Capture And Repay – CAR JS

CARJS project aims to provide remote Capture and Replay facilities in order to help the developers in debugging, testing and thereby make the web application more reliable. Few important points to be noted here

- Logs must be maintained exclusively on the user system.
- Instrumenting at the application level would be recommended.
- Both deterministic and non-deterministic actions need to captured and replayed.
- A controller to the control the replay (in sequence) - which means the developer controls the replay and it isn't an automatic replay of the episode.
- Maintaining events with identifiers and thereby providing a dashboard for developers to "choose and replay" an execution - which means the developers, can choose specific Log episode and replay it alone.
- There needs to be a concept of Sandbox so that the requests from the replay are directed towards a sandbox server and not the original setup.

Relevant Papers:
Mugshot - Deterministic capture and replay for Java script applications

- Standard JavaScript implementation – Event based model.
- Covers DOM Events, Timing Events and Non-deterministic events.
- More details http://research.microsoft.com/pubs/120937/mugshot.pdf

CARJS 01 – (Function Based Model)

The implementation and explanations for this model can be found on the below link.
https://github.com/indianburger/carjs

Function Based Model

- The JavaScript code is instrumented –> every named fn () is instrumented with the logging logic inserted & every anonymous fn is transformed into a named function and logging logic is embedded as well.
- Instrumentation process implemented in Java - input JS file / Output instrumented JS file.
- Logs stored on the server using Ajax requests on the capture phase capturing the execution.
- Replay fetches these logs and replays the execution.

Results for CARJS 01

Application	Events Captured	Events Replayed	Firefox	Chrome	IE
Tetris	Yes	Yes	Yes	NO – abruptly stops.	NO
Wave	Yes	Yes	Yes	Yes	Script Errors
Battleship	Not Working	No	No	No	No
Domtris	Yes	Yes	Yes	No	Script Errors

Events handled

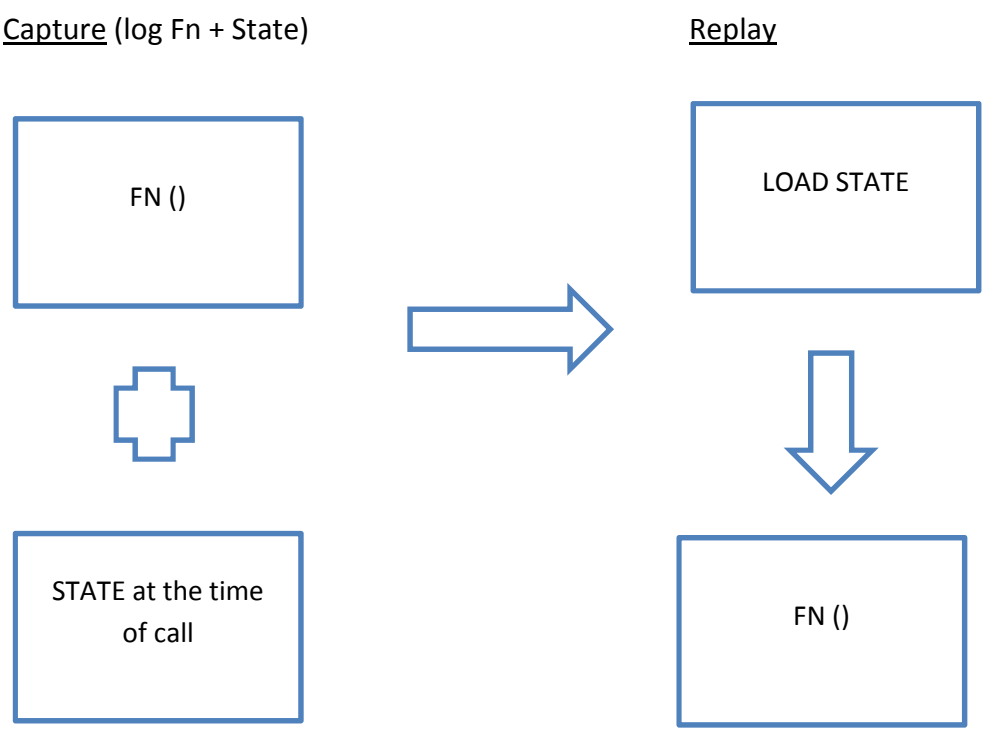
Events	Captured	Replayed
DOM Events (All Js Function Calls)	Yes	Yes
SetTimeOut	Yes	Yes
setTimeInterval	Yes	Yes
Random	Yes	Yes
I Frames	No	No
Ajax	No	No
Date()	No	No

