EZ Replay Manager

App Version: 1.53
Document version: 3.1
Last Update: 14 december 2013
Required Unity version: 4.2.2 or above
Author: SoftRare - www.softrare.eu
Genre: Unity3d plugin
Description: This tool allows you to record any game for direct replay. Record any game scene and replay it backwards or with double speed! It can also easily record camera movement. In version 1.5 a new feature has been introduced on popular demand: save replays to file and review them later.

Always find the most recent version of this document here:
http://www.softrare.eu/unity/EZReplayManager/Readme.pdf

Table of Contents

1. What is this?

The EZ Replay Manager is an extension for Unity3d which allows you to record whole games or only parts of it, and replay it back while being able to precisely configure what possibilities for configuration the user has.

2. Features

- Record any game object (also cameras, rigidbodies, characters,...).
- Specify easily what should be recorded.
- Programming skills not necessarily required.
- Configure whether the user should be able to interact (starting/ending the recording/replay).
- Change replay speed, pause, stop, rewind
- Full heavily commented source code
- Fully extendable
- Since version 1.5: To lower the amount of data only real changes are being recorded (keyframe method)
- Since version 1.5: Save a recorded replay to a file and reload it when you please.
- Since version 1.5.3: In-editor documentation
- Since version 1.5.3: callback functions to be able to intervene in certain situations
- Since version 1.5.3: functionality to support a wider range of complex GameObjects

3. Quick start

Usage is extremely easy: If you have an easy scene without game object instantiation you don't even have to be able to program.

1. Drag the prefab *EZReplayManager* into your scene.

Non-programmers:

2. Click on it in the scene hierarchy. It has a list called *Game Objects To Record*.
3. Drag your crucial game objects from the scene just on the list itself (right above the Size field). Leave all other **EZReplayManager**-parameters as they were.
(Continue reading at **step 5** below..)

If you feel comfortable programming, the steps are equally easy:

2. Make a new script, call it *ObjectToRecord.cs* (for example).
3. In the start-function type the following:

```
void Start() {
     EZReplayManager.get.mark4recording(gameObject);
}
```

4. Drag this script on all prefabs and gameObjects which you would like to have recorded.

5. Start your game.
6. Hit *Start recording* and play your game.
7. Hit *Stop recording* to stop without viewing the replay, and *Replay* to do both instantly.

Congratulations ☺ You have just recorded and replayed your first game scene!

See a working demo at http://www.softrare.eu/ez-replay-manager.html

Need support? Get it for free in this thread: http://forum.unity3d.com/threads/92854-EZ-Replay-Manager

In order to save replays to and load from files please view chapter 4.2 of this document.

4. Advanced Programming How-To

4.1 EZReplayManager-API details

For comfortable recording and replaying any game, it is important to understand the plugin structure. Fortunately, this is easy as pie. In this chapter we will walk you though a number of important functions and variables that you are able to use in order to exactly implement the functionality you need.

If you are using this plugin since an earlier version you will notice very slight changes, but nothing earth shattering ☺

If you are the "self-teaching" type of programmer, we additionally encourage you to use a scripting editor like MonoDevelop (comes with Unity by default) to explore the plugins possibilities with the syntax highlighting feature:

1.
In one of your own scripts type

```
EZReplayManager.
```

to be able to view all static variables and functions the plugin has to offer.

Then, and more importantly, type

```
EZReplayManager.get.
```

to be able to view all variables and functions bound to the singleton instance of the plugin.

2.
While your game is running, you can basically mark any game object (also cameras, skies, GUI elements, and of course objects crucial for the game) to be recorded by calling

```
EZReplayManager.get.mark4recording(YOUR_GAME_OBJECT_HERE);
```

Nothing will happen so far. The only thing you say to the Unity engine and the EZ Replay Manager framework is this: "When I hit record, observe this game object, record what it does!". So you haven't hit that yet (to let the plugin save a recording to file, read chapter 4.2 of this document).

As of version 1.53 there is a new feature to be used with complex objects in which children change order/structure during gameplay. For recording objects like that call

```
EZReplayManager.get.mark4Recording(YOUR_GAME_OBJECT_HERE, "", ChildIdentificationMode.IDENTIFY_BY_NAME);
```

Note that when using this method you should make sure all children of this particular GameObject have unique names.

