to win. Of course you shouldn't set CutsToWin to a number that is higher than your WinWiresCount. (because then the player would be required to cut more Win wires than they actually are available).
- Cuts To Fail: Same as above but for the Fail wires. If you cut too many Fail wires the bomb will detonate, activating the fail functions. Same rules apply as above.
- Timer Speed Change: This is the multiplier for when we cut the timer wire. Let's set this to 2. This way when we cut a timer wire the timer will count down twice as fast.
- Time Left: Keep it on true, better gameplay experience.
- Wire Colors: This variable allows you to assign the name of a tool from the player's inventory which will be required in order to unlock this lock.
- Wire Highlight: The color of the wires when we highlight them. Choose a bright color like white or light gray.
- Beeper Material: The material



of the beeper when it lights up.

- Rotate Speed: How fast we can rotate the bomb.
- Rotate Fail Speed: If we rotate the bomb too fast it may detonate, after it beeps for a while. This is the threshold after which the bomb may detonate.
- Rotate Fail Time: This timer starts counting when we rotate too fast. If it reaches 0, the bomb explodes.
- Wire Cut Time: How many seconds the process of cutting a wire takes. The animation of the wire cutter is 1 second long, so I set this to 1 second.
- Required Tool: This variable allows you to assign the name of a tool from the player's inventory which will be required in order to unlock this lock.
- The rest of the values are the same as in the regular lock, sounds, GUI, etc.

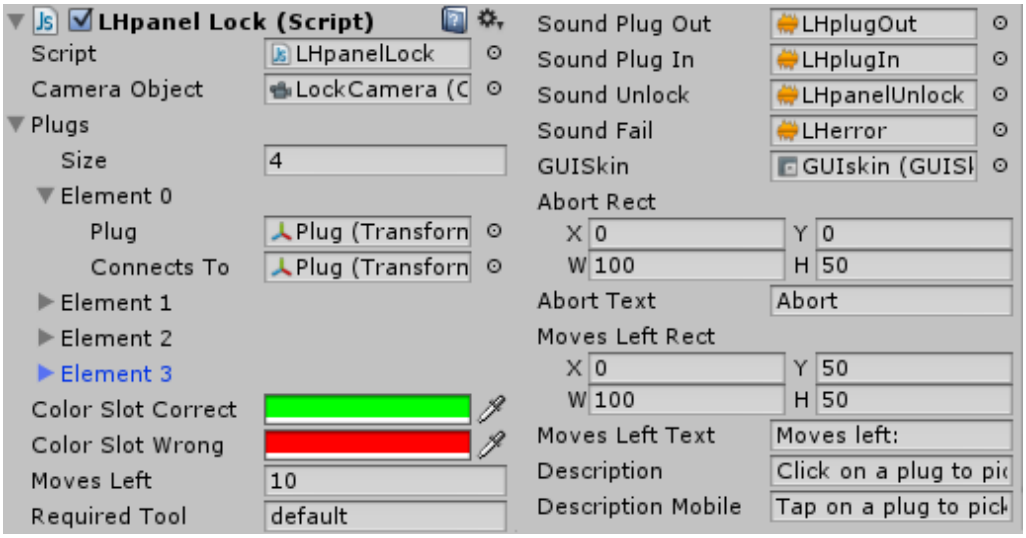
The panel lock

The panel lock (LHpanelLock) is a challenging type of game with focus on logic rather than speed. It's based on the panel puzzle from Resident Evil Extinction. Let's take a look at it. Select it from Prefabs > Generic and drag it to the stage. You will notice the following attributes:

- Camera Object: This is the camera object attached to this lock. You'll find it in the hierarchy of the lock itself.
- Plugs: This is a list of all the plugs in a lock. Each plug is connected to another with a wire between them. Gameplay-wise, if a wire intersects with another, you can't solve the puzzle.
- Color Slots: Here you can assign the color of a slot when it's correct and when it's wrong. A slot is correct if the plug is placed in it correctly.
- Moves Left: The number of moves you have left before losing this lock game. A move is counted when you pull out a plug and place it in a

different slot. If you pick up a plug and place it in the same slot you don't lose a move. The moves replenish each time you start an unsolved panel lock game.

