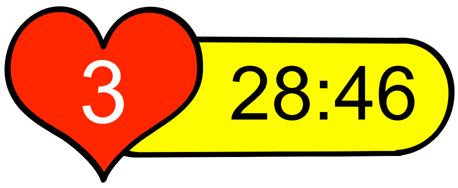
Time-Based Life System

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.



Basic Configuration

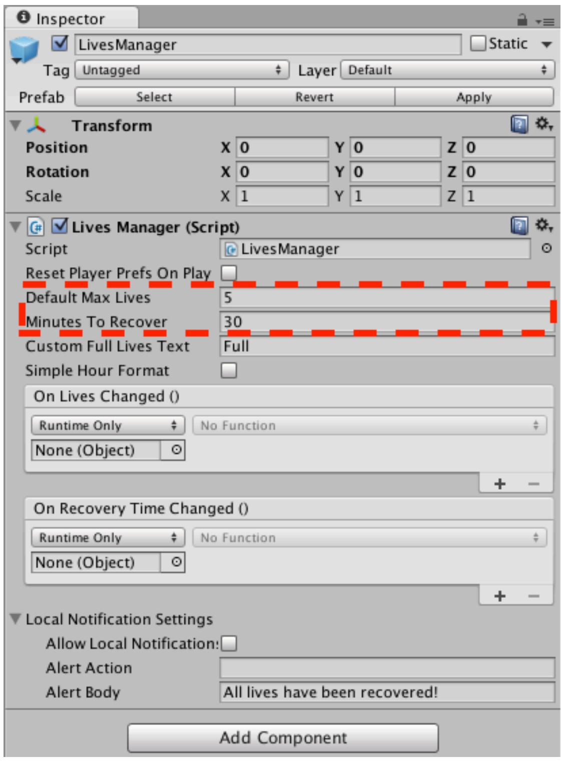
Overview

1. Import Time-Based Life System to your project.
2. Include the *LivesManager* prefab in your scene.
3. Set *Default 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 *Default 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:



Inspector
LivesManager — Static
Tag: Untagged — Layer: Default
Prefab: Select / Revert / Apply

Transform
Position: X 0, Y 0, Z 0
Rotation: X 0, Y 0, Z 0
Scale: X 1, Y 1, Z 1

Lives Manager (Script)
Script: LivesManager
Reset Player Prefs On Play: ☐
Default Max Lives: 5
Minutes To Recover: 30
Custom Full Lives Text: Full
Simple Hour Format: ☐
On Lives Changed (): Runtime Only — No Function — None (Object)
On Recovery Time Changed (): Runtime Only — No Function — None (Object)

Local Notification Settings
Allow Local Notifications: ☐
Alert Action:
Alert Body: All lives have been recovered!

Add Component

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

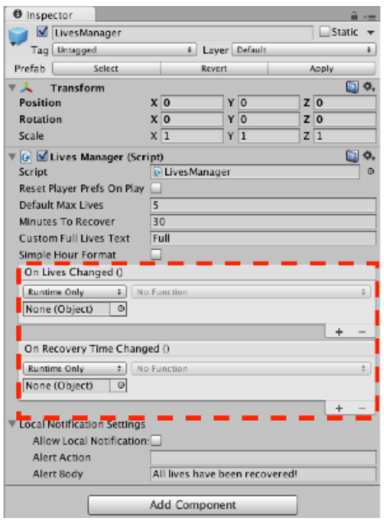
You may want to inform the player how many lives he has and how much time is remaining for the next life to come. The simplest way to do so is with a label. You can use whichever UI framework you want: legacy UIs, Unity UI, NGUI and so on.

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 as mentioned above.

```
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;
	}
}
```

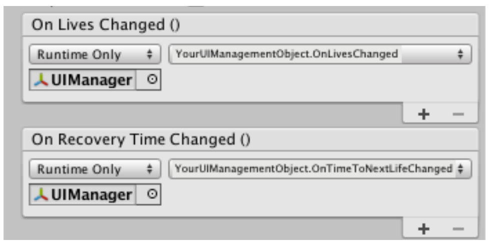
You can also create these event handlers in separate classes (for example, in each label object).

