HTML5 Research Journal

by Ciarán McCann

Introduction

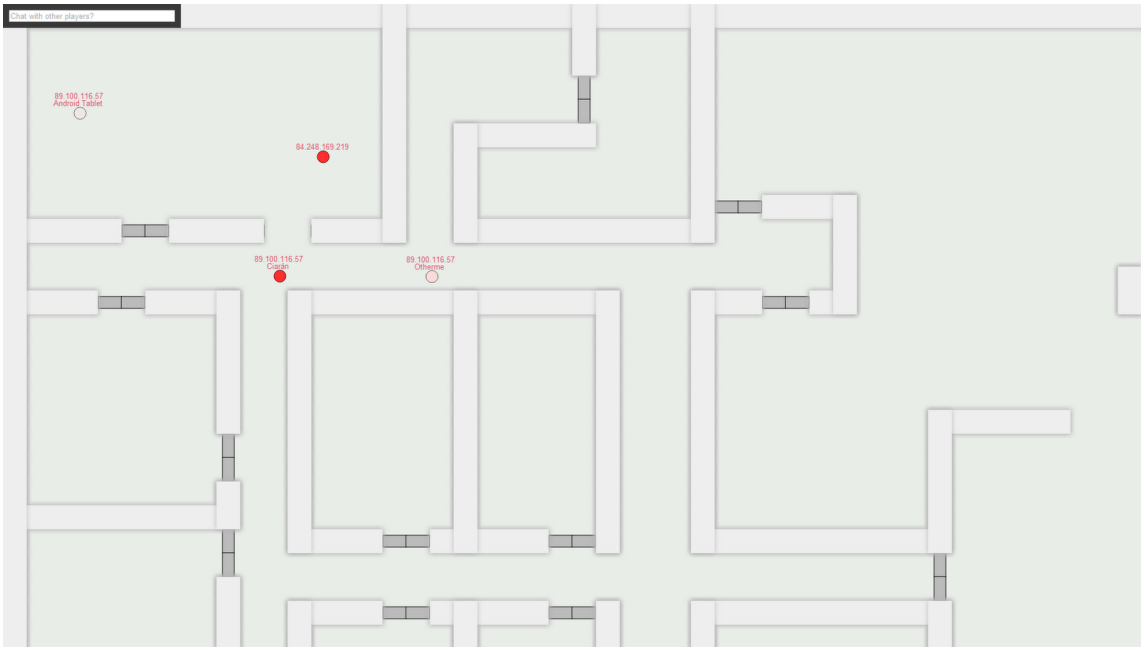
For this project I have taken on a focus of empowered learning and exploration of unknown technologies in the HTML5 area though not limited too it. So far the exploration has involved the following technologies Node.js, Web-sockets, Socket.io, Amazon EC2 Cloud Computing, Linux, SSH, BASH, Vim, WebWorkers.

The initial project brief which I wrote turned out to be to simple, the chat client. So I decide to make a multilayer networked game instead to really get to grips with Websockets, Node.js, socket.io and network programming in general. Below are some posts on the experiences on developing this game.

Demo

The game demo is available here http://107.21.108.104/NodeProject/index.html

Screenshot



In the screenshot above, the red and white circles are open client connections. (Players) These players can move around the environment and interact with other players.

Setting up live Node.js server

Working from local-host is quite limiting as it difficult to test will your code actually work in a deployment situation. For example simulating latency between various different connections to the server is difficult since both the clients and server are running from the same machine. So I needed a deployment server and an elastic IP which I could share with others.

So I have explored the various options of getting a live Node.js server up and running. Initial I though I could run it off my shared hosting server, that I have with a Carlow based company, Blacknight. Though I don't have SSH access to the server because its shared server. Secure Shell (SSH) is a protocol which allows the secure execution of commands from one computer to another. It gives the sudo user full access to the machine and allows them to install software, such as Node.js.

Amazon EC2

This is amazons cloud computing solution, they offer a small free package, so I signed up for it.
I was able to create a virtual server and install any OS I wished on it. As I am an Ubuntu Linux user myself I installed Ubuntu server edition though SSH and went about installing the various compilers and packages I needed to compile and build Node.js on the server.

Learning out comes

I learnt quite a bit from this process and it took me quite some time to get it all setup. I had never worked with cloud computing before so it was a fun and very educational experience. All this process was though SSH, so naturally my UNIX command line skills and knowledge have greatly improved. Heck I even was using Vim to write most of my code for a day or two. My understanding of how services and ports work in Linux was greatly improved. I also improved my Bash skills as I wrote some helper scripts for deploying code to the server. I have a one line command now that, pulls the latest build of my code from the git repository and starts the node websocket server.