- Required Tool: This variable allows you to assign the name of a tool from the player's inventory which will be required in order to unlock this lock. In this case it's set to "default".
- The rest of the values are the same as in the regular lock.



Creating your own panel lock

The following video shows you a step by step process of customizing a panel lock. We will duplicate a generic panel lock and create our own prefab. We will then proceed to rearrange the slots in a smiley shape, place the plugs correctly in them, and then mix up the plug positions to create a challenging puzzle. Check out the video here:

https://www.youtube.com/watch?v=tJhN9j-cARs

Component details sheet

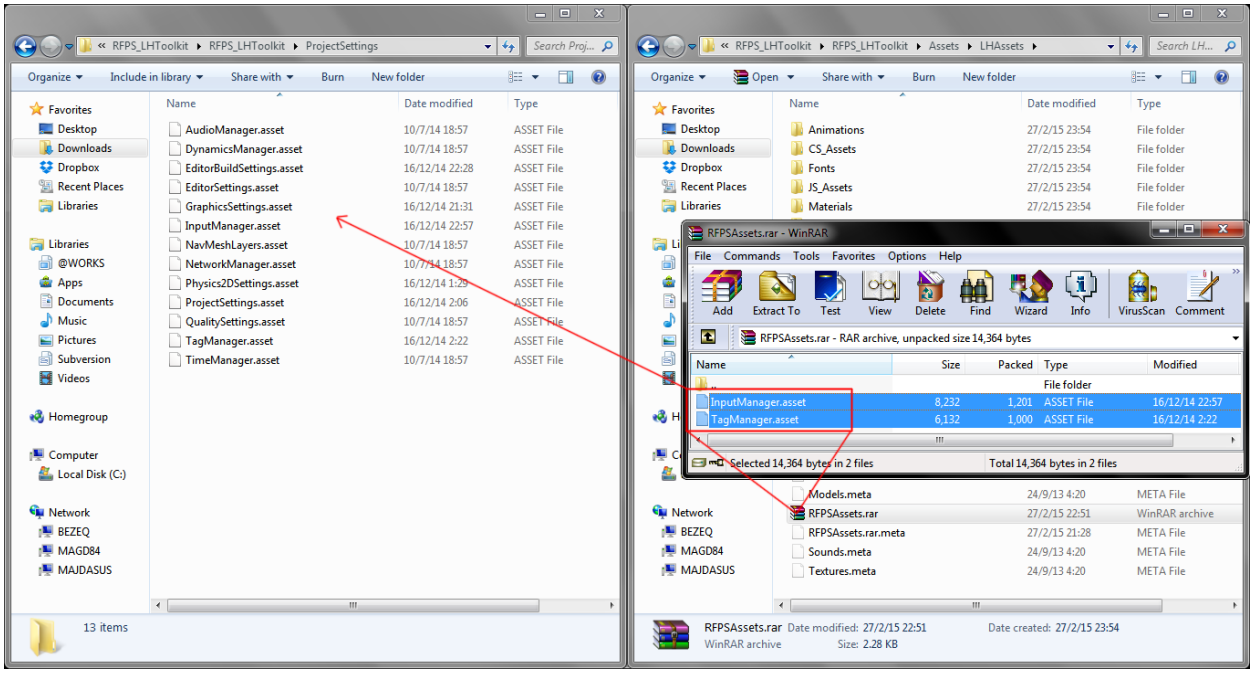
The following is a table of all the components in the package, their variables, and some details about them:

CLICK HERE TO OPEN COMPONENT DETAILS SHEET IN EXCEL

Integration with RFPS

This guide will help you import LHToolkit into your RFPS project. Let's start! As of the latest update to RFPS this guide is relevant only to the Unity 5 versions of LHToolkit.