We will discuss the parameter in the middle (prefabLoadPath) shortly.

3.
To actually record, call

```
EZReplayManager.get.record();
```

Or just use the in-game GUI.

The framework will record the game objects you ordered it to record, for every frame, for every object. From this moment on you can still say

```
EZReplayManager.get.mark4recording(YOUR_GAME_OBJECT_HERE);
```

So this is useful to use on game objects which were created dynamically by calling *GameObject.Instantiate(..)* after recording started. Or just put on them the above mentioned *ObjectToRecord.cs*.

4.
So all this was pretty easy. Now it comes to replaying your game. Keep in mind that between switching the recorder actions such as record, play, and rewind, you should call

```
EZReplayManager.get.stop();
```

5.
This resets some important variables and avoids warnings being displayed. So now call stop, and then

```
EZReplayManager.get.play(0);
```

The parameter 0 says that the replaying speed should be the recording speed, so no speeding up or slowing down. You can do that by giving a value above 0 (standard between 1 and 5) for speeding up, and below 0 (standard between -3 and 3) for slowing down.

If you like to add some additional parameters that's fine too ☺ Here is the declaration of the function play:

```
public static void play(int speed, bool playImmediately, bool
backwards, bool exitOnFinished) {
      ...
}
```

So we discussed the first speed parameter. The next (*playImmediately*) tells the replay manager to start the replay immediately when in replay-mode without waiting for some user input. *backwards* results in the first replay being played in the opposite chronological order.. starting with the end of your recorded game back to the start. While in replay-mode, *exitOnFinished* exits the replay once the replay has been finished.
So,

```
EZReplayManager.get.play(-1,false,true,true);
```

i.e. results in a replay a little slower than normal, the replay won't start immediately, but played backwards by default and the replay mode exits after the replay is finished.

4.2 Saving to and loading from file

Save a just recorded replay to a file by calling

```
EZReplayManager.get.saveToFile(filename);
```

Load an earlier saved replay by calling

```
EZReplayManager.get.loadFromFile(filename);
```

(or press the corresponding button in the included examples)

The parameter *filename* can be a relative (i.e. only *replay.ezr* , the file is then being saved or loaded from the project directory) or absolute filename (including filepath). The demo scenes which are shipped with the package already have *save*- and *load*-buttons integrated.A few things have to be kept in mind:

1. The **EZReplayManager** does **not** save whole GameObjects (including meshes, scripts, ect) to the file. It only saves to file, what is preconfigured to (by default only position, rotation and if a GameObject emits particles. You are free to add anything to your liking. How to do this, is described in chapter 4.4 of this document).
2. That means the GameObject-data has to be taken from somewhere when loading a replay. So there has to be ready-to-use prefab from which the GameObject can be loaded. You can specify a prefab-path yourself or you can let the plugin precache GameObject structures for you. Keep in mind that the precaching process can take quite a while to finish. That leads to the following ground rule:

IMPORTANT: In order to save recordings to file and to recover them afterwards you have to enable the option *precacheGameObjects* in the EZReplayManager-prefab.

Precaching actually has other benefits than only being able to save to file. If children of your gameobject are being deleted during gameplay, this option is mandatory in order for the plugin to be able to replay correctly.

To avoid precaching on a certain GameObject you can specify prefab-paths yourself. Put your GameObject in a directory called *Resources* and mark the GameObject like that:

```
EZReplayManager.get.mark4Recording(YOUR_GAME_OBJECT_HERE,
PREFAB_LOAD_PATH_HERE);
```

One Example for this is example1.unity scene included in the package, where you have the following line in the file Heli2record.cs:

```
EZReplayManager.get.mark4Recording(gameObject,"Heli");
```

Heli is a prefab in EZReplayManager/example1/Resources . By using this function call the chopper prefab itself doesn't have to be precached in Assets/Resources/EZReplayManagerAssetPrefabs/ (which it otherwise would). To clear the cache you can always just remove this directory.

Note: Children objects of GameObjects will always be precached if *precacheGameObjects* is activated.

4.3 Important variables

Dealing with these variables is not absolutely necessary if you are satisfied with the standard behavior of the replay manager. But in some situations it can be helpful.

