Time-Based Life System

Thanks for purchasing Time-Based Life System by ExaGames!

Create a local time-based life system for free-to-play (F2P) games with a single prefab and just a few lines of code.

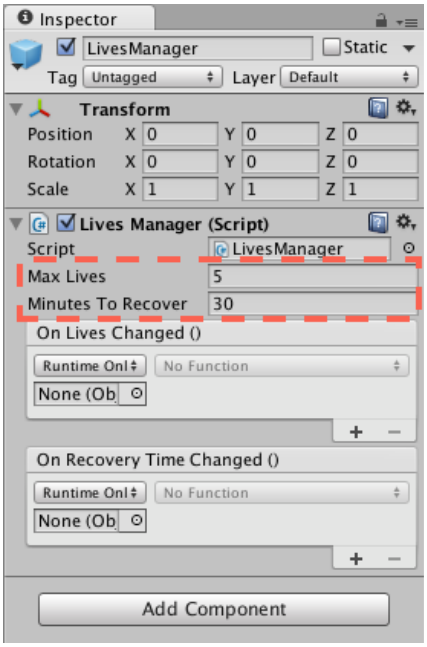
Overview

1. Import Time-Based Life System to your project.
2. Include the *LivesManager* prefab in your scene.
3. Set *Max Lives* and *Minutes To Recover* to your desired values.
4. Create event handlers to change label values when lives or time change.
5. Assign these event handlers to *On Lives Changed ()* and *On Recovery Time Changed ()* events in the prefab.
6. Call *LivesManager.ConsumeLife()* from your UI management object.

Detailed Instructions

Include the Lives Manager prefab in your scene
Search inside the folder ExaGames/Common/LivesManager/Prefab to find the *LivesManager* prefab; drag&drop it into your scene hierarchy.

Set *Max Lives* and *Minutes To Recover* to your desired values
Select the *LivesManager* object in your hierarchy; you should see something like this in the inspector:



By default, the *LivesManager* object has a maximum capacity of 5 lives and recovers one life every 30 minutes. Feel free to change these values to fit your game.

Create event handlers to change label values when lives or time change

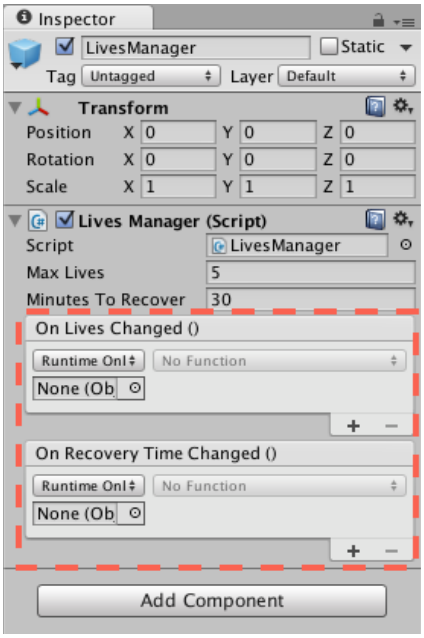
You may want to inform the player how many lives he has and how much time is remaining for the next life to come. The simplest way to do so is with a label. You can use whichever UI framework you want: legacy UIs, Unity UI, NGUI and so on.

Just create a couple of event handlers to show *LivesManager.Lives* and *LivesManager.RemainingTimeString*. You'll need to add a reference to the LivesManager object in your scene. Both event handlers must have the signature: *public void [your_handler_name]();*
The following example uses Unity UI Text objects, but you can use any UI framework you want: legacy UIs, NGUI and so on.

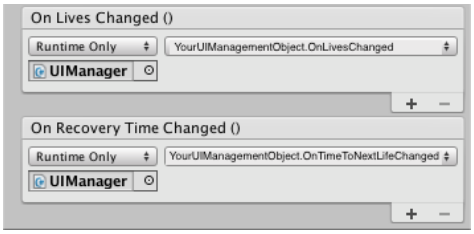
```
public class YourUIManagementObject : MonoBehaviour {
	// Reference to the LivesManager. Assign it by drag&drop in the inspector.
	public LivesManager LivesManager;
	public Text LivesText;
	public Text TimeToNextLifeText;

	// Other code…

	public void OnLivesChanged() {
		LivesText.text = LivesManager.Lives.ToString();
	}

	public void OnTimeToNextLifeChanged() {
		TimeToNextLifeText.text = LivesManager.RemainingTimeString;
	}
}
```

You can also create these event handlers in separate classes (for example, in each label object).

Assign these event handlers to *On Lives Changed ()* and *On Recovery Time Changed ()* events in the prefab
Go back to Unity Editor and select the *LivesManager* object in your hierarchy.



Drag&drop your UI management object to both events and then select the proper event handlers you have just created from the function list. If you have been following these instructions, your inspector should now look like this:



Call *LivesManager.ConsumeLife()* from your UI management object.
When you want to consume a life (for example, when the player presses the *play* button), just call *LivesManager.ConsumeLife();*

This method will return a boolean value indicating whether the player has enough lives to consume. If no lives where available, a false value is returned, and you can send the player to your store to buy lives.

To refill lives, either use *LivesManager.GiveOneLife()* or *LivesManager.FillLives()*. Don't forget to add a reference to the *LivesManager* object of your scene.

```
public class YourUIManagementObject : MonoBehaviour {
	// Reference to the LivesManager. Assign it by drag&drop in the inspector.
	public LivesManager LivesManager;

	// Other code…

	/// <summary>
	/// Play button event handler.
	/// </summary>
	public void OnPlayButtonPressed() {
		if(LivesManager.ConsumeLife()) {
			// Go to your game!
		} else {
			// Tell player to buy lives, then:
			// LivesManager.GiveOneLife();
			// or
			// LivesManager.FillLives();
		}
	}
}
```

Demo scene

The demo scene included in the folder ExaGames/Common/LivesManager/Demo recreates the instructions above with *DemoScript.cs* as UI manager.

You can use this demo scene as starting point for your development.

When deploying your game, you can safely remove this folder to save some disk space if you don't need it.

One more thing…

Check out other great assets at the ExaGames site in the Asset Store.

And don't forget to visit our website to stay up to date with ExaGames' new assets and games. You may also find some free games to play! Go to:

www.exagames-studio.com

Thank you from the ExaGames team!