하드코어 게임에서 웹으로의 이동 : Making History(역사 만들기)

작성자 : 매트 시그밀러 (Matt Seegmiller)
가마수트라 등록일 : 2010년 1월 5일

Making History II 발매가 다가오는 시점에서, Muzzy Lane Software의 The War of the World는 3D 전략 게임이 웹 브라우저로 도래를 가져왔고, 멀티 플레이어 경험을 더욱 할 수 있게 하는 특징으로써, 그 회사가 나아갈 새로운 길을 보여주고 있다.
Making History II: The War of the World

MHII는 Sandstone 상에서 만들어진 것이다. Muzzy Lane에서 사회적 네트워킹 특징을 가진 3D 브라우저를 기본이며 웹을 통합한 게임을 위한 새로운 플랫폼이 나왔다. 이러한 기술은 Muzzy Lane에서 중학생 과학 수업을 위한 게임 개발을 선도하는 Clearlab Project를 포함한 어려운 게임 프로젝트을 해낼 수 있는 능력을 갖게 해주었다.

MHII를 구축함으로, Muzzy Lane은 다양한 엔지니어링 도전 과제들을 직면하였다. 즉, 웹 브라우저에서 어떻게 3D 게임을 최상으로 렌더링 할 수 있을까? 후위 처리 장치(backend) 서비스를 어떻게 접근할 것인가? 내용물(content) 배포는 어떻게 다룰 것인가?

이 글에서는 Muzzy Lane이 직면했던 이러한 도전과제들과 그 문제를 어떻게 해결해 나갔는지에 대해 살펴보고 논의해보고자 한다. 그리고 Sandstone backbone을 구성하는 여러 핵심적인 디자인 결정들과 Making History와 더불어 다른 Muzzy Lane의 게임들이 플레이어들에게 역동적인 경험을 제공이 가능하게 해주는 디자인 결정들에 대해 설명할 것이다.

브라우저에서 3D Games 렌더링 하기

첫 번째 엔지니어링 도전과제 중에 하나는 브라우저에서 3D 게임을 실행시키는 최선의 방법을 결정하는 것이다. 진실로, 우리는 웹을 기초로 한 게임을 구축하는데 확실히 전념했으나, 브라우저에서 하는 게임적인 사고로 충분히 전념하는데 다소 시간이 걸렸다.

본래, 우리는 브라우저에서 막 발매될 게임들을 구축하고 개별로 운영할 계획을 갖고 있었다. 이렇게 개별 어플리케이션으로 하면 이행하기가 더욱 쉽기 때문이었고, 한편으로는 3D 게임들을 항상 꽉 찬 화면(full screen)으로 실행하기 때문이다. Flash 게임들은 브라우저에서 확실히 플레이가 되지만, 하드코어 3D 게임들도 실제로 되는가?

그러나 유튜브 비디오, 포드캐스트 그리고 e-book 리더스와 같은 웹 컨텐츠를 보면 볼수록, 그것을 접하며 갖는 감각은 적어지게 된다. 브라우저에서 플레이할 수 있는 3D 게임들은 꽉 찬 화면을 옵션으로 선택하여 감각을 갖게 해준다.

Making History II: 브라우저를 기초로 한, The War of the World는 Muzzy Lane Software의 Sandstone 플랫폼 상에 구축되어진 완전히 웹 통합적 WWII 전략 게임이다.

Making History II: The War of the World, a browser-based, fully web-integrated WWII Strategy Game, built on Muzzy Lane Software's Sandstone platform.

그것은 브라우저 윈도우에서 3D 그래픽을 가속화하는 하드웨어를 렌더링 하기에 전혀 어렵지 않은 것으로 판명되었다. 그래서 기본적인 윈도우 상에서 극복해야 하는 어려운 문제들은 한 개의 윈도우를 처리하며 해결할 수 있다. (이것은 리눅스나 Mac OS X에서 되는 것과 유사하다.)

