LITERATURE STUDY

Game Engines On The Web: Past, Present & Future

Supervisor: Professor Anton Eliens

Leonidas Diamantis
VU University 2013

Preface

I would like to start this report with a few words about the topic of this study and the actual process of me defining my research scope, investigating the resources, filtering out any inaccurate or irrelevant information and finally conducting the present document.

When I was presented with the original topic of the study, namely “Game Engines On The Web”, I knew it was going to be a challenge. Firstly because the topic itself is fairly wide (something unusual in general scientific research, where topics are being overly quantized and sometimes over-saturated down to their - possibly unimportant - details, often with unpleasant results), and it left a lot of space for important decisions on my side. The first big decision I had to make was how to actually approach the topic, with regards to the research I would perform. I decided to approach it first in a somewhat historical way, following a timeline over which I illustrate the evolution of Web-based gaming in the components which assemble it, leading to the second phase of my approach which deals with the nature of those technological components. Specifically, I tried to analyze their behavior from a technical perspective, how they communicate with each other to produce the final result, how they evolved etc.

The second big challenge of this study was the resources. It is a fact that a research work is as good as the resources it cites, references and holds on to in order to make its conclusions. In my case, since the topic was so wide, it was particularly hard to get into trustworthy resources with a good scholarly foundations, mostly because many of the technologies that I am studying are too new or sometimes not given enough attention from the scientific community (which I strongly believe should not be the case). I thus had to focus my attention on different kinds of resources on top of any scholarly articles I had found (e.g websites, open-source technical blogs, developer’s books etc) in order to gather a good amount of material for a complete and spherical research. And I am glad that I had my supervisor’s trust and encouragement to do so, this meant a lot to me. Finally, since I happen to have some extensive personal knowledge in the Web platforms, I sometimes used my personal reflection or insight to enrich my findings with some (hopefully) original thoughts.

I hope my report is pleasant for the reader and adds a bit of contribution to the general topic and the works of the community.

Leonidas Diamantis

Introduction

This report illustrates, describes and explains the findings of my research through different literature sources with regards to the topic “Game Engines On The Web”. Given the fact that the topic itself is quite wide (and the needs of this research course require that one should not go neither too deep nor too wide on the topic), I decided to take a somewhat historical approach on the topic (hence the “Past, Present & Future”). So, what follows in this document is an overview of game engines and technological (software-oriented as well as hardware-oriented whenever possible) stacks which in their times became the building blocks of Web-based games, using the browser as their output mediator instead of being stand-alone applications.

The Past

Gaming has always been once of the primary uses of a personal computer, since their birth in the early 1980s, with a lot of intelligence and technology being put together to transform games and gaming experience to the magnificent virtual gaming worlds we experience today. A similar situation was the Internet and the Web, which brought a fresh aura in the computing era in which it was created, with possibilities beyond the imagination of their creators. Quite early on, games started being created using the Internet and the Web as their platform, of course in a primitive format of I/O and interaction with the players. But then again, it all had to start from somewhere.

In this section, we will take a look on how Web-based gaming used to look like, with regards to the technologies which were used to develop games and what type of interaction was possible between player and game in the Web.

Text-Based Adventures

As it has been with all first-born, primitive (pioneering nonetheless) games, the first generations of browser games were not too impressive, at least for the eyes of the player. They were based on - as most of the games which appeared first on their respective platforms had - non-existent graphics, poor interaction & control, and a rich back story. It might sound awfully romantic, but the real graphics engine of the first online games was the players’ imagination.

However, the success for many of these games was not at all small. In fact, there are games like Travian and Kings Of Chaos which are still being plaid, have huge fan bases

and communities of gamers to support them and they even moved forward, being enhanced so they can catch up with the contemporary technologies of Web-based game development.

From the technological perspective, such a game would consist of plain HTML markup and simple CSS styling on its front-end, maybe with some additional scripting enhancements (e.g simple JavaScript computations). All the logic of the game would be implemented on its back-end, where all the computations, user input evaluations and game output preparations would take place. The back-end of such a game would have most probably been build in a solid server-side language, like PHP or JSP, and the HTML part of the front-end would be generated and manipulated server-side before being rendered rendered the players' browser. In short, most of the building blocks of a Web-based game back then were located in the back-end.