Problems with function based CARJS

- This would require instrumenting all the Js functions beforehand to capture all the elements being used Or Doing these at the capture time will cause an overhead in processing.
- Logs would contain something similar to Fn Name with Elements used and parameters (State). Finding the state is the real difficulty.
- State ➔ List of all elements + variables used by the function.
- Given the closure property, it's really difficult to find all the variables visible to the function.
- And instrumenting an anonymous function into a named function might make some variables no more accessible inside other functions.

Simple Representation of the Model

For each Fn call {



}

Now,

For each elm 'E' used in the fn () as a parameter or inside in a statement.

Get DOM Tree for "E".
Get Attributes for "E".
Get Data in E.
Get HTML of E.

How to find E in these cases (some)?
Consider the following lines are being used inside the function fn or passed as a parameter

$(".coloredRed") - *would return a list of div elms...*

$("#div").children () / siblings - the child tree of an elm may be used

```
If ($("#div").hasClass("registered")) reg! =reg;
$.post (ÚRL, "courseID: 1, regStatus: reg",
        $("#course").addClass("selected"); // STATE INFO SHOULD NOT BE MAINTAINED
);
```

It's really complex to instrument the functions to have all the parameter info as it is difficult to find the variables accessible inside a particular function - from the global scope, outer functions and from other JavaScript's as well.
And an instrumented version would cause a lot of JavaScript code change to the existing web application and there is this risk of breaking the application.

More Papers:

WARR Recording (instrumentation done at the browser level – Webkit **Chrome** Driver)
Capture:

- Log all the keystrokes, mouse clicks.
- Catch all the handleKeySroke / handleMouseClick Event Listeners.
- Log them as Event on Element model.

Replay:

- Similar to logging, dispatch them one by one.
- Special version of the browser for replay as Events triggering are read only so permissions required to trigger them.

Limitations:

- Pop – ups, logs all information including sensitive ones.
- Developer system with edited web kit for replay.
- Browser limited to Chrome only.

TimeLapse (**Mac Safari**)

- http://homes.cs.washington.edu/~burg/projects/timelapse/
- integrates record/replay into the browser – **Mac Safari**
- Written in C++ as it involves modifying the Webkit.
- Unlikely to port this into Gecko/Firefox.
- More information on the project page.

HP True Client

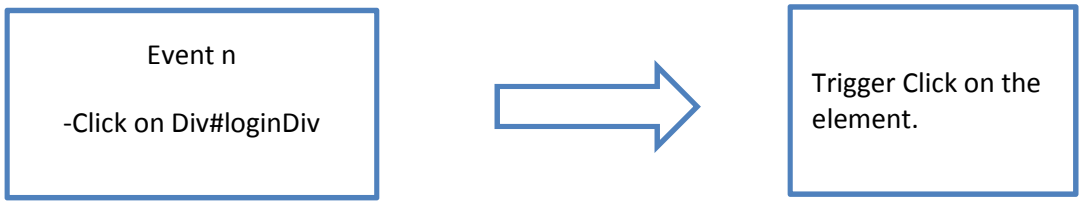
- Commercial tool for capture and replaying Ajaxified applications.
- http://www.youtube.com/watch?v=IuBxSbuNGDY

CARJS 02 (Event Based Model):

Simple Representation of the Model

For each event (deterministic or non-deterministic) {

Capture (Event & DOM element) Replay (get the Event and Trigger)



}

Events Handled

Events	Capture	Replay
Mouse		
	Click	**Click**
	Double Click	
	Hover	
Keyboard	**Key press**	**Key press**
	key down	**Key down**
	key up	**Key up**
Date()	**Log**	**Use the logs for replay**
Math. Random()	**Log**	**Use the logs for replay**
SetTimeOut / setTimeInterval (Timing events)	**Instrument the fn () on the fly and log when called.**	**Use the logs and invoke the corresponding timing event.**
Ajax	**Works!**	A separate version of Js used in replay version where the sandbox server can be specified

Design and Implementation

1) Capture.js

Mouse Clicks and Keyboard events:

Register new listeners for this on the DOM's top most element – document or the window. Write the logging logic inside these listeners to log the event details and the element on which this was invoked.