브라우저 plug-in을 사용하므로, 대부분의 브라우저에 있는 3D 솔루션이 최근 이러한 문제를 해결하였다. 그러나 이에 대한 두 가지 단점이 있다. 첫째로, 많은 사람들은 custom browser plug-ins을 신뢰하지 않는다. (Flash와 Java plug-ins 은 예외이다.) 둘째로, 각각

의 브라우저 플러그 인은 브라우저마다 매우 상세하다. 심지어 Mozilla plug-in 건축은 많은 부라우저를 지원하고 있지만, 여전히 각각의 개별 브라우저를 위해 약간씩 종종 비틀기를(tweak) 해 줄 필요가 있다. 이것은 결국 밤새 관리해야 하는 상황이 되기도 한다.

우리의 해결책은 대신 Java extension을 만들어 내는 것이었다. 일반적으로, Java applets은 클라이언트 머신으로 접근을 다소 어렵게 하는 안전한 sandbox로 운영되고 있다. 이것은 종종 실행되는 본래 코드를 허락하지 않으며, 하드웨어 가속화를 위해 필요하다고 한다.

이러한 방식은 클라이언트 머신 상에 설치되어진 Java extension을 만드는 것이다. Java extension에서 앤드 유저가 그것을 설치하도록 되어 있기 때문에, classes는 신뢰되어지는 것으로 여겨지고 있다. 이러한 classes 문제없이 본래 코드를 내려 받고 작동시킬 수 있다. 일단 Sandstone Player가 설치되어지면, (Java extension을 포함하여), applet class는 각 페이지 마다 놓여진다. 이러한 applet은 내용이 어떤 것이든 필요한 것은 다운로드 받을 수 있고 디스크에 그것을 저장할 수 있다.

한번 이러한 컨텐츠가 다운로드 받아지면, 게임 엔진을 다운로드 하면서, 막 다운로드 되어진 C++코드를 받은 Java에서 C++로 call이 만들어진다. 이러한 엔진은 운영되기 위해 다운로드 된 패키지로 불리며, 서버로 연결할 수 있게 한다. 가장 중요한 점은, 자바에서 애플릿에 의해 만들어진 본래의 윈도로 그것이 처리할 수 있게 생기고, C++에서 접근할 수 있게 된다. 이것은 엔진이 장소를 제공하게 하며, 브라우저에 웹 페이지가 실제로 생기게 한다.

브라우저와 C+ 사이에서 심(shim)으로써 자바를 사용하는 이 해결책은, 그 자체가 가진 단점들이 있다. 클라이언트 머신 상에 설치되어진 자바에 의존적이라는 점이 있다. 이것은 보통의 경우이지만, 그렇지 않을 때는, 앤드 유저가 인스톨을 해야만 한다.

자바 확대 장치 (Java extension mechanism)는 표면적으로 브라우저를 다시 시작하지 않고 설치할 수 있게 하여, 그 확대 설치하는 것을 지원한다. 이것은 항상 브라우저를 다시 시작하지 않고 샌드스톤 플레이어를 설치할 수 있고, 게임을 다운로드 하고, 게임을 할 수 있도록 웹 페이지에 누군가 들어가는 것을 허락해주며, 그리고 자바 인스톨이 이미 된 것처럼 여겨지게 하는 것이 큰 특징인 것 같다. 이것은 우선적으로 이 방식이 선택한 주요한 이유 중에 하나이기도 하다.

실제로, 항상 그렇게 작동되는 것은 아니다. 확대 장치는 다운로드 할 때에 유저에게 피드백이 거의 없다. 빠르게 화면상에서 진행상황을 보여주기 위해서는, 설치되고 있는 나머지 것을 다운로드 하는 인스톨러를 매우 작은 것으로 사용해야만 한다.