- Import the LHToolkit package into your RFPS project. Since RFPS uses C# as the basis for its code, we will work with C# assets of LHToolkit. Don't include any of the Project Settings from LHToolkit.
- RFPS and LHToolkit each use different Layer/Tag and Input settings, so we need to combine both settings. To do that extract the contents of RFPSAssets.rar (you'll find it inside LHAssets folder right at the bottom) and overwrite the Project Settings.



- RFPS by default continuously locks and hides the cursor, which causes mouse related actions to be negated by a script called SmoothMouseLook.cs. We need to change this script to make it play nice with LHToolkit. Open !RFPSP > Scripts > Camera > SmoothMouseLook.cs and add the following code to the Update function:

```
//If the player object is active, lock and hide the cursor
if ( playerObj.activeSelf == true && Cursor.lockState == CursorLockMode.None )
{
        //Hide the cursor
        Cursor.lockState = CursorLockMode.Confined;
        Cursor.visible = false;
}
```

```
56    void Update(){
57
58        if(Time.timeScale > 0.0f && Time.smoothDeltaTime > 0.0f){//allow pausing by setting timescale to 0
59
60            //Hide the cursor
61            Cursor.lockState = CursorLockMode.Locked;
62            Cursor.visible = false;
63
```

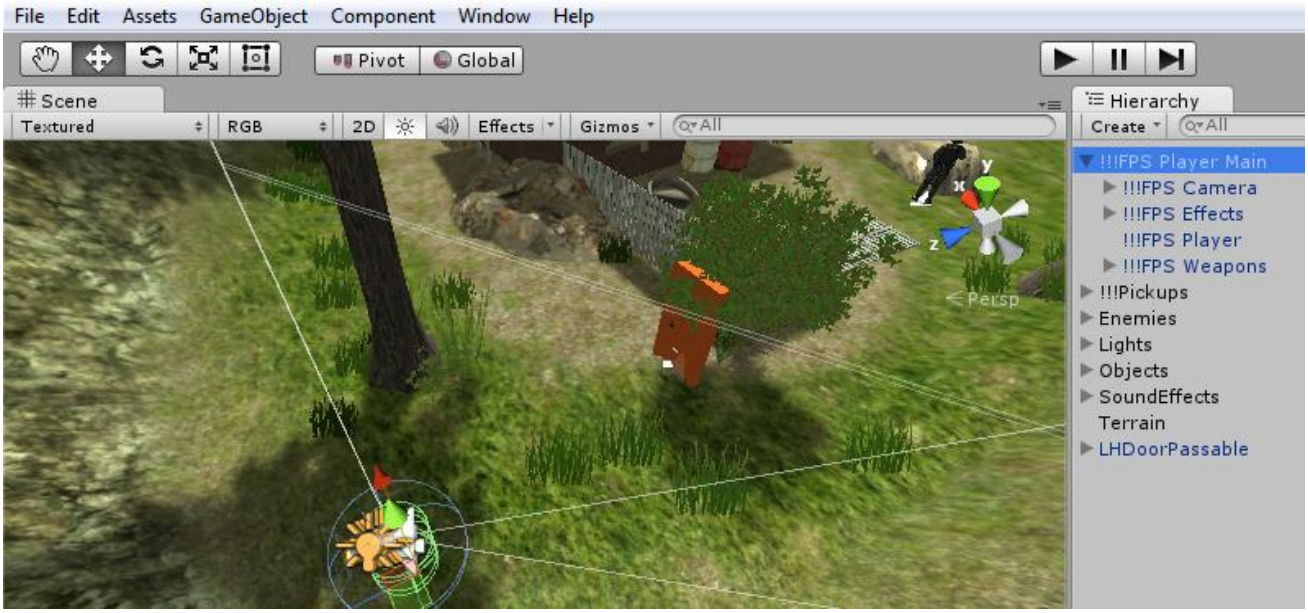
Replace this code

The result should look like this:

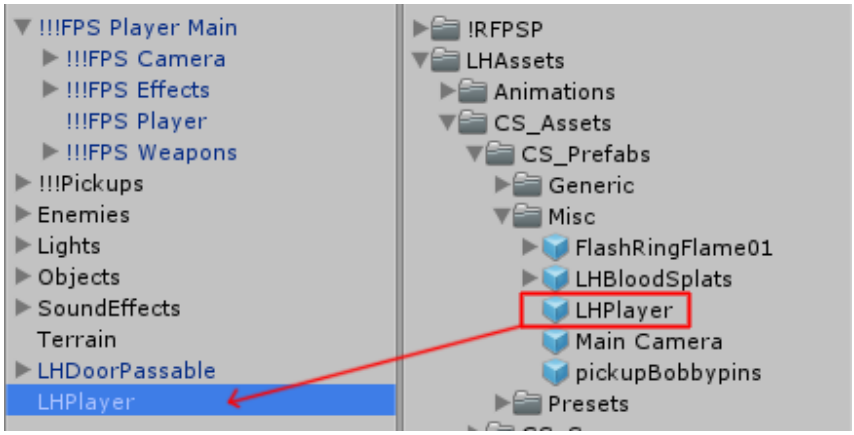
```
56    void Update(){
57
58        if(Time.timeScale > 0 && Time.smoothDeltaTime > 0){//allow pausing by setting timescale to 0
59
60            //If the player object is activate, lock and hide the cursor
61            if ( playerObj.activeSelf == true && Screen.lockCursor == false )
62            {
63                Screen.lockCursor = true;
64                Screen.showCursor = false;
65            }
66
67            // Read the mouse input axis
```

with this code

- Now we need to take care of the RFPS Player prefab by adding an inventory to it and fixing up its camera. Open any scene from RFPS and look for !!!FPS Player Main.



- Now we need to get the player prefab from LHToolkit. Drag it from CS_Assets > Misc > LHPlayer into the scene.



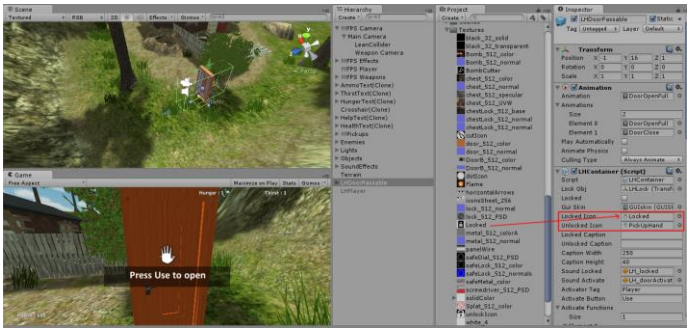
- Select LHPlayer and drag the component LHInventory to the object **FPSPlayer.** Don't forget to delete LHPlayer from the scene when you're done.

- Notice that in the current version of RFPS there are two objects tagged as "Player": The Player and LeanCollider object. In order to avoid any possible problems with this I suggest you set the tag of LeanCollider to "untagged". In future updates this will be solved by the RFPS developer.

- That's it! Time to test if it all works. Drag a passable door to the scene. You'll find one in LHAssets > CS_Prefabs > Generic > LHDoorPassable and place it somewhere close.

- Start the game and go up to the door. Press F to engage the lock. Once you unlock the door it opens and you can pass through. If you interact with the door again, it will close and you won't be able to pass through it. You can try out other lock types to see how they work in RFPS.

- You can also assign icons to the lock/unlock states of doors to better integrate the LHToolkit interface with RFPS. I made some lock/bomb reticules in the Textures folder of LHAssets.



