Autodesk®
Scaleform®

HUD 키트 개요

이 문서는 다양한 기능과 재사용성을 갖춘 AAA 등급의 일인칭 슈팅 게임용 사용자 인터페이스 솔루션인 Scaleform 4.3 HUD 키트에 대해 기술한 문서입니다. HUD 키트를 제어하는 Flash UI 콘텐츠와 C++ 코드를 중점적으로 다룹니다.

작성자: Nate Mitchell, Prasad Silva

버전: 2.0
최종 수정일: 2010 년 7 월 30 일

Autodesk®
GAMEWARE

Copyright Notice

Autodesk® Scaleform® 4.3

© 2013 Autodesk, Inc. All rights reserved. Except as otherwise permitted by Autodesk, Inc., this publication, or parts thereof, may not be reproduced in any form, by any method, for any purpose.

Certain materials included in this publication are reprinted with the permission of the copyright holder.

The following are registered trademarks or trademarks of Autodesk, Inc., and/or its subsidiaries and/or affiliates in the USA and other countries: 123D, 3ds Max, Algor, Alias, AliasStudio, ATC, AutoCAD, AutoCAD Learning Assistance, AutoCAD LT, AutoCAD Simulator, AutoCAD SQL Extension, AutoCAD SQL Interface, Autodesk, Autodesk 123D, Autodesk Homestyler, Autodesk Intent, Autodesk Inventor, Autodesk MapGuide, Autodesk Streamline, AutoLISP, AutoSketch, AutoSnap, AutoTrack, Backburner, Backdraft, Beast, Beast (design/logo), BIM 360, Built with ObjectARX (design/logo), Burn, Buzzsaw, CADmep, CAiCE, CAMduct, CFdesign, Civil 3D, Cleaner, Cleaner Central, ClearScale, Colour Warper, Combustion, Communication Specification, Constructware, Content Explorer, Creative Bridge, Dancing Baby (image), DesignCenter, Design Doctor, Designer's Toolkit, DesignKids, DesignProf, Design Server, DesignStudio, Design Web Format, Discreet, DWF, DWG, DWG (design/logo), DWG Extreme, DWG TrueConvert, DWG TrueView, DWGX, DXF, Ecotect, ESTmep, Evolver, Exposure, Extending the Design Team, FABmep, Face Robot, FBX, Fempro, Fire, Flame, Flare, Flint, FMDesktop, ForceEffect, Freewheel, GDX Driver, Glue, Green Building Studio, Heads-up Design, Heidi, Homestyler, HumanIK, i-drop, ImageModeler, iMOUT, Incinerator, Inferno, Instructables, Instructables (stylized robot design/logo), Inventor, Inventor LT, Kynapse, Kynogon, LandXplorer, Lustre, Map It, Build It, Use It, MatchMover, Maya, Mechanical Desktop, MIMI, Moldflow, Moldflow Plastics Advisers, Moldflow Plastics Insight, Moondust, MotionBuilder, Movimento, MPA, MPA (design/logo), MPI (design/logo), MPX, MPX (design/logo), Mudbox, Multi-Master Editing, Navisworks, ObjectARX, ObjectDBX, Opticore, Pipeplus, Pixlr, Pixlr-o-matic, PolarSnap, Powered with Autodesk Technology, Productstream, ProMaterials, RasterDWG, RealDWG, Real-time Roto, Recognize, Render Queue, Retimer, Reveal, Revit, Revit LT, RiverCAD, Robot, Scaleform, Scaleform GFx, Showcase, Show Me, ShowMotion, SketchBook, Smoke, Softimage, Socialcam, Sparks, SteeringWheels, Stitcher, Stone, StormNET, TinkerBox, ToolClip, Topobase, Toxik, TrustedDWG, T-Splines, U-Vis, ViewCube, Visual, Visual LISP, Vtour, WaterNetworks, Wire, Wiretap, WiretapCentral, XSI.

All other brand names, product names or trademarks belong to their respective holders.

Disclaimer

THIS PUBLICATION AND THE INFORMATION CONTAINED HEREIN IS MADE AVAILABLE BY AUTODESK, INC. "AS IS." AUTODESK, INC. DISCLAIMS ALL WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE REGARDING THESE MATERIALS.

Autodesk Scaleform 연락 방법

문서	HUD Kit 개요
주소	Scaleform Corporation
	6305 Ivy Lane, Suite 310
	Greenbelt, MD 20770, USA
홈페이지	www.scaleform.com
전자 우편	info@scaleform.com
직통 전화	(301) 446-3200
팩스	(301) 446-3199

차례

1 소개

Scaleform HUD(Heads Up Display) 키트는 게임 UI 의 커스터마이징과 적용을 용이하게 해주는 사용자 인터페이스 키트이며, 우수한 성능 및 풍부한 기능을 갖춘 AAA 등급의 여러 제품 중에서 단연 선두를 자랑합니다. Scaleform HUD 키트는 새로운 Scaleform® Direct Access API 를 사용하여 고성능 일인칭 슈팅 게임(FPS) UI 를 제작하는 방법을 보여줍니다.

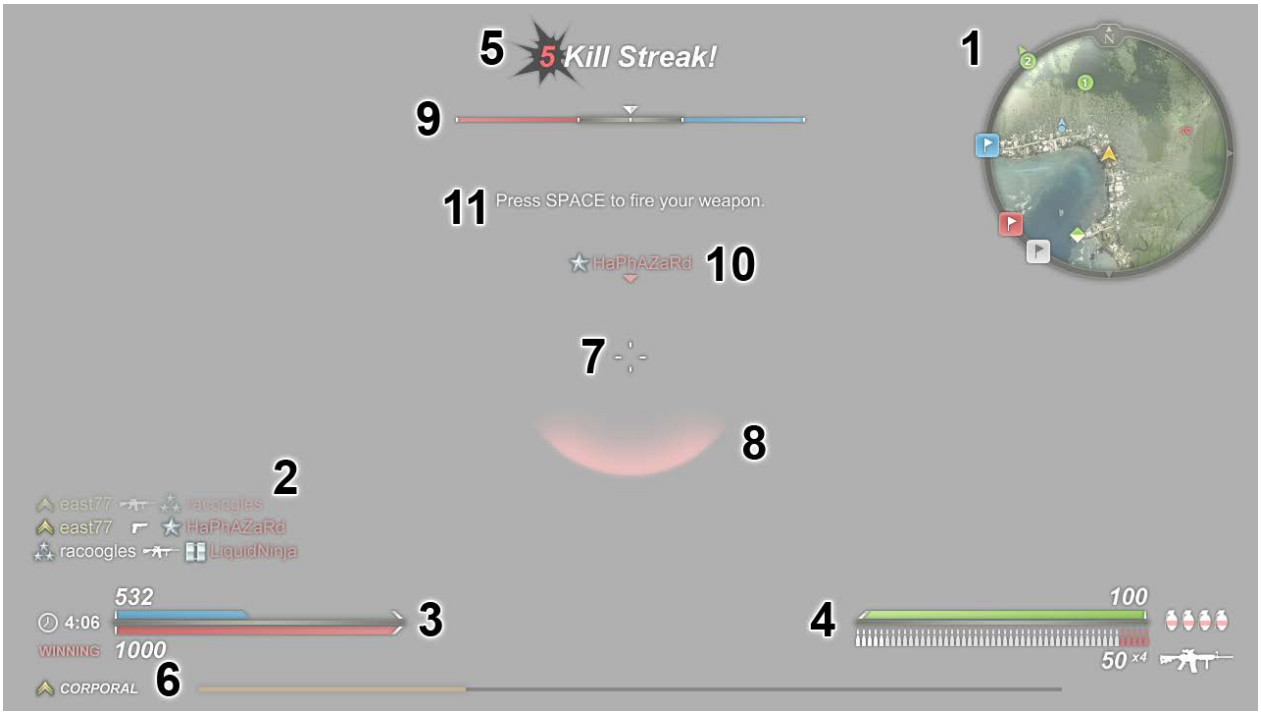


그림 1: HUD 개요

Scaleform HUD 키트에 포함된 재사용 가능한 UI 구성 요소는 다음과 같습니다.

1. 게임 미니맵
2. 게임 이벤트 애니메이션 로그
3. 점수판 및 팀 통계
4. 플레이어 체력/탄약/상태
5. 애니메이션 팝업 알림
6. 경험치 및 순위 디스플레이
7. 동적 조준점
8. 방향별 타격 표시기
9. 목표 점령 표시기

10. 빌보드/네임플레이트
11. 플레이어 텍스트 알림

Scaleform HUD 키트는 모든 FPS 게임에 즉시 적용 가능한 UI 솔루션을 제공하지만, 그 사용 방법은 제공되는 콘텐츠에 국한되지 않습니다. 본 키트에 포함된 각각의 구성 요소를 커스터마이징하거나 확장하여 다양한 형태의 게임이나 응용 프로그램에 적합한 새롭고 혁신적인 인터페이스를 제작할 수 있습니다.

2 개요

2.1 파일 위치 및 빌드 주의사항

다음 위치에 본 데모와 관련된 파일이 있습니다.

- *Apps\Kits\HUD* - 실행 가능한 HUD 데모의 C++ 코드 포함.
  Bin\Data\AS2 or AS3\Kits\HUD - Flash asset 및 ActionScript 코드 포함. (AS2 디렉토리는 ActionScript 2/Flash 8 의 에셋이 있고, AS3 디렉토리에는 ActionScript3/Flash10 에셋이 있다.)
- *Projects\Win32\{Msvc80 or Msvc90}\Kits\HUD* – Visual Studio 2005/2008 에서 사용 가능한 Windows 용 데모 프로젝트 포함.
- *Projects\Xbox360\{Msvc80 or Msvc90}\Kits\HUD* – Visula Studio 2005/2008 에서 사용 가능한 Xbox 360 용 데모 프로젝트 포함.
- *Projects\Common\HudKit*.mk – HUD 키트용 PS3 makefile.