게다가, 자바 플러그-인은 확대 설치자(extension installers)를 관리자로써 운영하는 환경에서 문제를 가지고 있다. 심지어 UAC의 모든 후프(hoops)가 적절하게 되었을 때에도 문제를 보인다. 잠시 동안, Sandstone Player은 매뉴얼 설치자가 되고 브라우저는 설치의 일부로써 다시 시작되어져야만 해야 하는 결과를 가져온다. 그렇지만 좋은 점으로는, 자바 플러그-인은 브라우저의 세부 문제 모두를 다룰 수 있어서, 우리가 매번 브라우전 기초로 하기

보다는 각각 플랫폼의 기초 위에 무언가 기초로 세우고 관리하면 되도록 해준다. 결과적으로, 우리는 윈도우상에서 많은 다른 브라우저들에서 플레이어를 운영시킬 수 있게 되어졌다. IE와 Firefox, 심지어 Chrome과 Opera는 모두 특별한 코드나 기타 지원 없이도 작동되고 있다.

The Sandstone platform supports true hardcore 3D games, like *Making History II* in the browser. Sandstone Player, but also the Java plug-in.

후위 처리 장치(Backend)서비스를 운영하기

초기에, 우리는 Sandstone을 지원하는 후위 처리 장치(backend) 상에 운영되는 수많은 서비스가 필요할 것이라 깨닫게 되었다. 이러한 서비스들은 각각 다른 서비스에 말을 할 필요가 있고, 사람들이 게임을 플레이 할 수 있도록 하는 웹 서버에 의해 접근할 수 있도록 일부 서비스가 필요하였다. HTTP 요청이 되는 이러한 서비스를 위하여 우리는 API의 대부분을 갖도록 선택하였다.

이것은 거기에 어떤 언어가 사용될지는 상관없이 Sandstone서비스에 말을 걸 웹 프레임웍을 수월하게 해준다. (대부분의 웹 프레임워크의 근본적인 능력이라 할 수 있는) HTTP요청을 하는 동안, 그것은 서비스들과 상호작용을 할 수 있다.

초기에, 대부분은 데이터는 XML로써 앞뒤로 통과하게 된다. 대부분의 데이터가 간단한 데이

터이기 때문에, 이것은 다루기 불편한 것으로 판명되었다. 결국, 우리는 이 데이터를 위해 평균 XML보다 훨씬 더 작고 우리가 사용하는 언어에서 훨씬 분석하기 쉬었던 JSON을 사용하도록 변경하였다.

이러한 서버 APIs가 HTTP요청에 따라 이행되어진 이후로, 우리는 HTTP요청을 이행시키기 쉬운 언어를 사용할 필요가 있었다.

우리가 PHP 혹은 Ruby 대신에 매우 일관성 있는 언어인 Python을 사용하는 것을 선택하였다. 그것은 웹 서버를 많이 보완해주며 쉽게 C++코드와 연결될 수 있다. 즉 게임 서버 운영을 위해 매우 용이하다.

내용(컨텐츠) 배포

웹 페이지에서 게임을 운영하는 것에 대해 논하자면, 브라우저에서 3D 렌더링 그래픽은 실제적인 문제는 아니다. 오히려, 문제의 가장 큰 부분은 우선 3D 예술 자산을 클라이언트 기계로 얻을 수 있게 하는 것이다. 3D 게임을 위한 예술적 자산들은 일반적으로 브라우저를 기본으로 하고 있는 전통적인 2D 게임보다 훨씬 더 커지고 있다.

처음부터, 우리가 필요할 때마다 그러한 자산들을 다운로드 하고, 메모리에 저장하는 것을 용인할 수 없을 것이라고 여겼다. 복잡한 Flash 게임들을 다운로딩하기 위해 오래 기다려야 하는데, 대부분의 3D 게임들은 이것보다 훨씬 용량이 크다.

추출하는 시간이 허용될 만한 지점에서 압축된 예술 자산의 방법을 따르기보다, 우리는 Sandstone 플레이어를 만드는 것을 시도하기로 결정했었고, 그래서 다운로드 하는 동안 더 나은 유저 경험을 창조하는 것이 가능하게 되었다.

우리가 브라우저와 C++게임 엔진 사이에서 차이를 연결시킬 Sandstone 플레이어에서 이미 자바를 사용하게 된 이후로, 우리는 자바에서 다운로더를 작성하는 것을 선택했다. 이것은 또한 C++에서보다 더욱 쉬운 것으로 판명되었다. 컨텐츠가 다운로드 되는 방식인 자바(Java)가 HTTP 요청을 만들기 위해 훨씬 더 나은 라이브러리(library)를 지원하고 있다.