Frequently Asked Questions

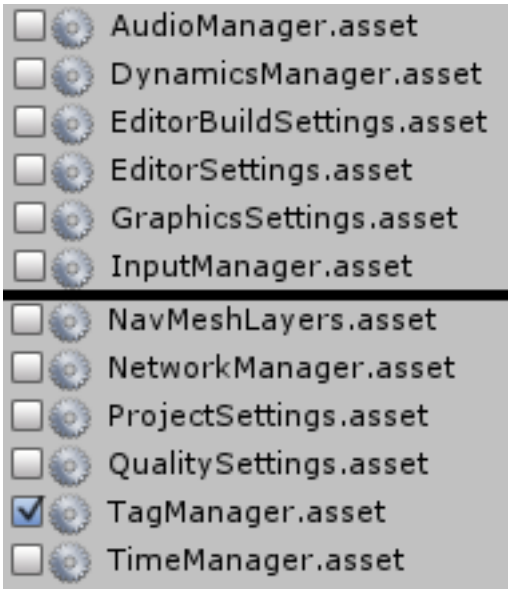
Does this package work on mobile?

Yes, this package has been successfully tested on both Android and iOS devices. The scripts for each lock type include controls for mobile that are detected automatically based on the platform it's built on.

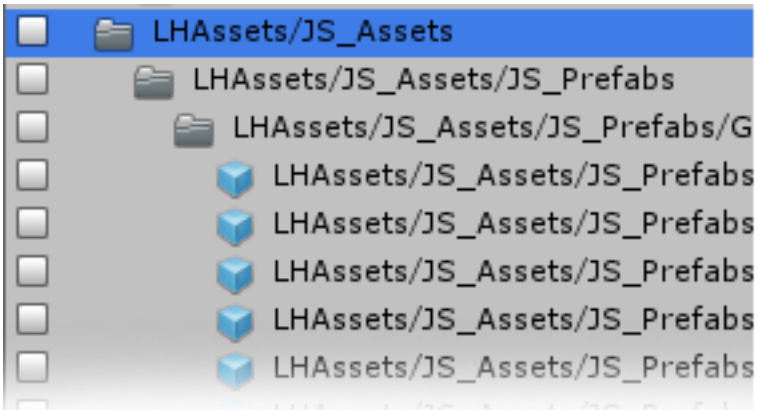
What should I import if I want to add LSToolkit to an existing project?

When importing LSToolkit into an existing project you need to do the following:

1. Uncheck all the Manager and Settings assets except TagManager.



2. LSToolkit includes code for both JScript and Csharp. You can import them both or you can choose one to work with. In the example below I chose to import the CS assets only, so I unchecked JS_Assets entirely (Just uncheck LHAssets/JS_Assets and everything related to it will not be imported).

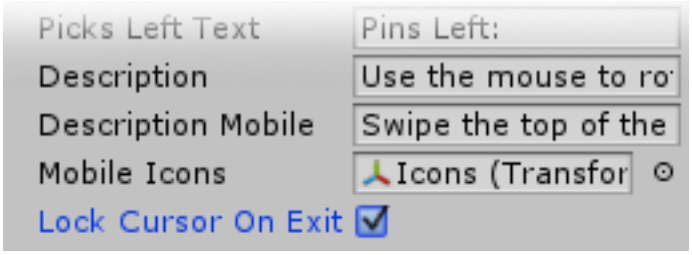


3. If you don't want any scene assets in this project you can also uncheck the demo scenes from the CS assets:

How can I lock the cursor (Screen.lockCursor) when the lock game ends?

First Person Shooter games lock the cursor to make it invisible and prevent it from going off screen or clicking on things it shouldn't click. Some of the lock games in LSToolkit show the cursor in order to allow the player to click on buttons.

Update: You can now set a Boolean to lock the cursor when exiting a lock game.



If you have any other questions please post them in the project thread
http://forum.unity3d.com/threads/200355-RELEASED-MVR-Tower-Defense-Starter-Kit

It is highly advised, whether you are a designer or a developer to look further into the code and customize it to your pleasing. See what can be improved upon or changed to make this file work better and faster. Don't hesitate to send me suggestions and feedback to puppeteerint@gmail.com

Good luck with your modifications!