Bin\Kits\HUD 에 미리 빌드된 Windows 용 실행 데모파일 HUDKit.exe 가 있습니다. *시작 메뉴나 브라우저에서도 접근할 수 있습니다.*

Windows 에서 이 데모를 빌드하고 실행하려면 *Projects\Win32\Msvc80\Kits*(또는 *Projects\Win32\Msvc90\Kits)*에 위치한 Scaleform 4.3 Kits.sln 파일을 사용하십시오. 솔루션에서 데모를 실행하기 전에 디버깅을 위한 "작업 디렉터리"를 반드시 *Bin\Data\AS2\Kits\HUD or Bin\Data\AS3\Kits\HUD* 로 설정해 주십시오.

Xbox 360 에서 이 데모를 빌드하고 실행하려면 *Projects\Xbox360\{Msvc80}\Kits\HUD*(또는 *Projects\Xbox360\{Msvc90}\Kits\HUD*)에 위치한 Scaleform 4.3 Kits.sln 파일을 사용하십시오. 데모 실행에 필요한 모든 에셋(asset)은 *Bin\Data\AS2 or AS3\Kits\HUD*에 위치해 있으며, 편집이 완료되었을 때 대상 Xbox 360 으로 배치됩니다.

PS3 에서는 Scaleform 설치 디렉터리 루트에서 *make* 명령을 실행할 수 있습니다. 이 명령을 실행하면 기본적으로 HUD 키트를 포함한 모든 실행 가능한 데모를 빌드합니다. PS3 에서 실행 파일은 SN 시스템 도구 집합을 통해 실행할 수 있습니다. 실행 파일은 *Bin\PS3* 로 빌드되며, 다음 옵션을 이용하여 실행되어야 합니다.

- app_home/ Directory: *{Local Scaleform Directory}\Bin\Data\AS2 or AS3\Kits\HUD*

2.2 데모 사용 방법

HUD 키트 사용에 대한 확실한 전달을 위해 프로젝트에는 홍팀과 청팀으로 나누어 전장에 흩어진 목표의 통제를 놓고 겨루는 간단한 게임 시뮬레이션이 포함되어 있습니다.

미니맵에 표시된 목표를 포획하고 적 플레이어를 죽이면 점수를 획득합니다. 목표를 점령하여 통제 중인 팀은 매 3 초당 1 점씩 획득합니다. 적을 한 명 죽일 때마다 1 점씩 획득합니다. 먼저 500 점을 달성하거나 라운드 시간(15 분)이 끝났을 때 점수가 높은 팀이 승리합니다.

모든 플레이어는 기관총 한 정, 권총 한 정, 수류탄 4 개를 가지고 시작하며, 총탄은 줄어들지 않습니다. 플레이어가 죽으면 무기와 총탄이 다시 보급됩니다.

회전하는 녹색 다이아몬드는 파워업 아이템으로, 전장의 무작위 위치에 생성됩니다. 해당 아이템을 입수하면 체력이 사전에 정해진 만큼 증가하고, 무작위로 지정된 무기의 총탄이 보급됩니다.

2.3 컨트롤 체계

2.3.1 키보드(Windows)

W, A, S, D	사용자의 플레이어를 제어합니다. W, S 를 입력하면 플레이어가 각각 전방, 후방으로 움직입니다. A, D 를 입력하면 플레이어가 각각 좌측, 우측으로 이동합니다.
SPACE	장착한 무기를 발사합니다. 범위, 위력, 발사속도는 장착한 무기에 따라 다릅니다.
R	장착한 무기를 재장전합니다.
G	수류탄을 던집니다. 범위 내에 근접한 플레이어를 대상으로 하여 폭발하며, 폭발 반경 내의 모든 적에게 데미지를 줍니다.
ESC	제목 표시줄과 텍스트 필드 사이에 응용 프로그램의 디스플레이 통계에 대한 표시 유무를 변경합니다.
B	사용자 플레이어의 AI 제어 유무를 변경합니다. 활성화 상태에서는 게임플레이와 관련된 입력이 불가능합니다.
1, 2	무기를 변경합니다. • 1 을 누르면 권총을 선택합니다. • 2 를 누르면 라이플을 선택합니다.

2.3.2 컨트롤러(Xbox/PS3)

방향 패드 **왼쪽 조이스틱**	사용자의 플레이어를 제어합니다. W, S 를 입력하면 플레이어가 각각 전방, 후방으로 움직입니다. A, D 를 입력하면 플레이어가 각각 좌측, 우측으로 이동합니다.
오른쪽 트리거	장착한 무기를 발사합니다. 범위, 위력, 발사속도는 장착한 무기에 따라 다릅니다.
X 단추 / 네모 버튼	장착한 무기를 재장전합니다.
왼쪽 트리거	수류탄을 던집니다. 범위 내에 근접한 플레이어를 대상으로 하여 폭발하며, 폭발 반경 내의 모든 적에게 데미지를 줍니다.
B 단추 / 동그라미 버튼	제목 표시줄과 텍스트 필드 사이에 응용 프로그램의 디스플레이 통계에 대한 표시 유무를 변경합니다.
Y 단추 / 세모 버튼	무기를 계속 변경합니다.

3 아키텍처

HUD 키트는 Flash UI asset 과 C++ 코드로 구성되며, 이들은 HUD 의 표시 상태를 업데이트합니다. 또, 더미 데이터로 HUD 를 구동할 수 있는 환경을 제공하는 C++ 시뮬레이션도 포함됩니다.

3.1 C++

C++ 코드는 크게 HUD 뷰에 표시하는 인터페이스와 HUD 뷰에 데이터를 제공하는 환경으로 나눌 수 있습니다. MVC 패러다임에서 인터페이스는 곧 컨트롤러이며, 환경은 모델, Flash UI 는 뷰입니다. Fx 접두사가 붙은 파일과 클래스는 HUD 키트 뷰 코어의 인터페이스를 정의합니다. 접두사가 없는 파일은 데모 또는 시뮬레이션에 필요한 인터페이스를 정의합니다.

어댑터 디자인 패턴은 HUD 뷰에서 시뮬레이션을 분리하기 위해 구현되었습니다. 어댑터는 시뮬레이션과 HUD 뷰 사이의 요청을 해석합니다. 새 게임에 HUD 뷰를 적용하려면 해당 게임과 HUD 뷰를 연결하는 사용자 정의 어댑터 클래스를 개발해야 합니다. 이것을 사용하면 코드를 크게 수정하지 않고도 HUD 키트를 재사용할 수 있습니다.

이 문서에 기술된 코드는 새 Scaleform Direct Access API 의 UI 업데이트 완료 및 애니메이션 속도를 이전보다 향상시켜주는 Scaleform 의 최적 구현을 제공합니다.

C++ 코드는 다음 파일로 구성되어 있으며, 해당 파일은 모두 *Apps\Kits\HUD* 및 하위 폴더에 있습니다.

3.1.1 데모

- **HUDKitDemo.cpp** – 표준 Scaleform Player 위에 빌드한 핵심 응용 프로그램입니다. C++ 컨트롤러를 통해 시뮬레이션 시작, 멤버 초기화, SWF 파일 불러오기 및 등록을 제어합니다.
- **HUDKitDemoConfig**. – FxPlayerConfig.h 를 기반으로 하는 HUDKitDemo 용 설정 파일. 컨트롤러 매핑과 Windows 응용 프로그램 정의 파일이 포함되어 있습니다.

3.1.2 HUD/미니맵 뷰 코어

- **FxHUDKit.h** – HUD 뷰 업데이트를 위해 사용되는 핵심 HUD 키트 형식 파일.
- **FxHUDView.h/.cpp** – 로그 및 빌보드 시스템을 포함하여 HUD 뷰에서 사용되는 형식을 제공하는 파일.

- **FxMinimap.h** – 게임 환경 및 응용 프로그램이 구현할 수 있는 인터페이스를 선언하여 미니맵 뷰에 연결하는 파일.
- **FxMinimapView.h/.cpp** – 미니맵 뷰에서 사용되는 형식을 제공하는 파일.

3.1.3 게임 시뮬레이션

- **HUDAdapter.h/.cpp** – 편리한 재사용성을 허용하기 위해 HUD 에서 시뮬레이션을 분리하는 어댑터 디자인 패턴 구현 파일.
- **HUDEntityController.h/.cpp** – 주로 시뮬레이션 로직과 AI 행위를 업데이트하는 데 사용되는 개체를 조종하는 컨트롤러 형식을 선언하는 파일.
- **HUDSimulation.h/.cpp** – 게임 UI 를 구동하기 위해 데모 환경을 구현하는 형식을 제공하는 파일. 이 환경에는 점령과 방어 게임 형식의 두 팀에 속한 플레이어가 모두 포함됩니다.

3.2 Flash

HUD 키트에서 사용되는 Flash 콘텐츠는 **HUDKit.fla** 와 **Minimap.fla** 두 개의 파일로 나뉩니다. 이들 FLA 파일은 Scaleform 를 사용한 조작을 위해 HUD 키트의 구성 요소를 정의하고 배치합니다. 또한, UI 에 표시되는 모든 이미지 및 아이콘을 포함합니다.

두 파일 모두 여러 층의 레이어로 나뉩니다. 기본적으로 각 구성 요소 또는 유사한 구성 요소의 집합은 고유한 레이어를 가집니다. 레이어는 층 순서에 따라 배열된 구성 요소의 author-time 컨트롤을 위한 편리한 방법을 제공합니다. 가장 위에 있는 레이어는 다른 모든 레이어보다 위에 표시됩니다.

Scaleform 에서 Flash 애니메이션은 C++, ActionScript 또는 Flash 타임라인을 통해 제어할 수 있습니다. 본 데모에서 대부분의 HUD 애니메이션은 Classic Tweens 를 통해 Flash 타임라인에서 제어되었습니다. 이러한 애니메이션은 보통 C++에서 호출하는 GFx::Value::GotoAndPlay()로 시작됩니다.

3.2.1 HUDKit.fla

Bin\Data\AS2 or AS3\Kits\HUD 디렉터리에 위치한 HUDKit.fla 파일은 본 데모의 주요한 SWF 입니다. 실행 시 HUDKitDemo 응용 프로그램에서 이 파일을 불러옵니다. 모든 Flash UI 구성 요소는 이 파일에 존재합니다. 단, 실행 시 HUDKit.fla 에서 불러오는 미니맵 뷰는 예외입니다.