Problem: *"Syncing JavaScript Objects across clients"*

In the game I have been developing, the game requires that the player object for each client be sent though web-sockets to the Node.js server and broadcast to the other clients. So I went about emitting an event from the client page once the page loaded, which would send the player object with the event to the server. The players object was of type Person, which was a javascript prototype object which I had made. Once this object reached the server it had lost a vital piece of information, its type. It no longer knew that it was a object of type Person, it was now simply a base javascript object type in the structure of a Person, but with out the Persons methods etc. Not only that but member variables of the Person object that were themselves complex types suffered the same fate. How did this amnesia come about?

How the problem accrues?

Well it comes about by the process of data transfer, the objects are converted to JSON to be sent over the wire, all seems fine, but for one thing. When an object type is JSON.stringifyed its members are stored as a base javascript object, no meta-data is saved about its type by default.

So when that object is passed around to the clients, they treat it like a normal Person object and call a dot draw function on it, it throws an erroring saying a draw function is not defined for the object. So how do we convert this object back to a Person and further still how do we do it in a nice generic way that doesn't require boiler plate code to be written for each new member variable of the class?

Approaches/Solutions

Approach I wanted

Since I have been a C-style programmer since I was 15 and working mostly in C/C++ based languages my solution was to make the wild and unruly JavaScript more like C++, in its class structures. I wanted a solution that looked like a copy constructor in C++. JavaScript doesn't have copy construction. So my solution would be to programmatically create default copy constructors for any class I make. So that it would look like the approach below.

```
var x = new MyClass({m1:10,m2:10,m3:10,m4:{h:5} });
var copyOfx = new MyClass(x);
```

Other Approaches

I researched various solutions on the Internet, but none could seem to achieve what I wanted. Most involved boiler plate code. Such as the below sample.

```
var x = new MyClass({m1:10,m2:10,m3:10,m4:{h:5} });

var copyOfx = new MyClass();
copyOfx.m1 = x.m1;
copyOfx.m2 = x.m2;
copyOfx.m3 = x.m3;
copyOfx.m4 = new ComplexType();
copyOfx.m4.h = x.m4.h;
```

Solution

After about 2 or 3 hours messing around with various ways to go about it with as little boiler plate code as possible, I got on to a winner. Since JavaScript objects are inherently just key/value pairs an object can be looped over. Also the Person object that had amnesia, still was structured like a Person, it just didn't function like a Person. So using this relationship I developed gernic recessive copy function. Below is a simplified venison of my copy function, without some defensive programming checks removed, so its easier to explain. Its also an arbitrary example in that a Vector2D only has two members and no member functions, though its only used to illustrate the point.

```
1  //Allows for the copying of Object types into their proper types, used for copy constructer
2  //for objects that are sent over the network. I have intergrated this function, into
3  // the constructor of the Person object so it acts like C-style copy construction
4  // WARNING: This creates a deep copy, so reference are not preserved
5  std.copy = function(newObject, oldObject) {
6
7      for( member in oldObject )
8      {
9          // if the member is itself an object, then we most also call copy on that
10         if( typeof(oldObject[member]) == "object" ){
11                 newObject[member] = std.copy(newObject[member],oldObject[member])
12         }else{
13             // if its a primative member just assign it
14             newObject[member] = oldObject[member];
15         }
16     }
17
18     return newObject;
19 };
20
21 //class
22 function Vector2D(data){
23
24         this.x = 0;
25         this.y = 0;
26
27         std.copy(this,data);
28 }
29
30 //usage
31 var vec1 = new Vector2D({x:10,y:10});
32 var vec1Copy = new Vector2D(vec1);
```

The method is simple but extremely powerful and useful in that it handles what would normally result in boilerplate code in a clean and silent manor.

Explanation

The function takes two objects, the first one is a basically the template or the map, which the oldObjects members get mapped to. We see on line 27 the call to copy takes the this pointer as the first parameter which provides the default constructed Vector2D object. We see on line 32 vec1 object is passed as the second parameter, ordinarily in JavaScript this wouldn't work, though thanks to the way this object is constructed and my copy method this functionality works perfectly and the programmer doesn't need to worry about it. So the copy method itself loops though the member variables of the object, checks it they themselves are objects and if so recursively calls copy and if not simple assigns its value to the new object.

The key to my solution is the fact that JavaScript objects are iterable, in that you can loop over both an objects member variables and member functions like any collection, as JavaScript objects are inherently key/value pairs. For more information and detail on how the code is used in practice check out https://github.com/CiaranMcCann/NodeProject

Evaluation

My solution work perfectly and exactly the way I wanted it to work, with no need to compromise.
The solution allows for copy constructor functionality for any Javascript class structure, just by adding one line function call to the constructor. I'm quite pleased with how clean my solution was. Developing the solution was also a very culturing experience, as I worked with the syntax and semantics of one language to mutate it to another language.