이 다운로더는 게임을 시행하는 애플렛에서 개별 애플렛으로써 운영될 수 있고, 그것 주변의 페이지와 함께 커뮤니케이션 할 수 있어서 그 페이지는 원하는 정보는 어떤 것이든 해낼 수 있다.

진행 막대progress bar)를 보이게 할 수 있고, 또한 게임 다운로드 하는 동안 다른 사람들과 함께 채팅 할 수 있게 해준다. 그리고 다운로드를 완성하는 것을 기다리는 동안 작은 용량의 2D 미니게임을 할 수 있게 해준다. 아래 줄에는 그 시점에 기술적 질문보다는 유저들의 경험적 질문을 더 볼 수 있게 해준다.

우리는 컨텐츠 배포를 증강시킨다는 목표를 위해 중요한 두 가지 결정을 내렸다. 첫째로,

우리는 다시 사용할 수 있는 소량의 컨텐츠를 깨뜨리기로(break) 결정했다. 그리고 둘째로, 우리는 이러한 덩어리를 클라이언트 기계의 로컬 디스크 캐시에 저장하기로 결정하였다.

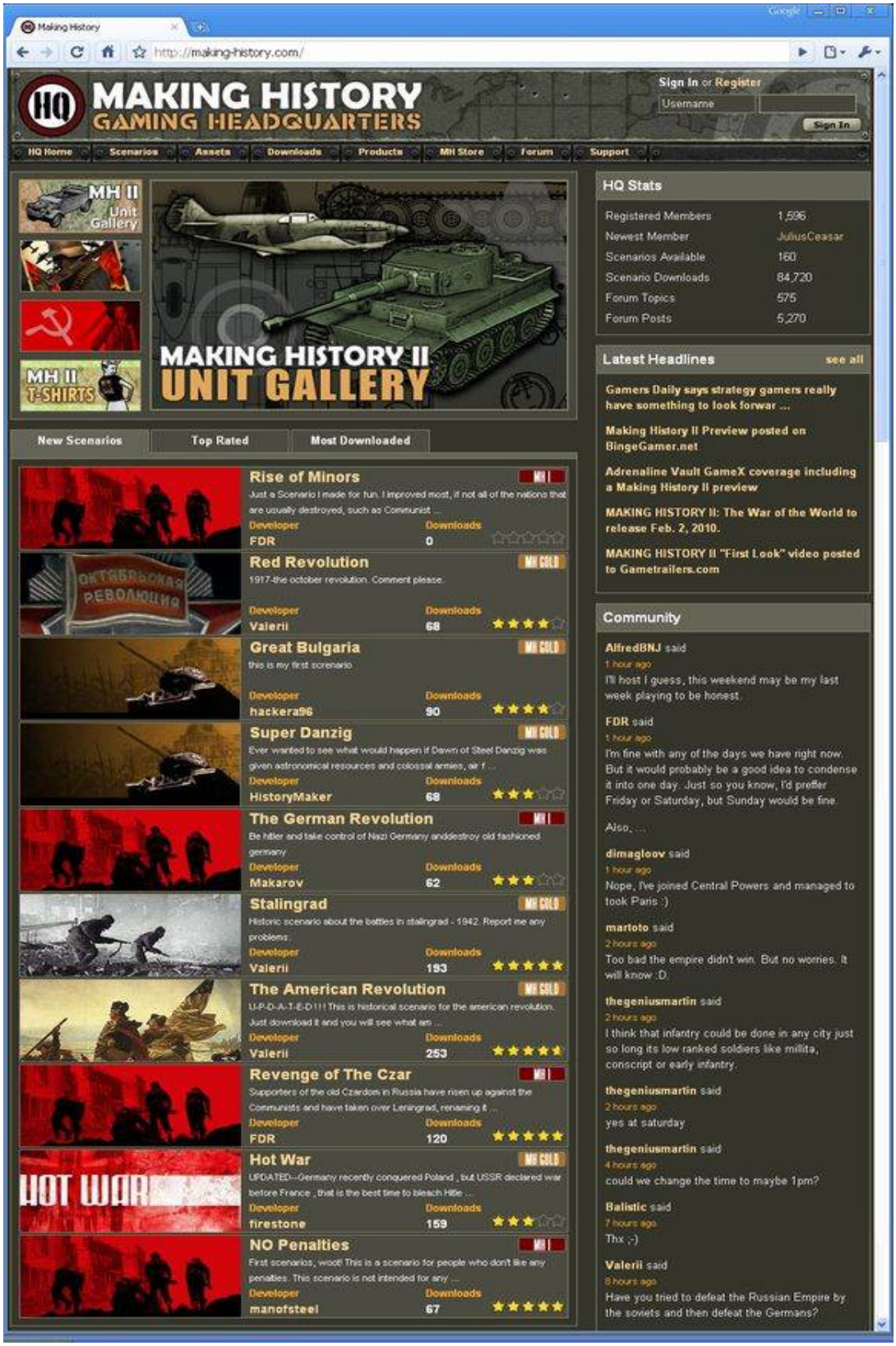
이러한 결정은 만약 두 가지 시나리오가 같은 컨텐츠의 99%를 사용한다면, 만약 사용자가 이미 시나리오 A를 플레이하기 위해 컨텐츠를 모두 다운로드 하였고, 이미 그 컨텐츠의 99%를 가지고 있다면, 그 컨텐츠는 시나리오 B가 필요하여 로컬 디스크 캐시에 저장되어진다. 그러므로 시나리오 B는 매우 작은 용량일 것이다.