The user's interaction with such games was obviously minimal. Many games did not even make use of both the keyboard & mouse for user input, and the player would most often have to submit some sort of HTML form in order to continue to the next steps of the game and move forward in its plot. The use of the keyboard and mouse as input methods in such games, had the input devices behave as they would in their natural use (the keyboard would just be used for the player to type characters in the input fields of the game and the mouse would be used for performing mostly offline clicks in the game's interface, perhaps to change the focus from one input field to another and finally click the element that would lead the user to the next step of the game). For the game to advance to the next step in the plot, it was required that the browser of the user would have to refresh their page, so that all the calculations - based on the logic of the game - would be performed in the back-end. Of course, multiplayer (at least in a - as much as possible - real-time context) was not an option. Often, in order to introduce some notion of real-time, the gameplay would require the user to wait a certain amount of actual (human) time before they can perform their next move. This was usually a gameplay characteristic of Web-based strategy games.

As it is natural with computer science to evolve, this scenery would not remain as such for long. When the communities began to get bigger and scale throughout the world and new technologies started emerging, the realm of Web-based gaming started evolving as well.

Java Applets & Embedded Code

A Java applet is a small application written in Java and delivered to users in the form of bytecode. The user launches the Java applet from a web page and it is then executed within a Java Virtual Machine (JVM) in a process separate from the web browser itself. Applets are very high performant pieces of software, because the JVM takes care of their execution, and this fact opened up the Web-based games world with new possibilities.

The first technical novelty was that with applets, all the logic of a game could be located on the client-side, without requiring a back-end infrastructure to run the game's engine anymore. In addition to this, with a language as powerful as Java entering this situation, more complex gameplay features could be implemented. We see **real-time multiplayer** games being made and achieving very good quality on their services, and a very good gaming experience indeed. Equally (if not more) important, Java applets standardized the **cross-browser** gaming, since the code of the applet would be executed by the JVM and that was the only requirement for the players' systems.

There also was the introduction of some formal graphics in Java applets. Of course, these graphics were minimal and limited to whatever would the native Java graphic libraries provide (at least this was the case back then). Nevertheless, a player could experience simple animation taking place on their gaming environment and was able to actively interact with objects within the game, as game control moved a small step further also. Last but not least, since the applet was pre-compiled and able to be executed exclusively on the client-side, there was no need of disruption in the game experience to refresh the browser.

A new era in Web-based gaming started with the introduction of Java applets. Huge Web platforms (e.g Yahoo) developed endless series of smaller or bigger Web games, attracting massive amounts of users to use their services. It was the period when all the classic board & table games like chess, backgammon & pool were being ported into the web, with the help of Java applets. And things only became better ever since.

The Domination Of Flash

Flash brought Web-based gaming before a paradigm-shifting situation, making its presence felt in all aspects of the Web with features and applications which are still being used today in a very wide scale. This technology took over where Java had left from, and enhanced the whole process, with impressive results. It helped developers define new standards for Web gaming, allowing for development of complex and information-sensitive

games, many of which still operate under the same paradigm today. This technology truly deserves all the credit for setting the fundamentals for all the innovations we have today.

The architecture of a Flash-based Web game is similar to the one with Java applets: the application is built in an offline environment, and then it gets embedded in a Web Page and executed via a Flash runtime environment, which is Flash Player. One big shift Flash introduced was that it re-defined what would be perceived as cross-browser with Java applets, since the runtime environment for Flash became a browser plug-in instead of a separate program in the players' system.

The Flash-based Web games are implemented in a language called ActionScript, a powerful programming language which is targeted for developing Web applications, optimized to be run by a browser -even with the help of a compatible plugin. Through ActionScript, Flash games were blessed with solid libraries for proper implementation of physics, game object interaction, animations and game control. Furthermore, Flash was extremely comfortable and performant with rendering of 2D graphics, easy and smooth integration of images and sprites. Being an object-orient programming language itself, ActionScript offered various levels of abstraction for more intuitive programming within a gaming context.