Rendering

Rendering using HTML5 canvas is slow, especially when not hardware accelerated. So one most do all they can to reduce rendering overhead. When I was developing the render system in the Flax HTML5 Game Engine I did extensive research into ways of reducing canvas overheads and ways to render objects more intelligently.

Some of the easiest optimizes are, just use canvas properly!! Its an OpenGL like implementation of immediate mode drawing, with a state machine to ogment the drawing. So first off

1. **Batch draw calls**
2. **Reduce changes to the state machine**
3. **Use integer based coordinates - Removes need for anti-aliasing**
4. **Don't use shadowBlur**

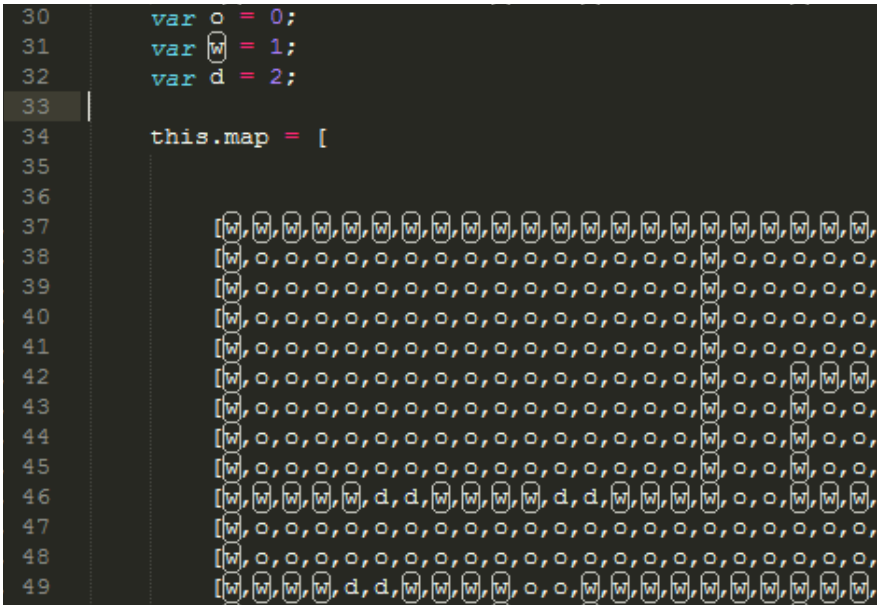
All the above alone will give a massive performance boost, more so the first too then anything else, though seriously don't use shadowBlur! Like any rendering system rendering a noise blur pattern is expensive, though even more so for HTML5 canvas element. There are a lot more canvas optimizations I use, though they are the main ones.

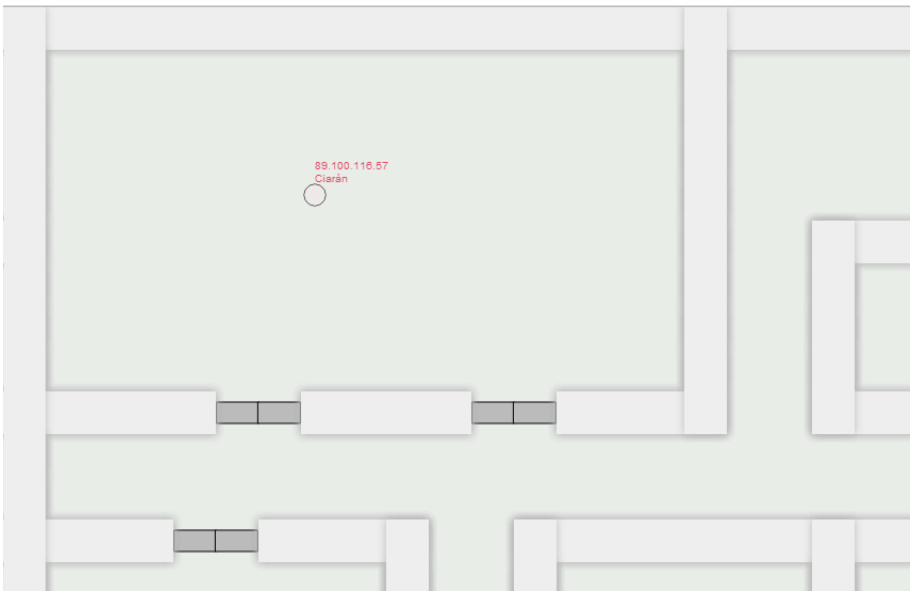
Smart-rendering

Now onto smart rendering, canvas will render what ever you give it. So you have to be smart about what you give it and how you determine what you give it. I'm always one for infinite worlds, so a rendering algorithm that can determine what to draw at said camera position in O (c) is a most! For example with Flax, I implemented a direct indexing with smooth interpolation between indexes for the titled map. So that map could be 1 million tiles by 1 million titles and not have a performance hit as direct indexing works in O (c) constant time. I have become a large fan of direct indexing and use it as much as i can.