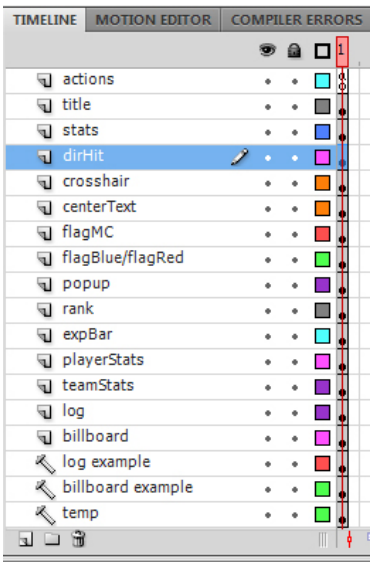


그림 2: HUDKit.fla 레이어

HUDKitDemo 는 타임라인의 Frame 1 을 호출하는 External Interface 를 사용하여 HUDKit MovieClip 을 등록합니다.

```
ExternalInterface.call("registerHUDView", this);
```

이것은 C++이 C++HUDView 에 등록할 수 있는 MovieClip 으로 포인터를 하나 전달하여 사용자 인터페이스를 처리합니다.

모든 레이어 및 레이어의 MovieClips, 레이아웃, 애니메이션과 Scaleform C++ 관리자는 HUD 뷰 섹션에서 보다 자세하게 다루도록 하겠습니다.

3.2.2 Minimap.fla

그림 3: Minimap 심볼

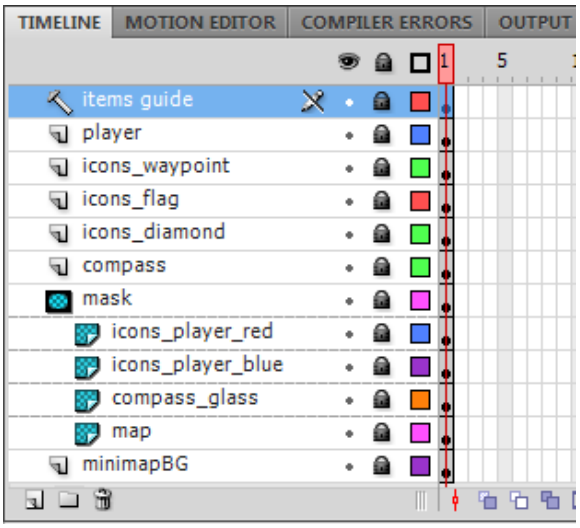
Minimap.fla 에는 미니맵의 핵심 요소인 '미니맵' 심볼이 포함됩니다. 본 데모의 스코프에 사용된 배경은 스태틱 이미지이며 인터랙티브한 3D 환경은 아닙니다.

플레이어는 스코프 중앙에 노란 화살표로 표시되고, 방위는 미니맵의 보더로 확인할 수 있습니다(북쪽이 표시되어 있음). 미니맵 뷰에는 플레이어 외에도 5 가지 아이콘 타입이 표시됩니다.

- *아군 플레이어(청색).* 방향 화살표는 높은 줌 레벨에서만 나타납니다.
- *적 플레이어(적색).* 방향 화살표는 높은 줌 레벨에서만 나타납니다.
- *점령 지점*(깃발 표시: 흰색은 중립, 청색은 아군, 적색은 적군)
- *파워업*(회전하는 녹색과 흰색 아이콘)

점령 지점과 파워업은 탐색 가능한 범위를 벗어나 보이는 영역 바깥쪽에 있을 때 뷰 보더에 고정되는 고정 아이콘입니다. Direct Access API 메소드는 탐색 가능한 범위에 들어오거나 해당 범위에서 벗어나는 아이콘을 화면에서 지우는 기능을 제공합니다. C++를 사용하여 런타임 동안 스테이지에서 MovieClips 를 붙이거나 제거하는 데도 사용할 수 있습니다.

이러한 '미니맵' 심볼은 여러 개의 레이어로 구성됩니다. 레이어는 층 순서에 따라 배열된 구성 요소의 author-time 컨트롤을 위한 편리한 방법을 제공합니다. 가장 위에 있는 레이어는 다른 모든 레이어보다 위에 표시됩니다.



그 4: '미니맵' 심볼을 구성하는 레이어

- **items guide**: 디자이너 및 아티스트의 작업을 도와주는 지시 레이어입니다. 이 레이어는 SWF 파일에 포함되지 않습니다.
- **player**: 노란색 플레이어 아이콘이 속한 레이어입니다.
- **icons_waypoint**: 웨이포인트 아이콘이 생성되고 캐시되는 빈 캔버스입니다.
- **icons_flag**: 깃발 아이콘이 생성되고 캐시되는 빈 캔버스입니다.
- **icons_diamond**: 파워업 다이아몬드 아이콘이 생성되고 캐시되는 빈 캔버스입니다.
- **compass**: 나침반(북쪽이 표시되지 않음)이 속한 레이어입니다.
- **mask**: 마스크 해제된 영역의 콘텐츠만 보여주는 마스크 레이어입니다.
    - **icons_player_red**: 적 아이콘(적색)이 생성되고 캐시되는 빈 캔버스입니다.
    - **icons_player_blue**: 아군 아이콘(청색)이 생성되고 캐시되는 빈 캔버스입니다.
    - **compass_glass**: 나침반 좌측 상단에 광택 효과를 부여하는 장식 레이어입니다.
    - **map**: 지형지도가 속한 레이어입니다. 본 데모에서는 지형 표현을 위해 대형 비트맵 이미지를 사용하였으나, 실제 제작 상황에서는 대체로 C++에서 업데이트된 텍스처를 사용하게 됩니다. 이 경우, 업데이트 로직은 마스킹된 지형지도를 사용할 때 필요한 지도의 오프셋 및 회전을 변경할 필요가 없습니다.
- **minimapBG**: 지형 비트맵 모서리와 잘 혼합된 청색 배경이 속한 레이어입니다. 이 레이어는 지형 비트맵이 시야 밖으로 이동할 때 부드럽게 전환되도록 해줍니다. 지형지도가 텍스처로 렌더링되어 C++에서 직접 업데이트할 경우, 이 배경은 필요하지 않습니다.

'미니맵' 심볼은 *Bin\Data\AS2 or AS3\Kits\HUD\com\scaleform* 에 위치한 com.scaleform.Minimap 클래스를 사용합니다. 이 최소화 클래스는 미니맵을 HUDKitDemo 응용 프로그램에 등록하고 지도 이미지를 로드하는 미니맵 생성자 및 지도 이미지에 경로를 정의합니다.

4 HUD 뷰

C++ HUD 뷰는 콘텐츠인 HUDKit.swf 를 제어합니다. HUD 뷰의 인터페이스는 다음 형식을 선언합니다.

- FxHUDView – HUD 의 코어 매니저(특히, HUDKit.swf)입니다. 시뮬레이션 상태에 근거하여 뷰와 모든 자식 MovieClips 를 업데이트합니다.
- BillboardCache – 적/아군 빌보드(미니맵 아이콘과는 다른 중앙 및 화면상의 지시기)의 매니저 및 캐시 시스템입니다.
- FxHUDLog – HUD 왼쪽에 표시되는 메시지 로그 매니저입니다. FxHUDMessages 를 추적 및 표시하고 재사용합니다.
- FxHUDMessage – MovieClip 레퍼런스와 메시지 텍스트가 포함된 메시지 정의입니다.

FxHUDView 의 캐시된 MovieClip 레퍼런스는 MC(MovieClip 의 단축) 접미사로 표시되지 않습니다.

초기화가 진행되는 동안 FxHUDView 는 레퍼런스를 런타임 시 UpdateView()에서 변경된 모든 주요 MovieClip 에 등록합니다. 이것은 매 업데이트 시 MovieClip 을 검색하지 않도록 하여, 뷰의 업데이트에 걸리는 시간을 최소화하기 위해 중요합니다. 또한, 초기화 과정에서 HUDKit.fla 에 정의된 일부 사용하지 않는 UI 구성 요소가 숨겨집니다.

FxHUDView::UpdateView()는 시뮬레이션 환경에서 제공된 데이터로 뷰의 모든 구성요소를 업데이트합니다. 해당 로직은 다음의 하위 메소드로 나뉩니다. UpdateTeamStats(), UpdatePlayerEXP(), Update PlayerStats(), UpdateEvents(), UpdateTutorial() 및 UpdateBillboards(). 이들 메소드는 모두 시뮬레이션 환경으로 포인터를 전달하며, 포인터를 전달받은 시뮬레이션 환경은 특정한 UI 구성요소를 업데이트하는 데 사용됩니다.

UpdateView()가 호출하는 동안 MovieClip 레퍼런스를 검색하는 일은 거의 없습니다. 매 프레임 MovieClip 을 업데이트하는 것은 지속적인 자원 낭비의 가능성이 있으므로 불필요한 MovieClips 업데이트는 실행하지 않습니다.

Scaleform Direct Access API 가 출시됨으로써 Flash 와 응용 프로그램 간에 주고 받는 모든 데이터의 흐름이 효율화되었습니다. 특히, GFx::Value 클래스에는 Scaleform 사용자들의 Flash 콘텐츠 제어를 도와주는 새로운 함수가 몇 가지 제공됩니다. GFx::Value::SetDisplayInfo(), GFx::Value::GetDisplayInfo()와 GFx::Value::SetText()는 MovieClips 표시 상태를 직접적이고 효과적으로 제어하기 위해 HUD 뷰 업데이트 로직 전반에 걸쳐 사용됩니다.

4.1 라운드 통계 및 점수판

teamStats MovieClip 는 HUDKit.fla 의 Scene 1 에 속한 teamStats 레이어에 있으며, 진행 중인 라운드의 점수판 및 통계입니다. 여기에는 승리와 패배 textFields(*redWinning*, *blueWinning*), 두 textField 에 출력되는 각 팀의 점수(*scoreRed*, *scoreBlue*), 그리고 진행 표시줄(*teamRed, teamBlue)* 및 roundtime clock textField(*roundTime*)이 포함됩니다.