Timing Events SetTimeOut and SetTimeInterval:

Override the window.setTimeOut insert the logging logic here. Instrument the fn which is to be called to contain the logging piece of logic. Call the old SetTimeOut with the new instrumented function.

Override the window. SetTimeInterval using similar logic.

Math. Random () and Date ()

Override the windows respective methods, logging the values generated on these calls.

2) Replay.js

Mouse Clicks and Keyboard events:

For each log trigger the respective event on the mentioned element. The element's xpath is available in the log. Find the element from its **xpath** and invoke the event on the Dom element.

Timing Events SetTimeOut and SetTimeInterval:

Override the window's respective methods to push all the invocations onto a stack. Once a timing event has been found on the log find its respective match from the stack using the fn ptr and invoke a SetTimeOut with this fn ptr and delay set to **0 for immediate execution.** The delay value has no meaning on the replay part.

Math. Random () and Date ()

Override the windows respective methods, return values from the logs on these calls.

Tools used

- JQuery.
- Local Storage plugin jStorage.
- Xpath Plugin from Firefox model.

Logging

- Current implementation uses the local storage of HTML for logging the events in a JSON format for each coding and decoding.
- JStorage may be used for more sophisticated store and retrieve.
- Each episode creates an identifier which serves as the key for the entire episode.
- This key can be used for retrieving a specific episode and replaying it.

- By default the last recording is chosen for replay.

More on Local Storage

HTML5 options available – localStorage, WebSql & indexedDB.

S no	localStorage	IndexedDB	WebSql
1	Name/value pairs	Key-value with support for indexes.	Similar to relational Sql
2	String – String value	This is an experimental technology.	All CRUD operations available
3	Max Storage 5MB	No limit	No Limit
4	Supported by all major Browsers	Supported by Chrome, IE and Firefox.	Not supported by Firefox and IE. They don't have plans to in the near future.
5	Use localStorage.clear () to flush the space if fully used.		

How To?

Two instrumented applications **Tetris** and **MyMiniTodo** are available here https://github.com/lokresh88/CaR-JS

- Include the Capture.js and other dependent libraries into the layout page of the application.
- Similarly include Replay.js and other dependent libraries into the duplicated layout page of the application which is to be used for replay.
- Dependent libraries include JQuery, Json, and JStorage etc.
- A separate folder with all these Js files can be found inside the above applications.
- Including the scripts must be done on the layout page and make sure it happens first.
- Parallel loading of JavaScript's need to be avoided.
- Enhancements/Fixes need to be performed on only these two capture.js and replay.js alone since.

Setting up a Server on your machine for the web applications

WAMP Server can be setup and your application folders can be moved into the www/ path in the wamp. WAMP can be downloaded and installed from http://www.wampserver.com/en/

