Autodesk®
Scaleform®

HUD 工具箱概述

本文介紹 Scaleform 4.2 HUD 工具箱。這是一個針對第一人稱射擊遊戲的全功能、AAA 級、可重複使用的使用者介面解決方案。本文重點講述 Flash UI 內容以及用來管理內容的 C++ 代碼。

作者： Nate Mitchell, Prasad Silva
版本： 2.0
上次編輯時間： 2010 年 7 月 30 日

Autodesk®
GAMEWARE

Copyright Notice

Autodesk® Scaleform® 4.2

如何联系 Autodesk Scaleform：

文档	HUD 工具箱概述
地址	Autodesk Scaleform Corporation 6305 Ivy Lane, Suite 310 Greenbelt, MD 20770, USA
網站	www.scaleform.com
郵箱	info@scaleform.com
電話	(301) 446-3200
傳真	(301) 446-3199

目录

1 引言

Scaleform HUD(抬頭顯示器)工具箱是一系列高性能、全功能、AAA 級品質的使用者介面工具箱中的第一個。開發人員可以容易地自訂這些工具箱並將其拖動到遊戲之中。HUD 工具箱演示如何使用新的 Scaleform® Direct Access API 來創建高性能的第一人稱射擊遊戲 (FPS) UI。

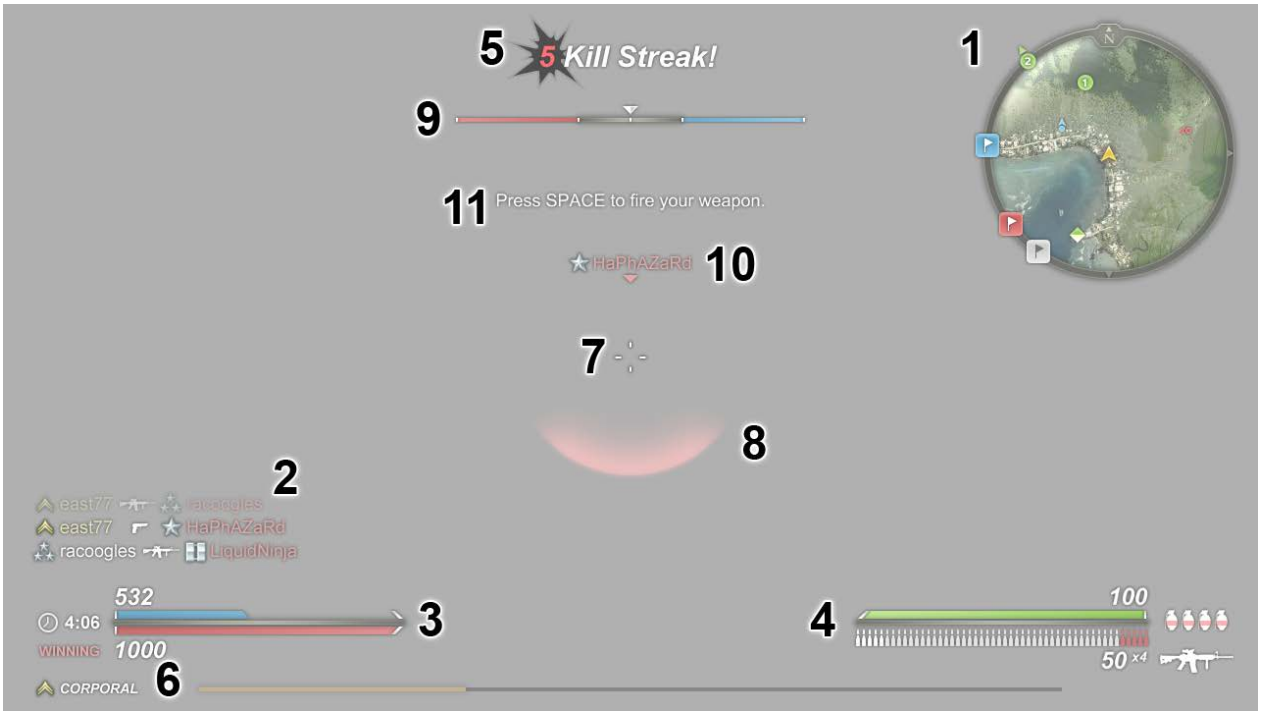

图 1：HUD 概览

HUD 工具箱包含下列可重複使用的 UI 元素:

1. 遊戲迷你地圖 (Minimap)
2. 遊戲事件的動畫日誌
3. 記分牌和團隊統計資料
4. 玩家健康 / 彈藥 / 狀態
5. 動畫彈出式通知
6. 經驗和級別顯示
7. 動態十字線
8. 定向命中指示器
9. 目標捕獲指示器
10. 看板 / 銘牌
11. 玩家文本通知

儘管此工具箱提供一套針對任何 FPS 遊戲的開箱即用的解決方案,但使用者不受已經提供的內容的限制。我們期望使用者自訂或擴展此工具箱的各個元素,以為各種類型的遊戲或應用程式創建新的具有創新意義的介面。

2 概述

2.1 檔位置和生成 (Build) 注釋

與本演示關聯的檔位於下列位置:

- *Apps\Kits\HUD\ - 包含用於 HUD 演示可執行檔的 C++ 代碼。*
- *Bin\Data\AS2 or AS3\Kits\HUD* - 包含 Flash 資源和 ActionScript 代碼（AS2 目錄有 ActionScript 2、Flash 8 資源,然後 AS3 目錄有 ActionScript 3、Flash 10 資源）。
- *Projects\Win32 \{Msvc80 or Msvc90}\Kits\HUD* –包含用於 Windows 上運行的 Visual Studio 2005/2008 的演示專案。
- *Projects\Xbox360\{Msvc80 or Msvc90}\Kits\HUD –包含用於 Xbox 360 上運行的 Visual Studio 2005/2008 的演示專案。*
- *Projects\Common\HudKit.*mk –用於 HUD 工具箱的 PS3 生成檔。

Bin\Kits\HUD 中可以找到該演示(用於 Windows)的一個預先生成的可執行檔 HUDKit.exe。它也可以經由開始功能表或 Scaleform SDK 流覽器進行訪問。

在 Windows 上,位於 Projects\Win32\Msvc80\Kits(或 Msvc90\Kits)目錄的 Scaleform 4.2 Kits.sln 檔可用來生成和運行本演示。確保從解決方案運行本演示之前將用於 Debugging(調試)的“Working 目錄”設置為 *Bin\Data\AS2\Kits\HUD* or *Bin\Data\AS3\Kits\HUD* 目錄。

在 Xbox 360 上,位於 Projects\Xbox360\{Msvc80 或 Msvc90}\Kits\HUD 目錄的 Scaleform 4.2 Kits.sln 檔可用來生成、部署和運行本演示。編譯完成時,用於本演示的所有資源(位於 Bin\Data\AS2 or As3\Kits\HUD)都將部署到目標 Xbox 360。

對於 PS3,應從 Scaleform 安裝目錄的根部運行 make 命令。預設情況下,這將會生成包括 HUD 工具箱 (HUD Kit) 在內的所有可用演示。在 PS3 上,可通過 SN Systems 工具集啟動可執行檔。.該可執行檔將生成到 Bin\PS3 並應使用以下選項進行啟動:

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

2.2 演示使用

為了最好地演示 HUD 工具箱的使用方法,該專案包含有一個非常簡單的類比遊戲,其中包含兩個隊:紅隊和藍隊,兩隊為控制散佈在戰場上的一組目標而展開競賽。

分數是通過捕獲目標(迷你地圖上標出)和擊斃敵方玩家來產生的。捕獲目標每 3 秒鐘就會為目前控制該目標的隊產生 1 分。擊斃一個敵人值 1 分。累計獲得 500 分的第一個隊,或一個回合(15 分鐘)終止時擁有最高得分的隊,將被宣佈為獲勝者。

玩家開始時配備有一挺機關槍、一把手槍、4 枚手雷和充足的彈藥。如果一個玩家被擊斃,其武器和彈藥將會得到補充。

以旋轉的綠色鑽石表示的能力提升 (Power-up) 將在戰場上大量產生隨機位置。每次能力提升都會對玩家的健康和拾取的隨機武器的彈藥產生預定的增強。

2.3 控制方案

2.3.1 鍵盤 (Windows)