1.
Most importantly and foremost it has to be said that as of version 1.53 documentation for the most commonly used variables has been moved inside the Unity3d Editor. There, read documentation on how to handle the all important **EZReplayManager**-prefab which has to be instantiatated in your scene in order to activate any replay-functionality. Just enable “show hints/documentation here”.

Find additional documentation in the following paragraphs:

2.

```
public const bool showErrors = true; //default: true
public const bool showWarnings = true; //default: true
public const bool showHints = false; //default: false
```

Declare here what kind of messages you would like to receive in the console. Set them in the code.

3.

```
public bool useRecordingGUI = true; //default: true
public bool useReplayGUI = true; //default: true
```

Use this to set whether you want to display the graphical user interfaces that the user playing your game can record, stop, replay, rewind, speed up and speed down all by himself without you having to implement the replay managers API.

4.

```
public float recordingInterval = 0.05f;  //default: 0.05f
public const int maxSpeedSliderValue = 3; //default: 3
public const int minSpeedSliderValue = -3; //default: -3
```

These values have immediate influence on the recorders performance. *recordingInterval* is the rate by which the manager records your game. 0.05 means 20 frames per second (fps). So the manager will record 20 states per second per object. Receive the number of frames by calculating 1/recordingInterval. If this is too high as update rate (meaning *recordingInterval* is too low actually) you will notice it by the replay going slower than your recording (your game scene) actually went. If the value is too low as update rate(meaning you should actually lower *recordingInterval*) you will notice this by your replay being played choppy. Find the right value for your game by adjusting it. A good strategy is to try 0.04 (25 fps) and lower it until you are satisfied.

Because this variable also has **a lot** of influence on the amount of data which is being created when recording, please refer to chapter 5 of this document when experiencing problems while saving or loading replays (from hard disk).

The other values determine how much the game can be speeded up (*maxSpeedSliderValue*) and slowed down (*minSpeedSliderValue*) in replay mode.

5.

```
public List<GameObject> gameObjectsToRecord = new
List<GameObject>();
public List<string> componentsAndScriptsToKeepAtReplay = new
List<string>();
```

The first list is meant to help newcomers who are not comfortable to program yet. You can drag and drop your game objects in design time here to have them marked for recording in the real game.

The second can be of importance if you have important scripts on your game objects which you don't want them to lose even if the game objects are currently in replay mode. Normally all scripts except for some vital are being deleted in this mode, but you can type in the name of the

scripts and components here which you want to keep while a game object is in replay mode. For excluding *CharacterController* (not recommended) call

```
componentsAndScriptsToKeepAtReplay.add("CharacterController");
```

Don't worry, your original game objects won't be touched at all, the replay manager will not delete any scripts on them, just on their replay-clone counterparts.

The use of this feature is demonstrated in example scene 1 (included in the package).

6.

```
protected ViewMode currentMode = ViewMode.LIVE;
protected ActionMode currentAction = ActionMode.STOPPED
```

currentMode and *currentAction* are important values. currentMode can have two values: *ViewMode.LIVE* and *ViewMode.REPLAY* . *ViewMode.LIVE* is set when the real gameplay is going on. *ViewMode.REPLAY* is set when the plugin is playing a recording.

currentAction describes the pretty self explanatory states in which the recorder can be in.. his values can be *ActionMode.RECORD, ActionMode.PLAY, ActionMode.PAUSED* and *ActionMode.STOPPED*.

To request the values from outside use these two functions:

```
     if (EZReplayManager.get.getCurrentAction() ==
ActionMode.PLAY)
          //...your code here...
     }

     if (EZReplayManager.get.getCurrentMode() == ActionMode.LIVE)
          //...your code here...
     }
```

7.

```
     private int recorderPosition = 0;
```

This variable also has big impact on the component. It describes at what recorder position the component currently is. It can be requested via this function from the outside of the class:

```
     EZReplayManager.get.getCurrentPosition();
```

4.4 Extending the EZ Replay Manager

4.4.1 Extending which values are recorded

The extension is very generic, should work on all games, but of course it is also easily extendable: Until now only the information for position, rotation and whether particles are emitted are being saved. So if your game object does something fancy, like i.e. being destroyed in a long lasting explosion, which slowly evaporates it. You should extend the file *SavedState.cs* with the code of the value for evaporation.