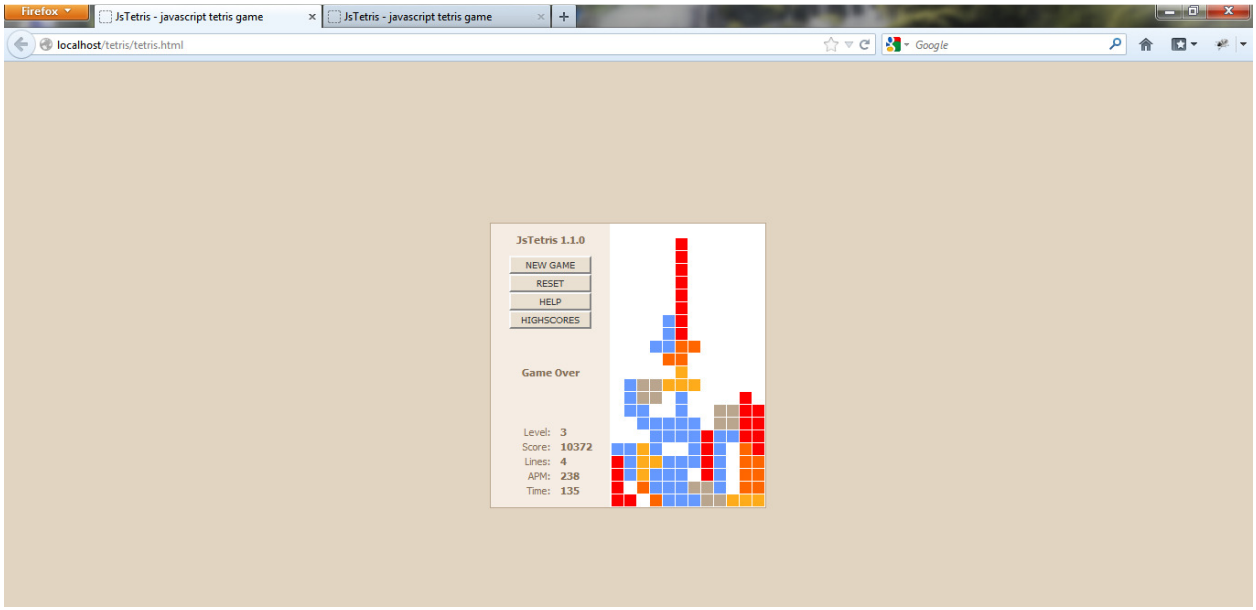
Once setup the application can be accessed through http://localhost/tetris/tetris.html.

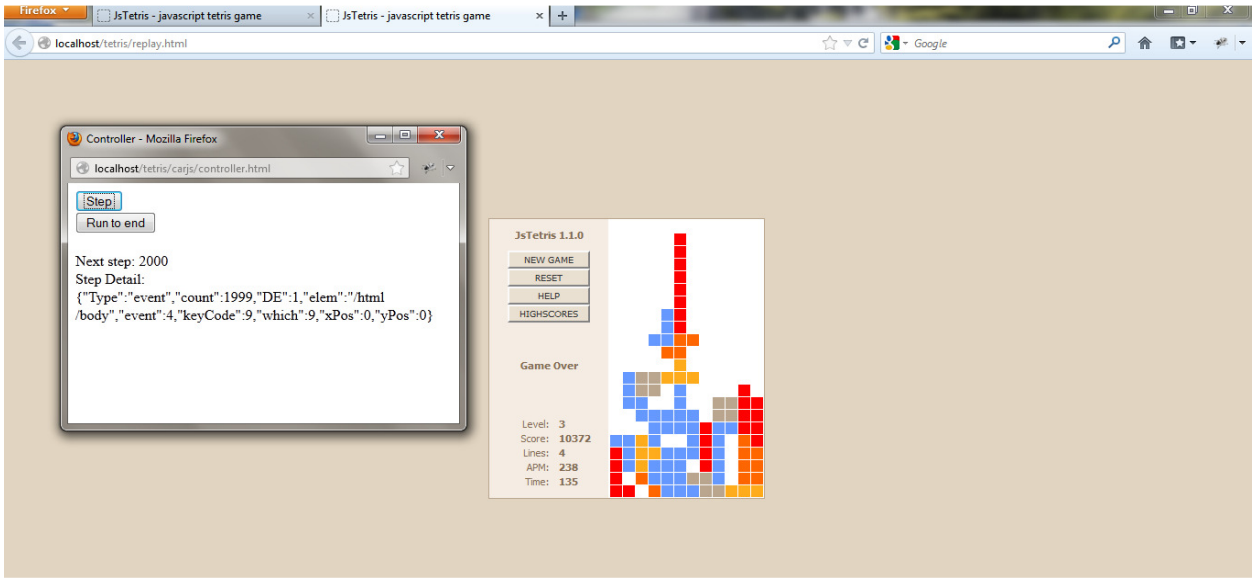
The above setup holds good for PHP-MySQL or static web applications. For J2EE kind Eclipse can be used setup the Web server. Create a new web application and import all the stuff.