W, A, S, D	控制使用者玩家。WS 可將玩家向前和向後移動。A, D 可分別將玩家向左和向右移動。
SPACE	用當前武器射擊。 所配備武器的射程、損傷能力和射速是獨一無二的。
R	給當前武器填彈。
G	投手雷。將對準射程內最近的玩家並爆炸,對爆炸半徑內的所有敵人造成損傷。
ESC	在標題列與文字方塊 (Text Field) 之間切換,可顯示應用程式的顯示器統計資料。
B	切换对用户玩家的 AI 控制。禁用游戏输入，直至失效。
1, 2	更換武器。 • 1 選擇手槍。 • 2 選擇步槍。

2.3.2 控制器 (Xbox/PS3)

定向墊 左控制杆	控制使用者玩家。W, S 可將玩家向前和向後移動。A, D 可分別將玩家向左和向右移動。
右觸發器	用當前武器射擊。 所配備武器的射程、損傷能力和射速是獨一無二的。
X / 方塊	給當前武器填彈。
左觸發器	投手雷。將對準射程內最近的玩家並爆炸,對爆炸半徑內的所有敵人造成損傷。
B / 圓圈	在標題列與文字方塊之間切換,可顯示應用程式的顯示器統計資料。
Y / 三角形	迴圈切換武器。

3 架構

HUD 工具箱包含 Flash UI 資源以及更新 HUD 視覺狀態的 C++ 代碼。同時還有一個 C++ 類比,提供一個以虛擬資料驅動 HUD 的環境。

3.1 C++

C++ 代碼的兩個主要部分是連接 HUD 視圖的介面以及向該視圖提供資料的環境。在 MVC 範例中,介面就是控制器,環境為模型,而 Flash UI 是視圖。帶 Fx 首碼的檔和類定義 HUD 工具箱的“視圖”(View) 核心的介面。沒有此首碼的檔定義本演示或類比遊戲的介面。

適配器設計方案的實現目的是使類比與 HUD 視圖脫鉤。適配器翻譯類比與 HUD 視圖之間的請求。要想用介面連接 HUD 視圖與一個新遊戲,使用者需要開發一個自訂的適配器類,該類應有連接遊戲與 HUD 視圖的介面。這樣,開發者就可以方便地重複使用 HUD 工具箱,而不必大幅度地修改遊戲代碼。

本文詳述的代碼介紹一種 Scaleform 的優化實現方法,它利用新的 Scaleform Direct Access API 來比以前更快地實現 UI 更新和動畫。

C++ 代碼包含下述檔(位於 Apps\Kits\HUD\ 及其子資料夾)。

3.1.1 演示

- **HUDKitDemo.cpp** –在標準 Scaleform Player 基礎上構建的核心應用程式。處理啟動類比、初始化成員以及利用其 C++ 控制器載入和註冊 SWF 檔的工作。
- **HUDKitDemoConfig**. –針對基於 FxPlayerConfig.h 的 HUDKitDemo 的配置。包括控制器映射和 Windows 應用程式定義。

3.1.2 HUD/迷你地圖視圖核心

- **FxHUDKit.h** –用來更新 HUD 視圖的核心 HUD 工具箱類型。
- **FxHUDView.h/.cpp** –提供 HUD 視圖使用的類型,包含日誌和看板 (Billboard) 系統。
- **FxMinimap.h** –聲明遊戲環境和應用程式為插到迷你地圖視圖中而可實現的介面。
- **FxMinimapView.h/.cpp** – 提供迷你地圖視圖使用的類型。

3.1.3 類比游戲

- **HUDAdapter.h/.cpp** – 實現適配器設計方案,使類比與 HUD 脫鉤,以便於重複使用。
- **HUDEntityController.h/.cpp** –實現適配器設計方案,使類比與 HUD 脫鉤,以便於重複使用。
- **HUDSimulation.h/.cpp** –提供實現演示環境以驅動遊戲 UI 的類型。此環境包含屬於一種捕獲並佔有遊戲類型的兩個隊的玩家。

3.2 Flash

用於 HUD 工具箱的 Flash 內容分為兩個檔:HUDKit.fla 和 Minimap.fla。這兩個 FLA 檔定義和佈置 HUD 工具箱的 UI 元素,以便於用 Scaleform 進行操縱。它們還包括 UI 中顯示的所有圖像和圖示。

兩個檔都分成若干層。一般來說,每個元件或每個具有相似元件的組都擁有自己的層。層提供一種方便的方法,用來在創作時根據深度控制元素排序。最上面的一層將顯示在所有其他層的上面,以此類推。