그 다음으로, 우리는 두 가지 버전의 시스템을 우리의 컨텐츠 시스템으로 두었다. 첫 번째 시스템은 간단히 컨텐츠 패키지의 identifier로 변환되어지고, 다른 컨텐츠가 “깨어지는(break)" 변화가 생길 때, 패키지의 주요한 업그레이드를 위해 사용되어진다.

예를 들면, 코드가 다른 모듈로 작업되지 않도록 변환되어지거나 3D모델로 바뀌거나 완전히 다른 크기가 되어서 사용하려는 장소에 맞지 않게 될 것이다. 이렇게 되면, 이 패키지를 사용하는 어떤 다른 패키지는 새로운 버전을 사용하도록 변환을 손으로(수작업) 해야 할 것이다.

이것은 새 버전과 옛 버전 모두 다르게 알아 볼 수 있기 때문에 유용하다. 그리고 그것들 모두 동시에 클라이언트 기계 상에 동시에 존재할 수 있게 된다. 그래서 시나리오 A는 버전 1을 사용하고 시나리오 B는 버전 2를 사용한다면, 클라이언트 기계는 버전 1과 버전 2를 시나리오 A와 시나리오 B를 모두 플레이를 하기위해 모두 동시에 디스크에 가질 수 있다.

다른 버전의 시스템은 그 장소에 어떤 패키지의 수정본(revision)으로 변환시킨다. 마치 버그를 고치는 것처럼, 이것은 한 패키지를 전혀 깨지지 않게 변환을 시킬 수 있다. 예를 들면, 메모리가 코드에서 새는 것(leak), 또는 3D 모델이 때때로 부정확하게 주어져 메시(mesh)를 고치려고 변환되는 것이다. 이러한 타입의 업데이트는 컨텐츠를 얻을 때에 자동적으로 발생한다. 그래서 당신이 플레이하는 게임은 항상 업데이트로 최신이 된다.



The Making History Gaming Headquarters (GHQ) facilitates the sharing of user created content.

게임에서 웹 서버로 커뮤니케이션 하기

우리가 서로 대화하는 서비스를 만들기 시작 했을 때, 우리는 서버가 항상 이용 가능한 것을 기록하는 방식을 소유할 필요가 있다는 것을 깨달았다. 예를 들면, 한 게임이 시작되었을 때, 거기에는 하나의 특정 게임 서버가 필요할 것이고 그것은 켜져 있어야 한다. 만약 여러 게임 서버들이 운영되고 있다면 어떨까? 우리는 그 여러 개 중에 무엇을 운영해야 할지 것이 무엇인지 어떻게 알겠는가?