Pre-rendering/Level Generation

In this game I'm working on I didn't need the level environment to change, so instead of direct indexing I used a pre-rendering technique. My level is rendered to a canvas by reading in an array of characters, 0 being free space and 1 being a wall segment, The algorithm then scans down the rows and columns of the array to find consecutive wall segment's both vertically and horizontal. These consecutive wall segments are draw as one long wall, giving a nice effect along with the shadowblur. The below pictures illustrate how it works. The first image is the ascii array and the second the image the rendered walk segments.





Once my level was rendered to an off screen canvas I used the toDataUrl method from the Canvas API to get the Canvas image data as a base64 encoded png and layered it over my main canvas, using CSS to control the positioning of it, allowing it to move and give the impression of a camera following the character.. Job done, level draw once and not every frame, magic!

Doors

The doors in the level are also generated from the ascii array, the array is scanned for two conseciative “d” characters vertically and horizontally. Once one is found, a Door object is constructed with the position data. Then the next one is constructed and the first door is assigned as the partner door, so when one opens the other opens.

Problems

This method works perfectly on almost any browser that supports HTML5 Canvas tag and API. Though I have noticed on the mobile verison of browsers, toDataUrl isn’t currently supported. So for the Android phones/tablets on lower then 2.2 the level will not appear and on iOS5 it doesn’t seem to work. There is not a major disadvantage as really this functionality is only for development and once the game is deployed permanently the image that's generated can be saved on the server and not generated every-time the game is loaded.

Problem: *"Networking/Physics variable frame-rate"*

This is the age old problem, of networking, variable frame-rate. So lets set the sence. we have a server with multiple connected clients, each client has a game loop. In the game loop we have rendering, logic update and physics calculations. If one players game loop runs slower then another, that means their physics calculations will run slow, so that player will move slower in comparison to the other player. Rendering is generally the one that cause the slow down, where one player may be playing it full-screen on a low end laptop and another with only half screen size. The smaller resolution, means less rendering and thus the game loop runs at the set speed and doesn't drop. The below video will illustrate the problem better.

Approaches

There are many different approaches to helping to solve and relive this problem.

Threads
One approach is to separate the render loop and the logic loop. In most PC games this is done by creating a new thread and having the physics run at a solid rate. So for a HTML5 game, we do have the ability to open whats called a WebWorker and run code asynchronously in a new thread. Somes problems with this approve is that not all web browsers support WebWorkers.

Sever-side Physics
Another approach would be to move the logic to the server, as the server holds current information about all clients. So updating each users physics logic on the server and updating their client side objects. This puts the computing load on the server instead of each client, so gives all clients an equal playing field. Both requires a lot of game logic and information to move to the server, and could case visual problems with players walking into walls.

Fixed time-interval
This approach is more suited to native PC games or console games. A base line frame-rate is selected, that most systems will be able to manage. Though with HTML5 games, they operate on a variety of devices ranging from mobile to PC's. Though this normally requires scaling back on features to lower the performance hit.

Solution

As I'm new to network programming I decided to cut back on cost of the game loop and set a fixed time-interval. So I needed to reduce as much rendering cost on the client side. I documented how I went about this in the rendering section of this document.

Evaluation

The solution has worked quite well for the moment, though I feel as I add more features to the game this will become a problem again. I hope to experiment with using WebWorkers to separate the render loop from the logic and physics loop.

Collision detection

Collision detection always tends to be expensive, in games were there are many collide-able objects, some static and some dynamic. Over the last few years I have used various methods of collision checking and collision optimization techniques. From basic box collision, polygon, radius, various bounding volume solutions, quadtrees, etc. So I decided to try something I had been meaning to experiment with for a while. A direct indexing collision system for static objects and relatively static items in the enviorment, eg the walls and doors in the level. So I took what I had used for smart-rendering and applied it for collision checking.

So for a collision check between a player and a wall segment, the players position is divided by the width of each tile. Then the Math.floor() function is used to cut off the floating point of the number to yield an integer, which can be then used to index into the level array, read weather the tile is free or filled with a wall. If there is a wall segment don't allow the player to move into that region, so some kind of collision response is need.
Though this isn't so simple when their is physic and general gradient movement envoled.

Problems

The method is extremely efficient and only requires 4 collision checks, with no look-up time to find what should be checked. Though as noticed in the demo, the collision can in some spots be a bit stickily, in that its not a smooth gradual collision, though this is to be expected with any direct indexing method.