First declare 2 instance variables

```
private bool evaporates;
private float evaporationValue;
```

The first (constructor) determines whether the evaporation takes place, the second, at what stage the evaporation is "right now". Let's say responsible for the evaporation is a script called *Evaporator*. Add in the constructor:

```
if(go.GetComponent<Evaporator>()) {
     this.evaporates = go.GetComponent<Evaporator>().evaporates;
     this.evaporationValue =
go.GetComponent<Evaporator>().evaporationValue;
}
```

So now the evaporation value is saved for every frame for every game object which has a component *Evaporator*. In these examples it has two values, which can be received and set: *evaporates* says whether the evaporation has been started, *evaporationValue* returns the value (for example 0f for no evaporation, and 1f for total destruction). To be able to see this in replay add the following line to your code (in some Start()-function).

```
componentsAndScriptsToKeepAtReplay.add("Evaporator");
```

The same can be done by editing this property in Unity3d Editor GUI on the **EZReplayManager** prefab.

Now back to *SavedState.cs*: In the function *synchronizeProperties* we determine what happens when the game is being played back to the user. Add the following:

```
if (this.evaporates)
     go.GetComponent<Evaporator>().evaporationValue =
this.evaporationValue;
else if ( go.GetComponent<Evaporator>() )
     go.GetComponent<Evaporator>().evaporates = false;
```

So if in this particular frame the evaporation was set to exist, we set the value like the one which was saved. If not we switch the evaporation off.

As of version 1.5 we have to further add code to a comparison function. Add this in the bottom of the body of *isDifferentTo(SavedState otherState)*:

```
if (!changed && (evaporates != otherState.evaporates ||
evaporationValue != otherState.evaporationValue))
     changed = true;
```

This is important to support the new recording method of this extension, to only record frames where a real change took place (keyframe method).

4.4.2 Extending with callbacks

As of version 1.53 you can now use callback functions in your own gameObjects to not have to overwrite the plugin classes unnecessarily (which a future update could again overwrite). To use them enable *sendCallbacks* on the **EZReplayManager**-prefab instantiated in your scene. You will see a new list appearing beneath that where you can configure which callback functions actually will be executed.

Just add one of the functions in the following list to one of your game objects. The callbacks are sent to all game objects in the scene so it does not matter which one you choose.

Function declaration	Executed when?
`void __EZR_live_prepare() { }`	Executed when still in replay mode just before the scene is prepared to switch to live scene again.
`void __EZR_live_ready() { }`	Executed when just switched to live scene again.
`void __EZR_replay_prepare() { }`	Executed when still in live mode just before the scene is prepared to switch to replay mode.
`void __EZR_replay_ready() { }`	Executed when just switched to replay mode.
`void __EZR_record() { }`	Executed when recording is being started.
`void __EZR_play() { }`	Executed when recording is being played.
`void __EZR_pause() { }`	Executed when replay is being paused.
`void __EZR_stop() { }`	Executed when stop is being called.

In order to save resources you can choose to just leave those functions in the list on the prefab which you actually need and use.

The use of callbacks is demonstrated in demo example 1 (included in the package) in file *DrawLines.cs*.

5 Known issues

1. System- and gameload: If you are recording a very heavy game scene or your system is very busy while recording, it is possible that the time delay between two frames is different on replaying. This can result in a situation where the replay has a different playing-length than the scene which was actually recorded. Due to these unpredictable variables as system- and gameload a replay in many cases is not of exactly the same length as the live scene which was recorded, although we took a lot of trouble canceling out visible differences.
2. **(PROBLEM NOT SEEN ANYMORE AS OF UNITY 4.2.2, SOLVED?)**