그것은 우리의 인터 서비스 커뮤니케이션 시스템인 MISSIVE에 근본적 위협이었다. 우리는 서로 대화할 방법이 필요했고, 이러한 서버들이 존재하고 있으며, 동시적으로 서로 간단한 메시지를 보낸다.

MISSIVE에 전임자는 다른 API와 같은 것들과 마찬가지로, HTTP 요청을 사용하기 위해 세워졌다. 그러나 우리는 재빨리 이것이 잘 작동되지 않을 것을 알아차렸다. 만약 그것을 요청하기로 선택했다면, HTTP요청은 오로지 poll 모델만 지원하고 정보는 거기서만 습득될 수 있다.

만약 서버가 시작되거나 멈춰졌을 때, 우리는 현존하는 서버를 통하여 이러한 정보를 분산시킬 필요가 있다. 이상적으로 말하면, 즉각적으로 해야 한다. 그래서 MISSIVE는 간단한 TCP/IP 연결과 메시지를 전하는 간단한 메시지 포맷을 사용한다. 이것의 주요 이행은Python 웹 서비스가 서로 커뮤니케이션 하도록 하면서 Python에서 행해진다.

우리가 이러한 시스템을 세워놓은 후에, 우리는 게임들 안에서 이용 가능한 유사한 어떤 것이 발생할 수 있음을 심히 깨달았다. 만약 한 게임이 어떤 중요한 일이 언제든 발생했을 때마다 메시지를 보낸다면, 이러한 메시지는 다른 서버에 의해 받아들여질 것이고, 데이터는 데이터베이스에 기록될 것이다.

일단 데이터가 데이터베이스 안에 있으면, 그것은 즉시 정보를 디스플레이하길 원하는 웹사이트 어떤 곳으로도 접속이 가능하다. 마침내 게임엔진을 위해 우리는 MISSIVE 클라이언트를 이행하게 된다. 그래서 한 게임 안에서, 이 데이터를 원하는 어떤 외부 서버에서든지 게임으로 데이터가 보낼 질 수 있다. 그리고 메시지 또한 외부에서 게임으로 보내질 수 있다.

게임 엔진/모듈의 선택

우리는 사용되고 있는 게임 엔진에 관한 가능성 있는 몇 가지 가정들을 내려 보았다. 우리는 Sandstone에 플러그를 연결하는 게임 엔진으로 실행되어지는 C++ interface와 함께 시작되었다.

게임 서버가 시작되었든지 브라우저에서 로컬 게임이 작동되고 있는지는 상관이 없이, Sandstone이 게임을 시작할 때마다, 그 게임은 인터페이스를 통하여 시작되어진다. 사실,

이러한 인터페이스의 실행은 컨텐츠 패키지 자체라고 할 수 있다.

그래서 사용되어진 전체 게임 엔진은 이전에 설치되지 않고, 그것은 마치 다른 컨텐츠처럼 다운로드 되고 업데이트가 된다. 이것은 쉽게 우리가 우리 엔진에 버그를 고칠 수 있게 해줄 뿐만 아니라 엔진의 다른 주요한 버전에서 운영되는 여러 게임들이 동시에 작업할 수 있게 한다.

비록 이 질문은 왜 우리가 LOCUST 게임 엔진을 만들기로 선택했는지에 관한 것이다. 왜 이미 존재하는 3D 게임 엔진을 우리의 것으로 사용하지 않는가, 그리고 엔진을 다르게 운용하기 위해 새로운 기술을 사용하는가? 그 진실은, 게임 엔진에 관한 우리가 세운 자격조건들을 맞출 수 있는 엔진이 없다는 것이다. 이러한 자격조건들은 다음과 같다.