Sample Applications (tested in Chrome and Firefox)
Available at https://github.com/lokresh88/CaR-JS

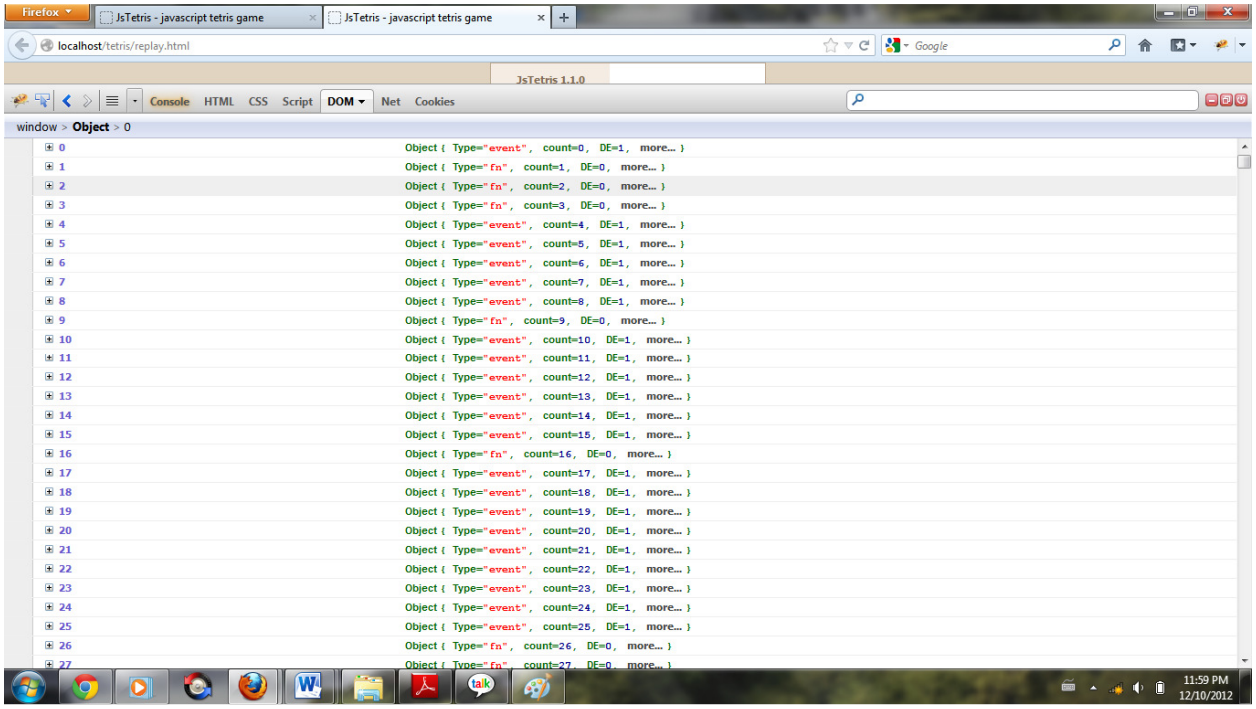
Tetris – static web application with mix of DOM Events + Non-Determinism + Timing Events.
Capture landing page: http://localhost/tetris/tetris.html
Replay landing page: http://localhost/tetris/replay.html