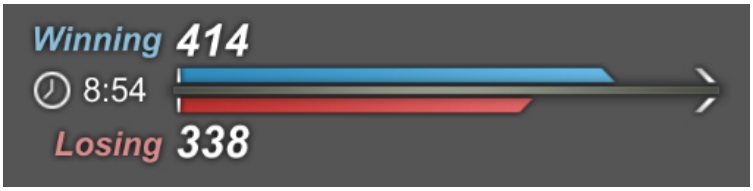


그림 5: teamStats 심볼

teamStats 는 배타적으로 FxHUDView::UpdateTeamStats()를 사용하여 업데이트됩니다. 다음은 *teamStats* MovieClip 의 구성요소를 업데이트하는 코드입니다.

```
void FxHUDView::UpdateTeamStats(FxHUDEnvironment *penv)
{
       // Retrieve the scores from the simulation.
       unsigned scoreBlue = unsigned(penv->GetHUDBlueTeamScore());
       unsigned scoreRed = unsigned(penv->GetHUDRedTeamScore());

       String text;
       Value tf;
       Value::DisplayInfo info;

       // We won't update any element who has not changed since the last update
       // to avoid unnecessary updates.  To do so, we compare the previous
     // simulation State to the latest simulation data.  If the two are different,
     // update the UI element in the HUD View.
       if (State.ScoreRed != scoreRed)
       {
              Format(text, "{0}", scoreRed);
              ScoreRedMC.SetText(text); // Update the red team score text.

              info.SetScale((scoreRed / (Double)MaxScore) * 100, 100);
              TeamRedMC.SetDisplayInfo(info); // Update and scale the red team score
                                              bar movieClip.
       }

       if (State.ScoreBlue != scoreBlue)
       {
              Format(text, "{0}", scoreBlue);
              ScoreBlueMC.SetText(text); // Update the blue team score text.

              info.SetScale((scoreBlue / (Double)MaxScore) * 100, 100);
              TeamBlueMC.SetDisplayInfo(info); // Update and scale the blue team
                                               score bar movieClip.
```

```
        }

        if (State.ScoreRed != scoreRed || State.ScoreBlue != scoreBlue)
        {
                // Update the "Winning" and "Losing" text fields for both teams
            // if the score has changed.
                if (scoreBlue > scoreRed)
                {
                        RedWinningMC.SetText(LosingText);
                        BlueWinningMC.SetText(WinningText);
                }
                else
                {
                        RedWinningMC.SetText(WinningText);
                        BlueWinningMC.SetText(LosingText);
                }
        }

        // Update the roundtime clock
        float secondsLeft = penv->GetSecondsLeftInRound();
        unsigned sec = ((unsigned)secondsLeft % 60);
        if (sec != State.Sec)
        {
                unsigned min = ((unsigned)secondsLeft / 60);
                String timeLeft;
                Format(timeLeft, "{0}:{1:0.2}", min, sec);
                RoundTimeMC.SetText(timeLeft);
                State.Sec = sec;
        }
}
```

점수는 Scaleform::String 에 저장되고, GFx::Value::SetText()를 통해 *scoreRed* 및 *scoreBlue* textField 로 전달됩니다. teamRed 및 teamBlue 점수 표시줄의 X scale 은 GFx::Value::SetDisplayInfo()를 사용하여 업데이트됩니다. 양 팀 모두 점수 표시줄 원래 길이의 0% 지점에서 시작합니다. 해당 라운드에서 승리하는 데 필요한 점수를 표시줄 원래 길이의 100%로 산정하여, 라운드를 진행하며 획득한 점수만큼 100% 지점 쪽으로 점수 표시줄의 길이가 늘어납니다.

blueWinning textField 와 *redWinning* textField 를 업데이트 하기 전에 양팀의 점수를 비교하여 불필요한 업데이트를 배제합니다. HUDView 가 동기화되는 동안 정의된 상수 "Winning"과 "Losing" 스트링을 사용하여 텍스트를 지정하기 위해 GFx::Value::SetText()를 호출합니다.

라운드 시간 textField 를 업데이트하기 전에, 마지막으로 UpdateView()를 호출했을 때 남은 시간을 가장 최근 시간과 비교합니다. 마지막 업데이트 이후로 1 초 이상 시간이 경과되었으면, GFx::Value::SetText()를 사용하여 *roundTime* textField 를 포맷된 시간 String 으로 업데이트합니다.

4.2 Player Stats 와 Ammo

playerStats 심볼은 HUDKit.fla 의 Scene 1 에 속한 playerStats 레이어에 있으며, 체력 textField 와 체력 표시줄 셰이프(*healthN, health),* 무기 아이콘 심볼(*weapon*), HUD 의 총탄 표시기, 그리고 총탄 부족 표시기(*reload*)가 포함됩니다.

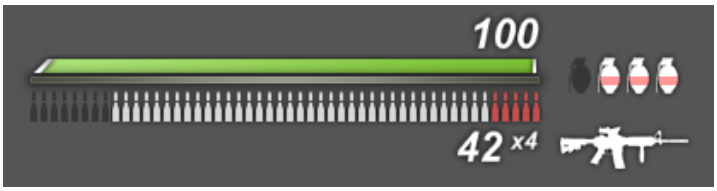


그림 6: playerStats 심볼

총탄 표시기는 여섯 개의 MovieClips(*clipN, ammoN, ammo, ammoBG, grenade,grenadeBG*)으로 나뉩니다. textFields *clipN* 과 *ammoN* 는 해당 무기의 탄창클립과 장전된 총탄 형태로 총탄의 잔량을 표시합니다.

ammo/healthVehicle 레이어는 두 개의 심볼(*ammo*, *ammoBG*)를 포함하며, 각각 적절한 하위 레이어로 나뉩니다. 두 심볼에는 모두 체력 표시줄 하단의 총탄 표시기 셰이프가 포함되어 있습니다. 이들 심볼은 각각 51 개의 Keyframe 으로 나뉘며, 개별 Keyframe 에는 이전의 Keyframe 보다 총탄 수가 하나 적은 총탄 표시기가 포함됩니다. GFx::Value::GotoAndStop()는 이들 MovieClip 의 각기 다른 Keyframe 에 액세스하고 표시된 셰이프의 개수를 변경하여 시뮬레이션 상태를 반영하기 위해 사용됩니다.

- *ammo*는 장비한 무기의 장전된 총탄 개수를 표시합니다.
- *ammoBG*는 장비한 무기의 클립에 들어갈 수 있는 최대 총탄 개수를 표시합니다.

The grenade/armor 레이어는 ammo 레이어와 비슷하게 두 개의 심볼(*grenade, grenadeBG*)를 포함하며, 각각 고유한 레이어로 나뉩니다. 두 심볼에는 모두 수류탄 표시기 셰이프가 포함되어 있습니다. 총탄 표시기와 비슷하게 이들 심볼은 각각 5 개의 Keyframe 으로 나뉩니다. Frame 1 에는 수류탄 셰이프가 4 개 있으며, Frame 5 에는 없습니다. GFx::Value::GotoAndStop()는 이들 MovieClip 의 각기 다른 Keyframe 에 액세스하고 표시된 모양의 개수를 변경하여 시뮬레이션 상태를 반영하기 위해 사용됩니다.

- *grenade* displays the number of grenades remaining.
- *grenadeBG*는 플레이어가 소지할 수 있는 수류탄의 최대 개수를 표시합니다.

다음은 *playerStats* 심볼을 업데이트하는 로직입니다.

```
void FxHUDView::UpdatePlayerStats(FxHUDEnvironment *penv)
{
     FxHUDPlayer* pplayer = penv->GetHUDPlayer();

       // Load latest player info.
       unsigned ammoInClip = pplayer->GetNumAmmoInClip();
       unsigned ammoClips  = pplayer->GetNumClips();
       unsigned clipSize   = pplayer->GetMaxAmmoInClip();
       unsigned grenades   = pplayer->GetNumGrenades();

       unsigned ammoFrames = 51;
       unsigned grenadeFrames = 5;

       String text;

       if(pplayer->GetHealth() != State.Health)
       {
              unsigned health = unsigned(pplayer->GetHealth() * 100);
              Format(text, "{0}", health);
              HealthNMC.SetText(text); // Update the health text field.

              Value::DisplayInfo info;
              info.SetScale(Double(health), 100);
              HealthMC.SetDisplayInfo(info); // Update and scale the health bar.
       }

       if(ammoInClip != State.AmmoInClip)
       {
              Format(text, "{0}", ammoInClip);
              AmmoNMC.SetText(text); // Update the ammo text field.

              // The ammo is divided into two parts: the white bullet icons (AmmoMC)
              // and their respective backgrounds (AmmoBGMC).

              // To update the number of bullets currently in the clip, we use
              // GotoAndStop with AmmoMC and select the frame with the proper number
            // of white bullet icons.  There are 51 frames in AmmoMC (defined
above
             // in ammoFrames)and the first frame displays every bullet icon.

          // To display X number of bullets, we subtract X from the total number
          // of frames in AmmoMC (51) and GotoAndStop on the result.
              AmmoMC.GotoAndStop(ammoFrames - ammoInClip);
       }

       if(ammoClips != State.AmmoClips)
       {
              Format(text, "x{0}", ammoClips);
              ClipNMC.SetText(text); // Update the remaining ammo clips text field.
       }

       if(grenades != State.Grenades)
       {
            // Grenades are setup similar to Ammo.  The first frame for the
            // Grenades movieClip shows 4 white grenade icons.  The second frame
```

```
            // shows 3.

            // To display X number of grenades, we subtract X from the total number
            // of frames in GrenadeMC (5) and GotoAndStop on the result.
           // Note: GrenadeMC actually contains 10 frames but 6-10 are unused by
           // this implementation.
               GrenadeMC.GotoAndStop(grenadeFrames - grenades);
        }

        if (pplayer->GetWeaponType() != State.weaponType)
        {
               // Change the weapon icon if the player has changed weapons.
               unsigned weaponFrame = GetWeaponFrame(pplayer->GetWeaponType());
               WeaponMC.GotoAndStop(weaponFrame);

               // Update the ammo background icons based on clipSize.
               if (clipSize != State.AmmoClipSize)
                      AmmoBGMC.GotoAndStop(ammoFrames - clipSize);
        }

        // If the ammo in clip is less than 6 bullets, play the "Reload" indicator
       // movieClip.
        if (ammoInClip <= 5)
        {
               // ReloadMC will play until we GotoAndStop on Frame 1, so no need to
          //  start the animation more than once.
               if (!bReloadMCVisible)
               {
                      bReloadMCVisible = true;
                      ReloadMC.GotoAndPlay("on"); // Will cycle back to "on" frame
when
                                                  it reaches the end and play again.
               }
        }

        // If the reload indicator is displayed but there are more than six bullets
        // hide it and stop the animation by playing frame 1 of the movieClip.
        else if (bReloadMCVisible)
        {
               ReloadMC.GotoAndStop("1"); // Frame 1 contains stop();
               bReloadMCVisible = false;
        }
}
```

ammo 에서 Frame 1 에는 50 개의 총탄 표시기가 있으며, Frame 51 에는 총탄 표시기가 없습니다. 플레이어가 무기를 발사할 때마다 적절한 Keyframe 에 액세스하여 HUD 에서 장전된 총탄 표시기를 하나 제거하기 위해 ammoMC.GotoAndStop(ammoFrames - ammoInClip)를 호출합니다.