- 게임 클라이언트는 이슈 없이 또 다른 과정 안에서 운영될 수 있다.
- The game client can render as a child of a parent window that is passed in

게임 클라이언트는 마치 부모 윈도우(parent window) 화면에서 나오는 자녀 윈도우 화면처럼 제공할 수 있다.

- 게임 내용(컨텐트)은 작게, 재사용할 수 있는 덩어리들로 나눠질 수 있다.
- 게임 서버는 쉽게 웹 서비스들에 커뮤니케이션을 하기 위한 도구로 사용되어 질수 있다.

대부분 게임 엔진들은 처음에 두 가지 지점에서 작동되지 않는다는 것이 놀랍다. 대부분 게임 엔진들은 그 프로세스의 작업 디렉토리가 .exe 파일 위치해 있고, 거기에서 내용을 찾을 수 있도록 되어있다. 만약 프로세스가 브라우저 프로세스라면 이것은 간단히 작동되지 않고, 그것은 몇 가지를 작업 디렉토리에서 행해질 수 있을 것이다. 그리고 그 전체 게임 엔진의 기초적 방식에 대한 가정이 된다. 즉 #1은 실현가능하지 않다는 뜻이다.

두 번째는 종종 그러한 케이스가 있다는 점이다. 많은 게임 엔진들은 게임이 탑 레벨 윈도우에서 작업되는 것으로 가정하고 있다. 부모 윈도우 하에서 제공할 수 있는지 묻는 것을 전혀 지원하지 않는다. 이러한 이슈에도 불구하고, 우리는 Sandstone을 사용하는 브라우저에서 실제적으로 제공하기 위해 두 개정도의 게임 엔진에 안달내고 있었다. 그것이 뛰어나지 않으며, 우리가 그러한 게임을 나르기를 전혀 원하지 않는 그런 것이었다. 그러나 그것은 가능성을 보여줄 수 있도록 충분히 작동을 하였다.

그러나 어떠한 게임 엔진이 한 윈도우 안에서 제공될 수 있다고 하더라도, 대부분의 게임 엔진들은 #3 또는 #4를 만들려고 하지 않을 것이다. 대부분 게임 내용(컨텐츠)는 어떠한 한 시나리오 혹은 극대화 할 수 있는 셋트로 전체 레벨이 되어있는 레벨을 기초로 만들어져 있다. 이것은 어떠한 재사용할 수 있는 덩어리들, 각 개별적으로 다운로드 받아야 끝낼 수 있는 전체 레벨을 보충하지 않고 있다. 마지막으로, 그것은 웹 서비스로 게임의 데이터를 커뮤니케이션하는 것이 용이하였다. 그리고 우리는 이렇게 쉽게 할 수 있는 게임 엔진이 필요했었다.

추가로, 우리는 우리의 게임을 만드는 것을 향상시키기를 원했고 'Making History's lively modding community' (역사 만들기를 생생하게 바꾸는 공동체)를 지원할 수 있는 변

환 능력을 원했고, 우리의 중요한 게임 작업에서 훨씬 더 나은 개발 속도와 힘을 제공하기를 원했다. 우리는 첫 번째 Making History 게임(The Calm and the Storm: 고요와 폭풍)을 위해 개발하는 구조를 세웠고, 그 게임 데이터는 XML에서 만들 수 있게 하였다.

우리는 XML에서 전체적으로 완벽한 레벨을 만들 수 있는 게임 엔진을 확장시켰다. 스크립팅을 위해 JavaScript 끼워 넣는 것을 사용하였다. 우리는 Making History II 와 다른 프로젝트들의 작업에서 생산성과 유연성을 향상된 것을 확인하고 있다.

웹 통합적, 브라우저 기초적 게임으로 가는 우리의 길은 많은 어려움을 겪기도 하였다. 또한 우리는 그 결과에 대해 우리는 신이 나기도 하였다. 그리고 앞으로 곧 발매될 Making History II와 웹 통합적 3D 게임들과 관련된 새로운 종류들을 효율적으로 사용하고 창조하기 위해 플렛폼이 제공하는 가능성들을 보고 흥분하여 들떠있다.