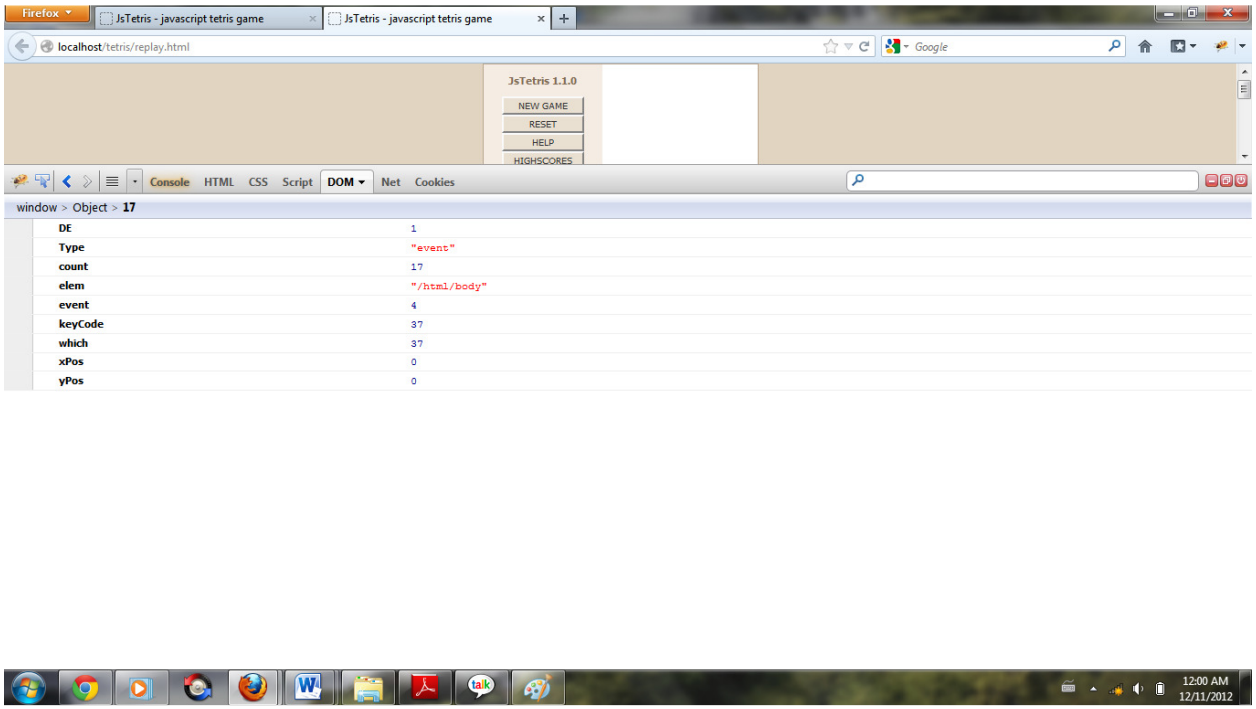
Snapshots of Capture and Replay





Logs snap of an Episode





MyMiniTodo – dynamic web application with has more of DOM Events + Ajax.
Capture landing page: http://localhost/mytinytodo/index.php
Replay landing page: http://localhost/mytinytodo/index-replay.php

The capture replay logic is available under /carjs folders respectively.

Next Steps

The set of actions that need to be handled

1. I Frames, Popups, Dialogs – separate listeners can be registered for I Frames – accessing I Frames from a different window might cause Security errors in Chrome. The "Cross-Domain" fix needs to be adapted here.
2. Dropdown selection + Hover and other minor events for mouse events.
3. Sandbox server for the replay. The replay script is available as a separate Js file. Something like $.Ajax Start () could be written here and a generic redirection can be applied here instead of changing the applications script.
4. Improve the storage of logs. LocalStorage provides us 5MB which is less. IndexedDB seems to be good option but it's experimental. WebSql is not supported by all the browsers!
5. Compression of logs by GZipping JSON – the operations take a bit of time.
6. Try it out on sophisticated applications.
7. Develop a mechanism for remote replay – dashboard for developers.

**Using JQuery provides more power as its compatible across almost all browsers.

Difficulties

- In some applications events may be stopped from bubbling to the document which makes capturing of those events difficult. Instead of adding capture listeners to the document/window we could dynamically add/remove these listeners for all the elements on the html page.
- Log space is less and when there is a high processing with respect to logs the performance of the application may be hindered.

References / Useful links

1. http://research.microsoft.com/pubs/120937/mugshot.pdf
2. http://homes.cs.washington.edu/~burg/projects/timelapse/research.html
3. https://developer.mozilla.org/en-US/docs/Gecko
4. http://www.webkit.org/
5. https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Global_Objects/Function
6. http://jquery.com/
7. http://www.wampserver.com/en/
8. http://www.html5rocks.com/en/features/storage
9. http://www.jstorage.info/