If Java was the language which brought all the vintage board games on the Web, then Flash was the technology which allowed developers to bring into the Web all the loved 2D arcade and console games. Within a short period of time, more and more Flash-based arcade game emulators started appearing in the Web, and every new one would have more impressive graphics than the previous ones, smoother gameplay and better game control. Incorporation of both mouse and keyboard to control a game became a common practice (whenever a game required so, of course), and the available devices were incorporated in the gameplay in such way so that they can perform more meaningful operations within a gaming context (e.g the "arrow up" button of the keyboard would make Super Mario jump). Additionally, Flash brought a more solidified use of sound in Web-games, something that was missing from its technological predecessors.

After a long period of clear domination in the world of Web-based gaming, Flash started to decline in terms of quality, it became unstable. One reason for this is the fact that Web browsers started becoming more advanced, implementing in-house engines to render graphics and handle communications, as if their developers were preparing for some new technology to come and take over. In parallel to this advancement, browsers became much more demanding from the third-party software that were running under them. Flash had to enter into a race of being compatible with more and more browsers which were

changing all the time themselves, and that affected the overall result. Instabilities were being caused, Flash-based Web games would crash more often, browsers would crash while trying to load the Flash player plugin, and an overall decline in the gaming experience. At that point, The desire for something new & fresh was expressed from all sides in the Web community. And that something new was imminent.

The Present

Web-based game development, as well as Web development in general, has passed a long period when the practice of composing a Web application out of heterogenous components was a standard. It is well known that browsers have been (and still are) participating on a constant race of feature support, sometimes in an upward direction (e.g Google implementing their own JavaScript engine for Chrome, called “V8”) and other times in a downgrading manner (e.g Safari Mobile discontinuing their support of Flash). Nonetheless, the desire for a more coherent set of universally supported features in the world of the Web was coming out of all sides of the field, with all parties involved (especially Web developers) were asking for a feature set that will be native, built within the fundamental blocks of basic HTML. And that desire started becoming reality with the launch of HTML5, and everything that came along ever since.

In this section, the research focuses on the technological tools which assemble the “star system” of today’s Web-based gaming, starting from the basic new native HTML5 elements up to advanced back-end systems which are built following the client-side Web development model, and the integration of much more advanced controller devices, always from the perspective of what impact all these have on Web gaming and Web-based game development.

HTML5: The Canvas

One of the most prominent primitive features that came along with HTML5 to change the picture of displaying and animating graphics on the browser is unquestionably the `<canvas>` object. Admittedly, it does not offer too much by itself (in terms of attributes and functions to operate on it), but without a doubt it is a platform for developing interesting concepts - with generic colors, images, animations, physics, video etc. - in a native fashion. As mentioned above, terms like “native fashion” describe a concept which had been expressed loudly in the past, and we can argue that it was a truthful desire. The clearest indication to support the latter is the fact that since HTML5 (and the Canvas in particular,

more than any other feature at least at the beginning) made its appearance, developers started shifting towards this new paradigm of developing Web applications by making use of native components. In fact, there are cases within the community where we see a generation of HTML5 purists manifesting, people who become dogmatic about using exclusively native HTML5 components for their web applications. In any case, technologies like Flash are certainly experiencing a significant decline on their impact and their high position of importance in this “support race” of Web browsers.

The HTML5 Canvas is closely akin to the Flash Stage. It is a rectangular piece of screen real estate that can be manipulated programmatically. Advanced Flash developers might recognize the Canvas as a close cousin to both the BitmapData and Shape objects in ActionScript. One can draw directly to the Canvas with paths and images, and transform them on the fly [1]. Of course, the main difference between the Canvas and Flash Stage as components is that the former is operated by a JavaScript API whereas the latter is programmed with ActionScript. Thus, Canvas gives web developers an API to directly manage drawing to a specific area of the browser. This functionality makes game development in JavaScript much more powerful than ever before [2].

The basic native Canvas API provided by HTML5 spreads into two realms: the regular API and the text API. The former offers a small set of simple JavaScript functions which allow drawing rectangle shapes, and filling them with a color, which can be set as well as the stroke of the rectangle itself can be set. The text API (as one would imagine) is a similarly small set of functions which allow drawing text patterns inside the canvas, again being able to customize the color of the letters.

Of course, the extension of native HTML5 components such as Canvas in order for them to work with advanced third-party systems, APIs & libraries is a fact. A good example of this is Processing.js, which is a JavaScript port of the very popular, Java based, Processing graphics engine. What this port works as follows: it creates a - somewhat transparent - layer which binds regular Java Processing code with a Canvas object with JavaScript. Ultimately, the Canvas object is used as the output medium of the original Processing code, which is interpreted and executed as JavaScript via the API which Canvas supports. This way, one who is comfortable in Java & Processing can create impressive & complex (and also interactive) graphics, perhaps have them interact with meaningful data coming from various sources of the Web and use the Canvas as their output.