ammoBG 의 경우, 총탄 표시기 배경의 X 숫자를 표시하는 Keyframe 에 액세스하는 데 ammoBGMC.GotoAndStop()가 사용됩니다. 여기서 X 는 장착한 무기의 클립 크기에 따라 다릅니다. 예를 들어 기관총의 클립 크기는 50 이므로, ammoBG.GotoAndStop(1)는 Frame 1 에 50 개의 총탄

표시기 배경을 모두 보여줍니다. 이 MovieClip 은 마지막으로 FxHUDView::UpdateView()을 호출한 이후로 장착 무기를 변경했을 때만 업데이트됩니다.

4.3 깃발 포획 표시기

The *flagMC* 심볼은 HUDKit.fla 의 Scene 1 에 속한 flag 레이어에 있으며, 깃발 포획 표시기의 컨테이너입니다. *flagMC* 에 포함된 *flag* 심볼은 깃발 포획 표시기 비트맵과 화살표 표시기(*arrow*) 두 부분으로 나뉘며, 화살표 표시기는 깃발의 CaptureState 에 따라 좌우로 움직입니다. 표시기는 플레이어가 포획 대상에 근접하면 페이드인 되고 범위에서 이탈하면 페이드아웃 됩니다.

그림 7: flagMC 심볼

flagMC 의 페이드인과 페이드 아웃 애니메이션은 Classic Tween 을 사용하는 Flash 타임라인에서 이루어집니다. *flagMC* 의 타임라인은 5 개의 Keyframe 으로 구성된 4 개의 상징적인 상태를 가집니다. Hidden, Visible, Fade-In, FadeOut. 이들 상태는 GotoAndStop()와 Keyframes 의 라벨/숫자(1, 5, "On", "Off", 등)를 사용하여 액세스합니다.

아래에 제시된 FxHUDView::UpdateEvents()는 시뮬레이션 데이터를 사용하여 *flagMC* 의 상태를 제어하고 화살표 방향을 업데이트합니다.

```
// Flag capture indicators and reticule behavior.
void FxHUDView::UpdateEvents(FxHUDEnvironment *penv)
{
	Value::DisplayInfo info;

	if (penv->IsHUDPlayerCapturingObjective())
	{
		if(!bFlagMCVisible)
		{
			SetVisible(&FlagMC, true); // Set the Flag MovieClip's
                                       visibility to true.
			FlagMC.GotoAndPlay("on"); // Play the fade-in animation once.
			bFlagMCVisible = true;
		}

		// Shift the x-coordinate of the Flag Arrow to show the capture state
		// of the flag the player is currently capturing.
		info.SetX(penv->GetHUDCaptureObjectiveState() * 177);
		FlagArrowMC.SetDisplayInfo(info);
```

```
            info.Clear();
        }
        else if (bFlagMCVisible & !penv->IsHUDPlayerCapturingObjective())
        {
            // If the flag indicator is still visible and the player is
            // no longer capturing an objective, play the fade-out animation
            FlagMC.GotoAndPlay("off");
            bFlagMCVisible = false;
        }
}
```

flagMC.GotoAndPlay("on")가 호출되면 *flagMC*는 페이드인 애니메이션을 재생하고, Visible 상태에서 재생을 멈춥니다. flagMC.GotoAndPlay("off")가 호출되면 *flagMC*는 11~20 번 프레임(페이드아웃 애니메이션)을 재생합니다. 애니메이션이 Frame 20 에 도달하면 자동으로 Frame 1 부터 재생을 다시 시작합니다. 이것은 Flash 의 기본적인 성질이며, ActionScript 나 C++로 추가적인 작업을 할 필요는 없습니다. Frame 1 에는 stop(); Frame 1 의 재생이 끝나면 MovieClip 이 중지됩니다. 심볼은 Alpha 가 0 일 때 Frame 1 속에 숨겨져 있으므로 이는 이상적인 현상입니다.

*arrow*는 GFx::Value::SetDisplayInfo()를 사용하여 수평으로 옮겨집니다. 이 메소드는 개체의 CaptureState 를 바탕으로 업데이트된 x 좌표와 함께 새 GFx::Value::DisplayInfo 를 전달합니다.

4.4 무기 조준점

reticule 심볼은 HUDKit.fla 의 Scene 1 에 속한 crosshair 레이어에 있으며, 발사 조준점의 컨테이너입니다. *reticule*에는 *left, right, bottom,top*의 네 가지 심볼이 포함되며, 각각 고유의 레이어에 속합니다. 조준점 조각이 각각 고유한 심볼에 속해 있어서 각 조각을 개별적으로 제어할 수 있습니다. 조준점은 사용자의 플레이어가 무기를 발사할 때마다 시뮬레이션에 의해 발사되는 PlayerFire 이벤트에 응답합니다. 플레이어의 무기 발사에 따라 조준점은 다이내믹하게 확장됩니다. 발사를 멈추면 조준점은 원래 위치로 돌아옵니다.

그림 8: reticule 심볼

조준점 업데이트 코드는 FxHUDView::UpdateEvents() 메소드와 FxHUDView::OnEvent() 메소드의 두 부분으로 구성됩니다. PlayerFire 이벤트가 확인되면 FxHUDEvent 에서는 ReticuleHeat 와 ReticuleFiringOffset 이 개별적으로 증가하고 지정됩니다.

```
// If the player is firing his/her weapon, update the reticule's heat.
// The reticule elements are updated in FxHUDView::UpdateEvents() based on
// the ReticleHeat and the ReticleFiringOffset.
case FxHUDEvent::EVT_PlayerFire:
{
       if (ReticleHeat < 8)
              ReticleHeat += 2.5f;

      ReticleFiringOffset = 2;
}
```

FxHUDView::UpdateEvents()는 변수 두 개를 사용하여 MovieClip 을 top, bottom, left, right 로 움직입니다. 여기서는 SetDisplayInfo()를 사용하여 각 MovieClip 의 X 또는 Y 좌표를 움직입니다. 해당 방정식의 첫 번째 숫자(6 또는 -6)는 0, 0 좌표 상의 시작 X 오프셋 또는 시작 Y 오프셋입니다. 각각의 SetDisplayInfo() 호출에 대해 GFx::Value::DisplayInfo 의 단일 인스턴스가 재사용됨을 주의하십시오. 하지만, DisplayInfo 가 잘못 재사용되는 일이 없도록 하기 위해 DisplayInfo.Clear()는 각각의 GFx::Value::SetDisplayInfo() 호출이 끝난 다음에 호출됩니다. 이 메소드는 DisplayInfo 인스턴스를 위해 사전에 정의된 속성을 모두 제거합니다.

```
// Shift the XY-coordinates of the reticule elements based on the
// firing duration and rate of fire.
if (ReticleHeat > 0 || ReticleFiringOffset > 0){
       if (ReticleHeat > 1.0f)
              ReticleHeat -= .02f;

       ReticleFiringOffset -= .02f;
       if (ReticleFiringOffset < 0)
            ReticleFiringOffset = 0;

       info.SetY((-6) - (ReticleFiringOffset + ReticleHeat));
       ReticuleMC_Top.SetDisplayInfo(info);
       info.Clear();

       info.SetX((6) + (ReticleFiringOffset + ReticleHeat));
       ReticuleMC_Right.SetDisplayInfo(info);
       info.Clear();

       info.SetX((-6) - (ReticleFiringOffset + ReticleHeat));
       ReticuleMC_Left.SetDisplayInfo(info);
       info.Clear();

       info.SetY((6) + (ReticleFiringOffset + ReticleHeat));
       ReticuleMC_Bottom.SetDisplayInfo(info);
       info.Clear();
}
```

본 데모에서는 단순한 연출을 위해 간단한 동적 조준점 방정식을 사용했으나, 이 코드를 수정하여 더욱 박력 있고 창조적인 조준점 동작을 연출할 수 있습니다.

4.5 방향별 타격 표시기

dirHit 심볼은 HUDKit.fla 의 Scene 1 에 속한 dirHit 레이어에 있으며, 방향별 타격 표시기의 컨테이너입니다. *dirHit* 에 포함된 8 개의 MovieClip 은 적당한 이름이 붙은 레이어로 나뉘며 다음과 같습니다. *tl, l, bl, b, br, r, tr, t*.