We noticed that saving files to hard disk and especially loading files from hard disk can become very heavy operations. The higher the amount of data, which has to be written to file, the higher the possibility of the occurrence of a Unity bug: If the amount of data to be saved to file exceeds approximately 100 MB and the amount of data to be loaded from file exceeds approximately 50 MB it is not unlikely that you will experience an application crash accompanied with the error message: *Fatal error in gc: Too many heap sections*. This is not a bug of the *EZ Replay Manager*, actually maybe not even a bug in Unity, but in Mono or even more underlying application layer (like gc). Please refer to the following urls on this bug:

a. http://answers.unity3d.com/questions/161653/memory-leak-adventures-in-editor-34-lot-of-instant.html
b. http://answers.unity3d.com/questions/137224/laggy-gameplay-and-occasional-crashes.html
c. http://kerbalspaceprogram.com/forum/index.php?topic=2631.0
d. http://www.inkscapeforum.com/viewtopic.php?f=22&t=3336
e. http://forum.unity3d.com/threads/58746-Fatal-error-in-gc-Too-many-heap-sections
f. http://answers.unity3d.com/questions/184680/memory-leak-gallore.html
g. http://kerbalspaceprogram.com/forum/index.php?topic=6220.0

The bug seems to be well known and hopefully the Unity developers will address this issue very soon (i.e. by updating the technical platform Unity runs on). As far as we now know the error occurs when deserializing the data from a saved file, because not the amount of data in the memory is the problem but the amount of data when deserializing. On the other hand deserializing is 1 command, and there is little chance of writing this command somehow in a way that circumvents this bug.

Don't get it the wrong way: The amount of data you have to save or load to receive is bug actually enormous: You have to have tenth of objects which are permanently moving "preferably" at a high sampling rate (recording interval) and are being recorded for quite a long time. So in theory it is possible to receive this bug when endlessly spawning

objects, permanently moving them and recording with a high sample rate and for a long time. But in practical situations you will likely not experience it, if you yourself take care of performance and memory only a little bit. A good tip for a very crowded scene i.e. is to lower the frames per second (fps) to be recorded. Also when involving animations a frame rate of 12.5 fps (recording interval of 0.08) gives you absolutely fluent replays and does not damage the user experience of it, but actually cuts the amount of data to be written to file in half (in comparison to 25 fps = recording interval of 0.04). By data **only** the data is meant which changes from frame to frame (on default that is position, rotation and if a gameobject is emitting particles, please see chapter 4.4 for easily add properties to be recorded). Meshes and other data are not included. The size of the mesh of your gameobject does not matter for the size of the file which is being written to hard disk.

Because of it being a bug not caused by this extension but rather by one or a combination of different application layers beneath it, it is very possible that it will be fixed in the future just by updating Unity! Nevertheless we will keep a close eye on this and try to give you the best advice on lowering your data which is to be recorded and continue to lower the overall data which is being saved to file. Because of this bug we introduced a new recording method in version 1.5: Rather than saving everything in any frame we are now using a keyframe system to only record actual changes.

3. Saving and loading big amounts of data can take several seconds.
4. Webplayer builds cannot save to file because they are run in a browser sandbox.
5. Iphone and Windows 8 store builds cannot handle serialization yet, so on these platforms replays cannot be saved/loaded from file.
6. **(PROBLEM PROBABLY SOLVED AS OF VERSION 1.53, read chapter 4.1 about `ChildIdentificationMode`)**
   Swapping children of gameObjects or changing the order of them can lead to problems when replaying.
7. **(PROBLEM NOT SEEN ANYMORE AS OF VERSION 1.53, SOLVED?)**

Using `OnDestroy()` on gameObjects which are to be recorded to destroy the whole gameObject once called (when the script gets removed from the gameObject) will lead to problems when replaying.

8. **(PROBLEM NOT SEEN ANYMORE AS OF VERSION 1.53)**
   `EZReplayManager` class uses `SetActive` to reactivate the real gameObject once replay is done. In a future version the system will remember which chrildren were active and which weren't before replay. Stay tuned!
9. When you selected to precache gameobjects it is possible at the first run of a scene you get the message “Error moving file”. That is not to worry. Just click “Try again”. This will only show at first precache of a gameobject. Later this will not occur anymore.
10. If you encounter the following error: Destroying components immediately is not permitted during physics trigger/contact or animation event callbacks. You must use Destroy Instead.

Thanks to ninjaboyjohn the problem can be solved: http://forum.unity3d.com/threads/ez-replay-manager.92854/page-16#post-1654067

If you experience undocumented errors or know a possible solution to one of these issues please let us know. Your input can make a very valuable difference.

Need support? Get it for free in this thread: http://forum.unity3d.com/threads/92854-EZ-Replay-Manager