In the world of Web games, we can say that the HTML5 Canvas (together with its native API and all the external tools built to operate upon it) is the first contender of what used to

be achieved in the past with technologies like Flash. As we will see in the following sections, there are ways of creating more complex graphics in the browser, but Canvas can be definitely considered as a very good start.

WebGL And Advanced Graphics In The Browser

WebGL is a powerful API that is incorporated into high-profile browsers like Firefox, Chrome and Safari. This API provides JavaScript bindings to OpenGL ES functions making it possible to provide hardware accelerated 3D content on web pages [3].

If we want to make a conceptual connection of WebGL and the Canvas, it is appropriate to say that WebGL uses the Canvas to draw 3D graphics on its surface (in technical terms: it exploits the 3D context of Canvas). Following the linguistic harmony of this new order of things in the Web, the WebGL API is in JavaScript as all the native component APIs are in HTML5. The syntax and function prototypes it consists of are extremely similar to the ones used for OpenGL ES (the equivalent full-standalone version of 3D accelerated graphics and predecessor of WebGL), which means that developers of OpenGL ES can easily code WebGL, given they are comfortable with JavaScript as a language.

Coming back to the HTML5 - Flash debate, WebGL comes as a contender to the very “fancy” results which were produced by Flash, all the 3D based graphics and animations in the browser. Main disadvantage of Flash is the need to have an installed plugin into the browser. This has always been problematic as it is even impossible for some computers to allow external plugin installations. Furthermore, mobile devices which are growing exponentially in the usage share, limit their Flash support every day.

It is a fact that such a technology as WebGL has not yet reached its peak on what it can contribute to gaming in the Web. However, it has already given a glance of what could be possible in this context. Potentially, the high levels of quality in advanced, hardware accelerated graphics together with high performance is bound to happen in the Web, and it will create a platform which will perform as many advanced console or standalone computer game platforms do.

Real-Time Communication With WebSockets

WebSockets is a newly introduced HTML5 construct which defines a full-duplex communication channel that operates through a single socket over the Web. HTML5 Web Sockets is not just another incremental enhancement to conventional HTTP communications; it represents a colossal advance, especially for real-time, event-driven web applications [4].

Although the WebSocket does not aim - as a technical component - to add up to the visual aspect of Web gaming, it contributes something that was never seen before in the Web in such fashion: the notion of **real-time communication**. Within a gaming context, the amount of game types that are possible of happening by using this construct is vast, if not countless. The very first and most obvious massive feature which is brought by this technology is the multi-player support for any game, in real-time, with minimal interference from any centralized server. This technology literally took away the “offline” part of gaming in the Web, since the player does not need to wait, refresh their browser or in any way disrupt themselves anymore from the gaming experience, given that the gameplay does not require them to do so.

A big advantage about WebSockets - which is the case for most of the cutting-edge new features of HTML5 - is that there are libraries built which take care of all the underlying infrastructure and make their usage for the developer as simple as using a AJAX-based API. From a developer’s perspective, the implication here refers to simple JavaScript calls and - relatively - simple callback functions; standard event-driven programming.

With such a powerful technological tool in their hands, developers can focus on games which may not be overwhelming on their visual aspect, however they can be extremely rich in the depths of interaction and responsiveness they provide. WebSockets make it possible to easily build scalable multi-player games, with real-time interaction via multiple channels for multiple purposes. And this fact by itself opened the Web gaming field up for more interesting games.

Smooth Back-End Systems: Node.js

Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. It uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices [5].

An interesting fact about how Node.js is implemented is that it does not use multithreading to execute tasks which need to run concurrently. It uses the event-driven model instead. Think of the Node server process as a single-threaded daemon that embeds the JavaScript engine to support customization. This is different from most eventing systems for other programming languages, which come in the form of libraries: Node supports the eventing model at the language level [6].