이들 MovieClip 에는 각각 알파 블렌딩 처리된 빨간색 반원이 포함되며, 이 반원은 다음 사항에 대한 표시기로 사용됩니다.

a) 유저 플레이어가 받는 데미지
b) 데미지가 들어오는 곳의 방향

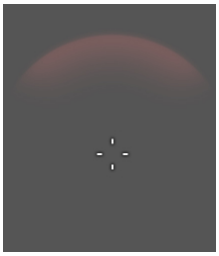


그림 9: dirHit 심볼

플레이어가 타격을 받으면 타격 표시기가 나타나며, 최대 1 초 후에 사라집니다. 해당 페이드아웃 애니메이션은 Flash 타임라인에서 알파 블렌딩 처리된 Classic Tween 입니다. 모든 타격 표시기 MovieClip 은 GFx::Value::GotoAndPlay("on")의 호출에 의해 나타났다가 1 초 이후에 페이드아웃 됩니다. 데미지가 들어오는 방향에 따라 알맞은 MovieClip 이 나타납니다.

```
void FxHUDView::OnEvent( FxHUDEvent* pevent )
{
	switch ( pevent->GetType() )
	{
		// If the Damage Event (Player was hit) is fired, show the appropriate
	    // directional hit indicator.
		// We can use GotoAndPlay("on") to play the fade-in animation.  It
will
	    // fade out on its own  after ~1 second (this animation is setup on the
	    // Flash timeline)
		case FxHUDEvent::EVT_Damage:
		{
			FxHUDDamageEvent* peventDamage = (FxHUDDamageEvent*)pevent;
			float dir = peventDamage->GetDirection();
			if (dir > 0)
			{
				if (dir > 90)
				{
					if (dir > 135)
```

```
                                        DirMC_B.GotoAndPlay("on");
                                else
                                        DirMC_BR.GotoAndPlay("on");
                        }
                        else
                        {
                                if (dir > 45)
                                        DirMC_TR.GotoAndPlay("on");
                                else
                                        DirMC_T.GotoAndPlay("on");
                        }
                }
                else
                {
                        if (dir < -90)
                        {
                                if (dir < -135)
                                        DirMC_B.GotoAndPlay("on");
                                else
                                        DirMC_BL.GotoAndPlay("on");
                        }
                        else
                        {
                                if (dir < -45)
                                        DirMC_L.GotoAndPlay("on");
                                else
                                        DirMC_TL.GotoAndPlay("on");
                        }
                }
            break;
        }
}
```

4.6 순위 및 경험치 디스플레이

expBar 심볼은 HUDKit.fla 의 Scene 1 에 속한 expBar 레이어에 있으며, 플레이어의 현재 순위를 통해 진행 상황을 추적하는 화면 하단의 경험치 표시줄입니다. *expBar* 에는 *exp* MovieClip 가 포함되는데, 이것은 플레이어가 적을 죽이거나 점령을 통해 경험치를 획득할 때 길어지는 노란색 막대입니다.

rank 심볼은 HUDKit.fla 의 Scene 1 에 속한 rank 레이어에 있으며, 플레이어의 현재 순위를 보여줍니다. *rank* 에는 18 개의 Keyframe 이 포함되며, 각각 하나의 순위를 표시하는 데 사용됩니다. 각각의 Keyframe 은 순위와 연결된 아이콘 및 타이틀을 표시합니다.

그림 10: expBar 및 rank 심볼

```
void FxHUDView::UpdatePlayerEXP(FxHUDEnvironment *penv)
{
	FxHUDPlayer* pplayer = penv->GetHUDPlayer();
	if (pplayer->GetLevelXP() != State.Exp)
	{
		Value::DisplayInfo info;
		// Update the rank icon.  Each rank icon is on a different frame of
	   // the rank movieClip.
		RankMC.GotoAndStop(unsigned(pplayer->GetRank() + 1));
		info.SetScale(pplayer->GetLevelXP() * 100, 100);  // Scale the EXP bar
                                                          movieClip.
		ExpBarMC.SetDisplayInfo(info);
	}
}
```

exp 는 팀 점수 표시줄과 비슷하게 플레이어의 경험치가 변경될 때 현재 경험치를 바탕으로 증가합니다. 순위를 업데이트하기 위해 Keyframe X 에 GFx::Value::GotoAndStop()가 사용됩니다. Frame 0 은 존재하지 않으므로 X 의 값은 플레이어의 현재 순위 + 1 이 됩니다.

4.7 텍스트 알림

The *centerTextMC* 심볼은 HUDKit.fla 의 Scene 1 에 속한 centerText 레이어에 있으며, 화면 중앙에 사용자 알림을 표시하기 위해 사용되는 애니메이션 textField 입니다. 본 데모에서는 매 라운드 시작 시 간단한 튜토리얼을 표시하고 최근에 죽은 플레이어를 알리는 용도로 *centerText* 가 사용되었습니다. 텍스트는 출력된 다음에 페이드아웃됩니다.

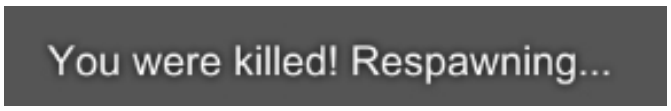


그림 11: centerTextMC 심볼

centerTextMC 에는 실제 textField 가 포함된 *centerText* MovieClip 인 *textField* 가 포함됩니다. 해당 MovieClip 은 애니메이션 출력을 위해 이런 식으로 정렬되어 있습니다. *centerTextMC* 의 타임라인에는 "on5"과 "on" 두 개의 Keyframe 이 있는데, labeled 는 페이드아웃 애니메이션을 재생하기 전에 5 초간 텍스트를 표시하고, "on"은 페이드아웃 애니메이션을 재생하기 전에 3 초간 텍스트를 표시합니다. 타임라인이 끝나면 *centerTextMC* 의 alpha 가 0 인 Frame 1 에서 해당 MovieClip 재생이 다시 시작되며, "on" 또는 "on5"가 다시 호출될 때까지 MovieClip 을 감춥니다.

```
TextFieldMC.SetText(RespawnText); // Set the text field of the Center Text.
CenterTextMC.GotoAndPlay("on"); // Play the fade-in animation, it will fade on its
                                own after ~3 sec.
```

여기서 *centerTextMC* 의 *textField* 는 적당하게 설정되었고, 해당 MovieClip 은 GotoAndPlay("on")를 사용하여 표시됩니다. 타임라인의 Classic Tween 때문에 MovieClip 은 최대 3 초 후에 페이드아웃 됩니다.

4.8 팝업 및 알림

popup 심볼은 HUDKit.fla 의 Scene 1 에 속한 popup 레이어에 있으며, 애니메이션 이벤트 표시기입니다. 본 데모에서는 유저 플레이어가 3 명 이상을 연속으로 죽일 때(연속 킬)마다 *popup* 이 재생됩니다. 표시될 때는 페이드인 되어 팝업 애니메이션을 재생하며, C++ 호출에 따라 3~5 초 후에 페이드아웃 됩니다.

그림 12: popup 심볼

Popup 은 팝업의 코어 컨테이너입니다. 하위 구성요소에 대한 모든 애니메이션은 Classic Tweens 를 사용하여 *popup* 의 타임라인에서 이루어집니다. *popup* 에는 *actions, popupNumber, popupText,popupBG* 4 개의 레이어가 포함됩니다. *popupBG* 은 숫자 뒤에서 "폭발"하는 검은색 셰이프입니다. *popupText* 와 popupNumber 에는 팝업 메시지를 위해 텍스트와 숫자를 표시하는

textField 심볼이 포함됩니다. 이러한 심볼 체계는 해당 타임라인의 textFields 를 애니메이션 시키기 위해서 중요합니다.

해당 심볼에는 애니메이션 재생을 위해 GotoAndPlay()를 사용하여 호출할 수 있는 "on" Keyframe 이 포함됩니다. 처음 애니메이션이 완료되면 팝업이 페이드아웃 됩니다. 또, 재생 프레임이 Frame 1 로 반환되어 재생이 중지되고 Alpha 가 0 으로 지정되면서 사라집니다.

```
// If a KillStreak event is fired, display the kill streak pop-up.
case FxHUDEvent::EVT_KillStreak:
{
	FxHUDKillStreakEvent* peventKS = (FxHUDKillStreakEvent*)pevent;

	// Format the number of kills
	String text;
	Format(text, "{0}", peventKS->GetSrcKillStreak());
		PTextFieldMC.SetText(text); // Set the text field of the PopUp.

	// Play the fade-in animation, it will fade on its own after ~4 sec.
	PopupMC.GotoAndPlay("on");
}
break;
```

여기서는 FxHUDEvent 가 KillStreak 이벤트로 cast 됩니다. 연속 킬의 숫자는 플레이어로부터 획득하여 적절하게 포맷됩니다. 프리캐시된 *popupText* 의 *textField* 심볼인 PTextFieldMC 는 *popup* 디스플레이 애니메이션을 시작하는 데 사용되는 GotoAndPlay("on")과 C++을 통해서 지정됩니다.

4.9 메시지 및 이벤트 로그

log 심볼은 HUDKit.fla 의 Scene 1 에 속한 log 레이어에 있으며, HUD 의 좌측 하단에 표시되는 메시지와 이벤트 로그의 컨테이너입니다. 본 데모에서는 로그에 킬, 파워업, 깃발 점령 및 레벨 상승과 관련된 텍스트 메시지가 출력됩니다. 로그 아래쪽에 새로운 메시지가 추가되면서 이전 메시지를 위로 올립니다. 로그 메시지는 출력된 후 최대 3 초가 지나면 페이드아웃 됩니다.

그림 13: log 및 logMessage 심볼

로그 메시지는 *logMessage* 심볼의 인스턴스입니다. 로그에 추가된 메시지는 페이드아웃 애니메이션이 시작되기 전에 약 3 초간 출력됩니다. 이 애니메이션은 *logMessage* 심볼의 타임라인에 정의되어 있습니다. C++에서 메시지에 들어가는 텍스트를 보내며, 이 메시지는 htmlText 를 사용하여 textField 에 출력합니다.