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.



```
Inspector
LivesManager                          Static
Tag Untagged            Layer Default
Prefab  Select   Revert   Apply
Transform
Position   X 0   Y 0   Z 0
Rotation   X 0   Y 0   Z 0
Scale      X 1   Y 1   Z 1
Lives Manager (Script)
Script                     LivesManager
Reset Player Prefs On Play
Default Max Lives          5
Minutes To Recover         30
Custom Full Lives Text     Full
Simple Hour Format
On Lives Changed ()
Runtime Only   No Function
None (Object)
On Recovery Time Changed ()
Runtime Only   No Function
None (Object)
Local Notification Settings
Allow Local Notification:
Alert Action
Alert Body                 All lives have been recovered!
Add Component
```

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 the following:



On Lives Changed ()
Runtime Only | YourUIManagementObject.OnLivesChanged
UIManager

On Recovery Time Changed ()
Runtime Only | YourUIManagementObject.OnTimeToNextLifeChanged
UIManager

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 restore 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. It also includes the functionality to fill grant infinite lives and increase the maximum.

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.

Advanced Use

Give infinite lives

You can grant your players infinite lives for a limited amount of time just by calling the *LivesManager.GiveInfinite(minutes)* method and sending the amount of minutes you want them to be granted in the *minutes* parameter.

Increase maximum number of lives

Call the *LivesManager.AddLifeSlots(quantity)* method to increase the maximum amount of lives by the specified *quantity.*

An additional and optional second parameter can be used to indicate this method to fill lives to the new maximum.

Activating Unity Notification Services

Use this option to notify your player when his lives are replenished.

1. Select your *LivesManager* GameObject in the Hierarchy.
2. In the inspector, open the Local Notification Settings container.
3. Check the *Allow Local Notifications* option to activate Unity Notification Services.
4. Optionally, set the *Alert Action* field to a string that helps you identify the notification in case you want to handle it.
5. Set the *Alert Body* to the text you want to show your players when lives are full.

LivesManager Inspector fields

These are the values you can set in the inspector for the LivesManager:

- **Reset Player Prefs On Play**: For debug purposes. Set this value to true to reset the LivesManager preferences on start when playing in the Editor.

- **Default Max Lives**: Maximum number of lives by default for all players. Additional life slots can be added for the player with the *AddLifeSlots* method.

- **Minutes To Recover:** Time to recover one life in minutes.

- **Custom Full Lives Text**: Text to be used in the time observer (label) when lives are full. If empty, a string with "00:00" value is used.

- **Simple Hour Format**: When this field is checked and the remaining time to restore the next life is greater than one hour, the remaining time is shown as "> X hrs"; when false, the remaining time string is formatted as "hh:mm:ss".

- **On Lives Changed**: Event to be called when the number of lives has changed.

- **On Recovery Time Changed**: Event to be called when the time to recover one life has changed.

- **Local Notification Settings:**
    - *Allow Local Notifications: When checked, activates support for Unity Notification Services.*
    - *Alert Action*: Custom string to identify the notification.
    - *Alert Body*: The message displayed in the notification alert.

API

Besides the inspector fields, the *LivesManager* class exposes the following properties and methods which can be accessed from code.

Properties

- bool **CanPlay**: Gets a value indicating whether there's lives available.

- bool **HasInfiniteLives**: Gets a value indicating whether the player has infinite lives.

- bool **HasMaxLives**: Gets a value indicating whether lives are at their maximum number.

- int **Lives**: Gets the current number of available lives.

- string **LivesText**: Gets the text that should be shown as the number of lives remaining: either a number or an infinite symbol.

- int **MaxLives**: Gets the maximum number of lives for the current player.

- string **RemainingTimeString**: Gets the string to be shown as remaining time for next life, formatted as "mm:ss".

- double **SecondsToFullLives**: Gets the total number of seconds remaining to replenish all lives.

- double **SecondsToNextLife**: Gets the time remaining until the next life is restored, in seconds.

Methods

- **void AddLifeSlots(int quantity, bool fillLives = false)**: Increases the maximum amount of lives by the specified quantity, optionally restoring lives to the new maximum when *fillLives* is set to true.
- **bool ConsumeLife()**: Consumes one life if available, and starts counting time for recovery. Returns true when life could be consumed.

- **void FillLives()**: Restores all lives.

- **void GiveOneLife()**: Restores one life.

- **void GiveInfinite(int minutes)**: Grants infinite lives for the specified amount of *minutes*.

- **void ResetPlayerPrefs()**: Resets all the preferences of the Lives Manager. **Use with care**.

Support

If you have any issues, suggestions or inquiries about this asset, please write a note explaining your problem to this email address:

info@exagames-studio.com

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!