It is now obvious what the revolutionary introduction of Node.js is. It is the fact that it brought JavaScript - a language traditionally written and perceived to be executed in the client side - to the server side, with all the implications this may carry. The first, and maybe most important, implication is that one can now use a single programming language throughout their Web application for all their purposes. This gives the application more coherence and homogeneity as a piece of software, and of course better maintainability from the administrator's standpoint.

Although the notion of bringing JavaScript in the server-side is not an entirely new idea (for example, the Rhino JavaScript execution environment has been available for a long time [7]), Node.js managed to bring an enormous community of people together, people with enough knowledge and energy to write software that made Node.js look like a very promising platform for server-side implementations of Web applications. In fact - to a certain extend - it feels already natural for a developer to use Node.js as a back-end system for their Web application, especially if they want to build something fast, flexible, and totally up to date with the technological trends.

When developing a game, it is very important to find the proper technological tools to build the logic of the game, which is by no argument the backbone of the game, since the rules and general environmental and gameplay related properties belong to the logic. The need for a strong, fast, light-weight & robust back-end system is thus vital. What Node.js is offering for developers of Web-based games is exactly the above properties, plus the fact that the developer can code their game logic in the same language that they will use in their front-end, and the guarantee that their I/O - be it filesystem operations or database operations - will be asynchronous and blazing fast.

More Advanced Control: Microsoft Kinect

Whether in the context of game consoles or PCs (stand-alone or Web-games), game control was (and has been until relatively recently) somewhat static, in the sense that the player would only use their fingers to click on various types of buttons on a flat controller, in order to play the game. This was the case until the late 2006, when Nintendo came to revolutionize game control with Wii-Mote, a remote control device for their console, the Wii. This was a truly paradigm shifting invention for gaming, for it allowed for the players to use their whole body - and not just the fingers of their hands - in order to play a game. Now the position of the user, the angle they would hold the remote at, the speed they would move it and so many other properties of their body would matter while playing and affect their per-

formance. Clearly, the overall gaming experience for the players was taken to another level. And it was clear that the Wii-Mote would not monopolize the market for too long.

After this paradigm shifting introduction of full-body control in gaming, the device which very much took this type of control to a next level was Microsoft's Kinect Sensor. The Kinect sensor incorporates several advanced sensing hardware. Most notably, it contains a depth sensor, a color camera, and a four-microphone array that provide full-body 3D motion capture, facial recognition, and voice recognition capabilities [8].

The huge advantage of the Kinect over Wii-Mote is how programmable the former is compared to the latter. From early stages, Microsoft was officially releasing special SDK's, specifically written for developers to create applications of their own, making use of Kinect's sophisticated sensing functionalities (only in Windows). The Kinect for Windows SDK enables developers to use C++, C#, or Visual Basic to create applications and experiences that support gesture and voice recognition by using the Kinect for Windows sensor and a computer or embedded device [9].

Kinect was quickly incorporated to the world of Web-based gaming. The game engine which is most widely known that went towards this step is Unity3D. The Unity3D game engine provides a browser plugin allowing to display content inside the browser window, requiring a computer with average hardware, an average graphics card, a browser that supports JavaScript and the freely available Unity Web Player (IE, Firefox, Safari and every other Mozilla-based browsers support all the above) [10]. The stack which is created in order to run a game created by Unity3D in the browser is similar to the Flash paradigm: the initial software is made in a high-level language (C++), and then with the assistance of the plug-ins mentioned above the final executable is embedded and presented on a Web page. Unity3D is an extremely powerful engine for working with high-performance graphics and physics, and it is just as advanced with controllers like the Kinect. More specifically, the libraries which are implemented in Unity3D for the Kinect allow for the developers to basically re-program the device from scratch, in terms of what types of handling and what levels of sensing it should perform in the context of the game, according to their own desires. And the end result is transferable to the Web, which makes for a - potentially - very rich gaming experience with full body control, which is **multi-platform**.

With the general openness around Kinect being present, more ways of exploiting its capabilities in the Web have emerged, solutions which follow the "compatibility harmony" paradigm. Companies like Zigfu have created full-featured JavaScript APIs to control the device in the context of client-side-scripting. This makes the integration of a control device such

as the Kinect as simple as adding another layer in the client-side of a Web-based game, which implements handling and callback functions to translate the gestures into JavaScript events and various DOM manipulation directions.