다음은 `FxHUDMessage::Movieclip* FxHUDLog::GetUnusedMessageMovieclip()`의 일부분으로, 새 LogMessage MovieClip 을 생성하여 모든 로그 메시지를 위한 캔버스인 Log MovieClip 에 붙이는 코드입니다.

```
FxHUDMessage::Movieclip* FxHUDLog::GetUnusedMessageMovieclip()
{
    if (MessageMCs.IsEmpty())
    {
        // Request a new line in the SWF
        Value logline;
          LogMC.AttachMovie(&logline, "LogMessage", "logMessage"+NumMessages);
        FxHUDMessage::Movieclip* pmc = new FxHUDMessage::Movieclip(logline);
        MessageMCs.PushBack(pmc);  // Add new Message MC to list of unused
                                   message movieClips.
    }
     FxHUDMessage::Movieclip* pmc = MessageMCs.GetFirst();  // Use an unused Message
                                                               movieClip.
    pmc->SetVisible(true);
    MessageMCs.Remove(pmc);
    return pmc;
}
```

메시지의 내용은 이벤트 타입에 따라 C++에서 지정합니다. 다음은 Level Up 이벤트를 처리하고 로그에 "You are now a [Rank Icon] [Rank Name]"라는 새 메시지를 출력하기 위해 htmlText 를 지정하는 C++ 코드입니다.

```
void    FxHUDLog::Process(FxHUDEvent *pevent)
{
     String msg;
     switch (pevent->GetType())
     {
            case FxHUDEvent::EVT_LevelUp:
           {
                FxHUDLevelUpEvent* e = (FxHUDLevelUpEvent*)pevent;
                Format(msg, "You are now a <img src='rank{1}'/> {0}!",
                  e->GetNewRankName(), e->GetNewRank());
           }
           break;
      }

FxHUDMessage::Movieclip* pmc = GetUnusedMessageMovieclip();
FxHUDMessage* m = new FxHUDMessage(msg, pmc); // 3 seconds
```

```
Log.PushBack(m);
NumMessages++;
```

다음은 로그를 제어하기 위해 사용하는 HUD Update 로직입니다. 이 로직은 로그 메시지 레퍼런스에 대한 레이아웃과 애니메이션을 업데이트합니다.

```
void    FxHUDLog::Update()
{
      SF_ASSERT(LogMC.IsDisplayObject());

       Double rowHeight = 30.f;

       // Tick the message lifetimes
       Double yoff = 0;
       FxHUDMessage* data = Log.GetFirst();
       while (!Log.IsNull(data))
       {
       FxHUDMessage* next = Log.GetNext(data);
              if (data->IsAlive())
              {
              // Layout mgmt and animation
              Value::DisplayInfo info;
              info.SetY(yoff);
              data->GetMovieclip()->GetRef().SetDisplayInfo(info);
              }
              data = next;
              yoff += rowHeight;
       }

       // Layout management and animation
       Value::DisplayInfo info;
       info.SetY(LogOriginalY - (NumMessages * rowHeight));
       LogMC.SetDisplayInfo(info);

       // Remove dead messages
       data = Log.GetFirst();
       while (!Log.IsNull(data))
       {
              FxHUDMessage* next = Log.GetNext(data);
              if (!data->IsAlive())
              {
              Log.Remove(data);
              NumMessages--;

              FxHUDMessage::Movieclip* pmc = data->GetMovieclip();
              pmc->SetVisible(false);
              MessageMCs.PushBack(pmc);

              delete data;
              }
              data = next;
       }
}
```

로그에 대한 업데이트 로직은 FxHUDMessage 의 Alpha 가 0 으로 지정되었는지 확인하여 해당 로그에서의 현재 출력 여부를 확인합니다. 만약 출력되지 않았다면, FxHUDMessage 는 로그에서 제거되어 사용하지 않는 FxHUDMessage MovieClips 의 FxHUDMessage MovieClips 목록으로 되돌아간 것입니다. 새 FxHUDMessage 가 추가되었다면, 모든 MessageMC 는 GFx::Value::DisplayInfo.SetY()와 GFx::Value::SetDisplayInfo()를 사용하여 적절하게 위로 이동됩니다.

4.10 빌보드

HUDKit.fla 의 Scene 1 에 속한 billboard 레이어에 있는 *billboard_container* 는 플레이어용 빌보드 컨테이너입니다. 각 빌보드는 플레이어의 위치와 화살표, 플레이어 이름을 표시합니다. 빌보드는 게임에서 사용자 플레이어의 뷰 절두체에 속한 3D 물체의 추가 세부 사항을 표시하는 데 종종 사용됩니다. 이 데모에서는 사용자 플레이어로부터 일정 간격 거리에 플레이어 빌보드가 표시됩니다. 아군 플레이어 빌보드에는 항상 플레이어 이름이 표시되지만 적 빌보드는 대상이 시야 중앙으로 가까이 들어오기 전까지 화살표만 표시됩니다.

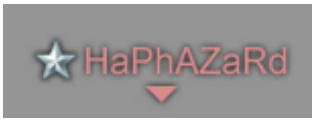


그림 1: billboard_enemy 심볼

이 데모에서 구현된 빌보드는 GFx::Value::SetDisplayInfo()를 사용하여 UI 에 플레이어의 위치에 따라 이동하거나 크기가 변경되는 2D 빌보드 MovieClip 을 생성합니다.

각 빌보드 MovieClip(*billboard_friendly, billboard_enemy)*은 텍스트 레이어에 위치한 하나의 *billboard_label_(friendly/enemy)*와 화살표 레이어에 위치한 하나의 화살표 이미지로 구성됩니다. *billboard_label_*은* 대상 플레이어의 이름이 표시되는 하나의 textField 를 가집니다. *billboard_friendly* 의 "Show" Keyframe 은 *billboard_friendly* 의 타임라인 상에서 Classic Tween 알파 블렌드를 사용하여 페이드 인 애니메이션을 시작합니다. 한 번 "Show"가 표시되면 빌보드는 숨김 또는 제거될 때까지 계속 표시됩니다.

빌보드 MovieClip 을 캔버스에 붙이는 C++ 로직은 `FxHUDView::BillboardCache::GetUnusedBillboardMovieclip` 에 있습니다.

```
if (UnusedBillboards.IsEmpty())
{
      Value::DisplayInfo info(false);
```

```
        Value billboard, temp;
        String instanceName;
        Format(instanceName, "{0}{1}", BillboardPrefix, BillboardCount++);
                    CanvasMC.AttachMovie(&billboard, SymbolLinkage, instanceName);

      BillboardMC* pbb = new BillboardMC();
      pbb->Billboard = billboard;
      billboard.GetMember("label", &temp);
      temp.GetMember("textField", &temp);
      pbb->Textfield = temp;

      pbb->Billboard.SetDisplayInfo(info);
      UnusedBillboards.PushBack(pbb);

}
```

각 BillboardCache 는 CanvasMC 라는 MovieClip 이 붙어있는 빌보드에 대한 레퍼런스를 담고 있습니다. 이 레퍼런스는 인스턴스화 될 때 BillboardCache 로 전달됩니다. 위 로직은 billboard_enemy/billboard_friendly 심볼의 빌보드 MovieClip 을 CanvasMC.AttachMovie() 호출에서 사전 정의된 캔버스에 붙이고 다시 사용할 수 있도록 레퍼런스를 보존합니다.

빌보드 업데이트 로직은 다음 메소드에 정의되어 있습니다. UpdateBillboards(), BeginProcessing(), EndProcessing() 및 GetUnusedBillboardMovieclip(). BeginProcessing() 및 EndProcessing()은 아군 또는 적 빌보드 세트 업데이트가 시작될 때 또는 종료될 때 호출됩니다. 이 메소드는 사용 및 미사용된 빌보드 목록을 관리합니다.

다음 코드는 UpdateBillboards()의 아군 빌보드 업데이트 로직입니다.

```
// Process friendlies
// Friendly billboards will always show names


BillboardMC* pbb = NULL;

FriendlyBillboards.BeginProcessing();
for (unsigned i=0; i < friendlies.GetSize(); i++)
{
       info.Clear();
      if (FriendlyBillboards.GetUnusedBillboardMovieclip(friendlies[i].pPlayer,
                                                          &pbb))
      {
        info.SetVisible(true);
            pbb->Billboard.GotoAndPlay("show");
            // Load player info
            String title;
            Format(title,
                        "<img src='rank{0}' width='20' height='20' align='baseline'
                        vspace='-7'/> {1}",
                          friendlies[i].pPlayer->GetRank()+1,
                       friendlies[i].pPlayer->GetName());
```

```
            pbb->Textfield.SetTextHTML(title);
        }
        info.SetX(friendlies[i].X);
        info.SetY(friendlies[i].Y);
        info.SetScale(friendlies[i].Scale, friendlies[i].Scale);
        pbb->Billboard.SetDisplayInfo(info);
}

FriendlyBillboards.EndProcessing();
```

UpdateBillboards()는 사용자 플레이어 열람 쿼리에서 검색한 목록을 기반으로 빌보드를 생성하고 업데이트합니다. 빌보드 디스플레이에는 GotoAndPlay("show")가 사용되며, MovieClip 의 X, Y 및 스케일 설정에는 GFx::Value::SetDisplayInfo()가, 표시되는 텍스트를 바꿀 때는 GFx::Value::SetText()가 사용됩니다.

textField 는 빌보드를 표시하고 배치할 때 htmlText 를 사용합니다. 텍스트 필드에는 플레이어의 순위 이미지와 이름이 포함됩니다. 순위 아이콘은 *src='rank{0}'* HTML 에 정의되어 있습니다. 위 예시에서는 *friendlies[i].pPlayer->GetRank()+1* 가 아군 플레이어 순위+1 에 해당합니다. 다른 *img* 옵션에서는 아이콘 이미지의 높이, 넓이, 위치 및 vspace 를 정의합니다. 플레이어의 이름은 {1}로 정의됩니다. 위 예시에서는 *friendlies[i].pPlayer->GetName()*가 아군 플레이어 이름에 해당합니다.

5 미니맵 뷰

미니맵의 C++ 관련 부분은 다음 파일로 구성됩니다. (*Apps\Kits\HUD*에 위치)

- **FxMinimap.h**: 게임 환경 및 응용 프로그램이 구현할 수 있는 인터페이스를 선언하여 미니맵 데모 코어에 연결하는 파일.
- **FxMinimapView.h/.cpp**: 미니맵 뷰에서 사용되는 형식을 제공하는 파일.

5.1.1 미니맵 뷰

미니맵 뷰 인터페이스에서는 다음 형식이 선언됩니다.

- **FxMinimapIconType**: 형식 ID 및 차후 전문화용 추가 하위 형식 값을 포함하는 아이콘 형식 정의.
- **FxMinimapIcon**: 뷰에 표시되는 요소에 연결되는 아이콘 리소스.
- **FxMinimapIconCache**: 특정 형식의 아이콘 세트를 저장하는 아이콘 캐시. 아이콘 뷰는 필요에 따라 캐시에서 관리되며 캐시 역시 탐색 가능 영역을 들어오거나 나갈 때 일어나는 페이드 인/아웃을 지원합니다. 캐시는 팩토리를 사용하여 새 아이콘을 생성합니다. 기본적인 팩토리 구현 방법은 MinimapIconFactoryBase 라는 클래스의 템플릿화입니다. 각 팩토리는 아이콘의 MovieClip 을 필요에 따라 적절한 캔버스 MovieClip 에 붙이고 미니맵이 제거될 때 MovieClip 을 제거합니다. 다음에 나열된 아이콘 형식을 지원하기 위해 다양한 형식의 팩토리가 정의되어 있습니다. 각 아이콘 형식은 변경, 회전 및 textField 변화 등 특정 업데이트에 따른 아이콘 형식을 구현하는 Update() 메소드를 가지고 있습니다.
    - *PlayerIcon*: 아군 및 적 아이콘 리소스 모두를 나타냅니다. 둘은 같은 로직을 공유합니다. 둘의 차이는 아이콘 생성 메소드뿐입니다.
    - *FlagIcon*
    - *ObjectiveIcon*
    - *WaypointIcon*
- **FxMinimapView**: 미니맵 뷰용 주 컨트롤러. 뷰를 새로 고칠 때마다 응용 프로그램에서 여기 포함된 UpdateView() 메소드를 호출합니다.

다음 코드는 PlayerIcon::Update() 메소드입니다(FxMinimapView.cpp; 116 번째 줄). Direct Access API 를 사용하여 뷰에 표시되는 플레이어 아이콘을 업데이트할 때 사용되는 로직입니다. *MovieClip* 멤버는 표시된 아이콘의 MovieClip 에 대한 레퍼런스를 포함합니다. SetDisplayInfo 메소드는 느린

SetMember() 메소드를 거치지 않고 MovieClip 의 디스플레이 속성(디스플레이 매트릭스, 가시 플래그, 알파 값)을 바로 수정합니다. 그래도 GFx::Movie::SetVariable 보다 GFx::Value::SetMember 의 속도가 더 빠릅니다.

```
virtual void    Update(FxMinimapView* pview, FxMinimapEntity* pentity,
                                                float distSquared)
{
     SF_UNUSED(distSquared);
     bool hasIcon = (pentity->GetIconRef() != NULL);
     PointF pt = pentity->GetPhysicalPosition();
     bool showDir = pview->IsShowingPlayerDir();

     // If entity did not have an icon attached before, then wait for the
     // next update to set the state data. picon is most probably not initialized
     // (the movie needs to advance after creation)
     if (hasIcon && (State != (int)showDir))
     {
       // state 0: no arrow, state 1: show arrow
          Value frame(Double(showDir ? 2.0 : 1.0));
          MovieClip.Invoke("gotoAndStop", NULL, &frame, 1);
          State = (int)showDir;
     }

     pt = pview->GetIconTransform().Transform(pt);
     Value::DisplayInfo info;
     if (State)
     {
          info.SetRotation(pview->IsCompassLocked() ?
               (pentity->GetDirection() +  90)  : (pentity->GetDirection() –
                                        pview->GetPlayerDirection()));
     }
     info.SetPosition(pt.x, pt.y);
     MovieClip.SetDisplayInfo(info);

}
```

6 성능 분석

응용 프로그램에서 표시된 다음 통계는 HUD 뷰 업데이트 소요 시간, 미니맵 업데이트 소요 시간, 업데이트된 미니맵 객체의 개수, Scaleform 콘텐츠의 총 디스플레이 시간 및 SWF 를 진행하는 데 소요된 시간을 의미합니다.

객체의 개수는 아군/적 플레이어, 파워업, 목표 등 모든 종류의 개체를 합산한 것입니다. HUD 업데이트 시간은 빌보드, 로그 메시지, 라운드 점수판 및 플레이어 수치 등 모든 UI 표시기 업데이트를 포함합니다. 미니맵 업데이트 시간은 위에 언급한 미니맵 객체, 가려져 있던 지형, 나침반 및 플레이어 아이콘을 업데이트하는 데 소요된 시간을 모두 포함합니다. 이러한 뷰 업데이트 작업에는 MovieClip 의 시각적 속성(X/Y 위치, 라벨, 현재 프레임, 회전, 스케일 및 변형 매트릭스 등)을 관리하는 모든 로직 수행이 수반됩니다.

6.1 성능 통계

이전에 구현된 Scaleform 에서는 Movie 의 뷰 업데이트 시 C++ GFx::Movie::SetVariable 및 GFx::Movie::Invoke 메소드만 사용할 수 있었습니다. 그리고 이로 인해 MovieClip 의 경로가 해제되는 등 중대한 성능 저하 문제가 발생했습니다. 이와 달리 Direct Access API 는 더욱 빠른 코드 경로를 통해 같은 기능을 수행하여 Scaleform 3.1 이상 버전보다 훨씬 효율적으로 복잡한 HUD 를 업데이트할 수 있습니다.

다음 그래프를 통해 SetVariable/Invoke 메소드와 달리 Direct Access API 메소드 사용 시 미니맵 성능(업데이트 시간의 저하)이 크게 향상됨을 알 수 있습니다. 성능 향상은 10-25x 순서입니다.

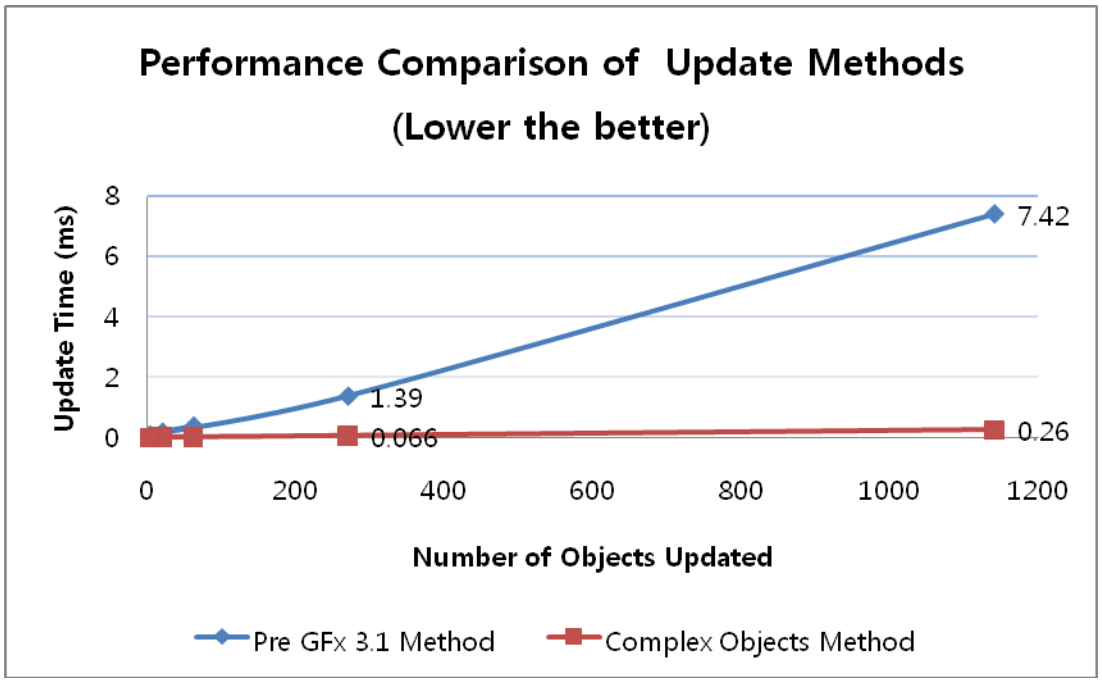


그림 15: 미니맵에 대한 Direct Access API 와 SetVariable/Invoke 업데이트 메소드 성능 비교

그래프에서 볼 수 있듯 Direct Access 메소드 업데이트 시간은 모든 작업 환경에서 HUD UI 평균 처리 할당 시간인 1ms 이하를 유지했습니다.

다음은 1 라운드 15 분 모의 실험을 통해 구한 전체 HUD Kit 에 대한 평균 성능 통계를 플랫폼별로 표시한 것입니다. 이 수치를 통해 Scaleform 에서 최고 성능으로 실행된 Flash HUD 의 성능을 추정할 수 있습니다.

플랫폼	FPS	업데이트(ms)	디스플레이(ms)	Advance(ms)	총합(ms)
Windows Vista MacBook Pro	1148	0.018	0.512	0.017	0.527
Xbox 360	1136	0.032	0.454	0.010	0.497
PlayStation 3	752	0.044	0.668	0.021	0.733

표 1: Scaleform HUD Kit 플랫폼별 평균 성능 통계

표 1 에서 볼 수 있듯이 HUD 업데이트 시간이 모든 작업 환경에서 HUD UI 평균 처리 할당 시간인 1ms 이하를 유지했습니다.

6.2 Memory Breakdown

다음은 HUD 데모에서 로드한 압축되지 않은 모든 콘텐츠에 대한 Memory Breakdown 입니다. 모든 콘텐츠는 GFxExport 초기 설정으로 보내기(Export)되었습니다. 총 메모리 사용량은 .GFX 파일, 이미지, 구성 요소 및 (Export) 폰트를 포함합니다.

1. **파일 형식: GFX 기본 보내기, 압축되지 않은 이미지**

총 메모리: 1858kb

- HUDKit.gfx - 33kb
- Minimap.gfx - 50kb
- fonts_en.gfx - 46kb
- gfxfontlib.gfx - 100kb

압축되지 않은 이미지

- HUDKit.gfx
  - 코어 UI 구성 요소 - 603kb
  - 순위 아이콘 - 72kb
  - 무기 아이콘 - 51kb
    - **참고:** 이 HUD 키트 버전에서 무기 아이콘(43kb)은 사용되지 않았습니다.
- Minimap.gfx
  - 구성 요소 - 644kb
- Map.jpg - 259kb