Within the general argument that the evolution of the types of games one can play in the Web depend as much to how sophisticated the game controllers are as it does to the technological tools which are available out there for developing game logic and graphics, and given how much did devices like the Kinect revolutionize the notion of game control, one can also argue with proof that with such novel and cutting-edge devices opened up for a whole different type of games that can be made for the Web, making the platform a true contender against the traditional gaming platforms we used to have before.

The Future

It is obviously impossible to accurately predict the future, but we can certainly make good guesses based on the current situation in gaming and the Web. Especially given the fact that there are tools and technologies appearing out there which - if anything - give us quite clear glimpses of what the future could be in developing games with the browser as the main output.

Complete Web Game Development Frameworks

Following all the new impressive features introduced by HTML5 & Javascript, and with the help of computer science & software engineering patterns for developing big projects (MVC, the Server-Client model, the notion of Middleware etc), we see a big number of actively maintained libraries which make the lives of Web-based game developers much simpler, taking care of the underlying infrastructure and offering multiple levels of abstraction. In addition to this, what we can also see is the tendency of developers to create entire frameworks, sets of helper functions and libraries which lay down all the foundations one needs to start developing their Web-based game, depending on what component they are aiming for.

Some of the most popular & widely used of these frameworks (also referred to as “game engines” in the Web context) are the following:

- EaselJS, which provides straight forward solutions for working with rich graphics and interactivity with HTML5 Canvas. It provides an API that is familiar to Flash developers, but

embraces Javascript sensibilities. It consists of a full, hierarchical display list, a core interaction model, and helper classes to make working with Canvas much easier.

- lycheeJS, which is an environment-independent Javascript game engine, which means it will run in any theoretical Javascript supporting environment. The publishing process is optimized for development inside the Web Browser using the HTML5 platform adapters.
- Crafty, which is a small, simple, and lightweight game engine that can greatly help building prototypal or fully-featured 2D HTML5 games.
- Turbulenz, where the engine libraries are implemented in optimized Javascript supporting rapid iteration of game code and data. The engine executes directly in the browser and includes many features.
- Three.js, which does a really great job of abstracting away the headaches of getting going with 3D in the browser. With it one can create cameras, objects, lights, materials and more, and there is a choice of renderer, which means one can decide if they want the scene to be drawn using HTML 5's Canvas, WebGL or SVG.
- Construct 2, which is an HTML5 game maker, meaning that one is not actually coding under the framework. Instead, one uses actions, events and conditions to define the game's flow. Its very popular, and its engine is heavily tested and used. It has a very active community and new releases weekly.

This is clearly the future of Web-based game development. Meta-software, frameworks which take care of all the groundwork and leaving headroom for the developer's creativity. The most profound thing here is that all the frameworks/game engines mentioned above are free and open-source (except for Construct 2 which comes with variable pricing plans, including a free one). If we consider that the game industry depends on each of their teams' private knowledge & technological competence to make financial profit, then the fact that there are so many pieces of quality software out there that provide this competence a-priori and are licensed under the standard MIT license is something very new. Maybe in the world of the Web it does not sound like something new - most of the tools used for building Web applications are open-source - but in the world of gaming it is a quite revolutionary notion. And that by itself is extremely promising for the future of gaming in the Web, but perhaps in general as well.

Intelligent Control With Extremely Advanced Controllers

According to what is illustrated in the sections above, we notice a deep degree of evolution of the game controllers as means of input for Web based games, parallel to the one of the actual technologies of game development. In fact, it has been the case many times that a more advanced controller would give birth to ideas for more advanced and more creative games, thus influencing the notions of gaming concepts. Let us have a look at some key aspects of which plot the further evolution of these peripheral devices.

Controllers themselves have become much more sophisticated with regards to what types of input they can translate into game control signals. More specifically, we have controllers which can trace body movement, facial expressions, depth and sound from the user's side and translate all these types of interaction into in-game actions. That said, we can expect in the future to have mass-produced controllers which accept more subtle input types such as brain waves, heart rate or even attention levels. That fact by itself paints a promising picture of possible interaction within a gaming context.

Standardized Protocols

Nowadays, the ways of connecting & operating with peripheral devices as well as having them communicate with each other are limited to a quite small number. Protocols like USB for connections and OSC for communication are universal standards, and it is taken for granted that any device that wants to exist among all competitive products of its field should implement such protocols.

All these standardizations and conventions in the communication and operation layers of the peripheral devices (namely in our case the game controllers) create a "world of compliancy and compatibility" among the various devices out there, opening the way for harmonically & organically incorporating multiple devices of various types for a single sublime gaming experience. Furthermore, such standardizations allow easier interaction of these devices' drivers and operating software with the Web browser, solving the ever-existing browser support problem for good.

To enforce the above argument, as mentioned in the Section about Web based game development frameworks, we already have complete middleware solutions which help structuring and developing a Web based game application. In software engineering terms, that kind of software tend to provide very high levels of abstraction and modularization of all the components that take part in the development process of a Web game, thus providing for easier and more intuitive development of such components. Moreover, as argued above

also, it seems that the evolution of controlling the game has been parallel to the actual game technologies, so it is only logical that the browser development teams would want their browsers to be capable of handling and interacting with more sophisticated devices, for achieving a richer browsing experience. And it is very probable that the enhancement of Web browsers to support such devices - given the fact that we will have solid middleware solutions as the ones described in the Web game frameworks Section - will take place in the future.

Smart-phones

Smart-phones have already been part of the everyday life & technological interaction for many people, for a few years now, and sometimes in a bigger scale than actual computer users. And the capabilities they project with regards to usage and technological relevance are very impressive, as they support features and software which sometimes really make the computer a bit of a redundant tool for the user.

Apart from the environment for developing native smart-phone applications (so-called "Apps"), the smart-phone Web browsers drew big attention as well. Within the last three years, the focus of every self-respect company/institution/organization/blog engine/social network - and so many other categories - have been focusing on adapting and optimizing their websites for the mobile platforms. A situation which brought new definitions of usability ideas, forward-thinking designs, and a general contribution to shaping the standards in Web experience (e.g if Mobile Safari supports WebGL, Desktop Safari should do so as well).

As devices, mobile phones have their own sophisticated capabilities (accelerometers, compasses, GPS transceivers etc), a fact that gives them the potential of being a type of advanced game controller themselves. But they also do have their own advanced software which, combined with the former fact, gives them the potential of becoming advanced gaming platforms themselves. So, it is very possible that in the future mobile phones will be the next generation "Nintendo Game-Boy", with game sets that will be Web-powered and multi-platform, playable by any device, and with possibilities of inter-activity with other devices/platforms/players.

Conclusions

Gaming has been a very prominent activity among users since the very early years of computers, with a very wide & very committed audience. We can say today that the Web

holds a similar position (if not even more important) in a users' perception about the general capabilities of a computer. And we can see, as the research revealed, that these two activities have begun to converge in the past few years, with impressive results on the final experience of the user. Concluding this report, one can only say that the future of gaming in the Web looks very promising.

Selected Web Game Examples & Relevant Resources

- http://www.chromeexperiments.com/
- http://www.html5gamedevelopment.com/StateofHTML5GameDevelopment/#.UlR75GSWHGY
- https://www.scirra.com/blog/85/the-great-html5-mobile-gaming-performance-comparison
- http://html5games.com/
- http://www.html5games.net/
- http://www.html5rocks.com/en/
- http://jsdb.io/
- http://html5quintus.com/

References

1. Steve Fulton & Jeff Fulton, "*HTML5 Canvas: Native Interactivity and Animation for the Web*", O'Reilly, 2011

2. Ibid

3. Leung et al, "*Enabling WebGL*", WWW Developers Track, 2010

4. Lubbers et al, "*HTML5 Web Sockets: A Quantum Leap in Scalability for the Web*", http://www.websocket.org, 2011

5. Node.js Website, http://nodejs.org, retrieved 3 Oct 2013

6. Titkov et al, "*Node.js: Using JavaScript to Build High-Performance Network Programs*", IEEE Internet Computing, 2010

7. Ibid

8. Zeng et al, "*Microsoft Kinect Sensor and Its Effect*", IEEE Multimedia, 2012

9. Microsoft Kinect Sensor SDK Website, **http://www.microsoft.com/en-us/kinectforwindowsdev/Start.aspx**, retrieved 3 Oct 2013

10. Wendel et al, "*Woodment: Web-Based Collaborative Multiplayer Serious Game*", Transactions on Edutainment IV, Springer, 2010