在 Scaleform 中,可使用 C++、ActionScript 或 Flash 時間軸來處理 Flash 動畫。在本演示中,多數 HUD 動畫是通過 Classic Tweens 在 Flash 時間軸上處理的。這些動畫一般通過調用 GFx::Value::GotoAndPlay() 來從 C++ 觸發。

3.2.1 HUDKit.fla

位於 Bin\Data\AS2 or AS3\Kits\HUD 目錄的 HUDKit.fla 檔是本演示的主要 SWF。此檔由 HUDKitDemo 應用程式在運行時載入。每個 Flash UI 元素都存在於此檔,但 HUDKit.fla 在運行時載入的迷你地圖視圖除外。

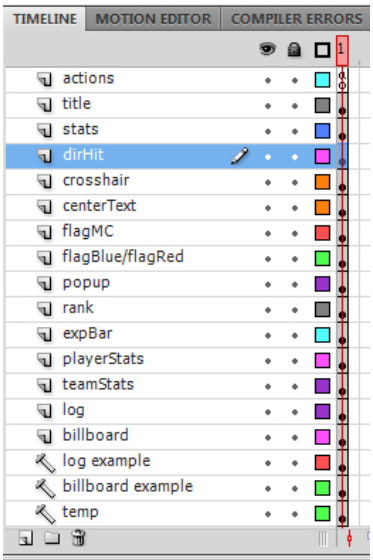

图 2：HUDKit.fla 层

HUDKitDemo 通過在時間軸的 Frame 1(幀 1)上調用 External Interface 來註冊 HUDKit MovieClip:
ExternalInterface.call("registerHUDView", this);

這會傳遞一個指向該 MovieClip(電影剪輯)的指標,C++ 可將該 MovieClip 註冊到 C++ HUDView 以便於操縱使用者介面。

本文“HUD 視圖”一節中將詳細講述每個層、其 MovieClips、 佈局、動畫和 Scaleform C++ 管理器。

3.2.2 Minimap.fla

图 3：Minimap 符号

Minimap.fla 包括 ‘Minimap’ 符號,即迷你地圖的核心。就本演示的範圍而言,背景是一個靜態圖像,而不是一個互動式 3D 環境。

玩家用位於中心的一個黃色箭頭來表示,而標題顯示在迷你地圖邊界上(“北”已標出)。除玩家之外,迷你地圖視圖還顯示五個圖示類型:

- 友方玩家(藍隊)。方向箭頭只出現在高縮放級別。
- 敵方玩家(紅隊)。方向箭頭只出現在高縮放級別。
- 捕獲點(用旗標 (Flag) 表示:白色代表中立,藍色代表友方,而紅色代表敵方)
- 能力提升(旋轉的綠色和白色圖示)

捕獲點和能力提升為粘圖示,它們在視野範圍外但在可探測範圍內時維持在視圖邊界。 Direct Access API 方法用來提供使進入和離開可探測範圍的圖示淡入/淡出的功能。它還用來在運行時使用 C++ 附加 MovieClips 和從台 (Stage) 上刪除 MovieClips。

此 'Minimap' 符號包含多個層。這些層提供一種方便的方法,用來在創作時根據深度控制元素排序。最上面的一層將顯示在所有其他層的上面,以此類推。

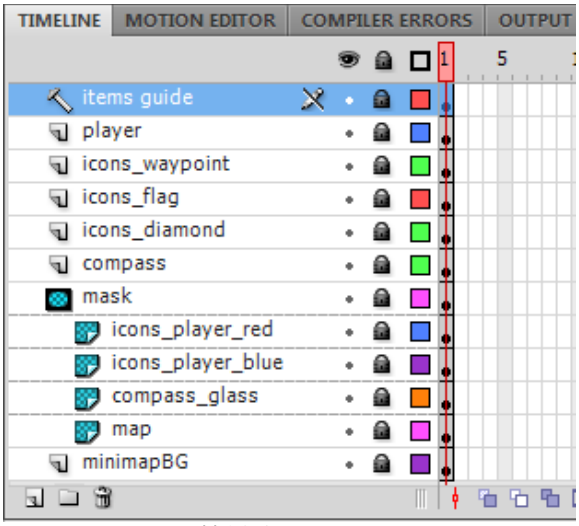

图 4：'minimap' 符号的层

- **items guide**：對於設計師/藝術家有用的引導層 (Guide Layer)。此層的內容不發佈到 SWF 檔。
- **player**：包含黃色玩家圖示的玩家。
- **icons_waypoint**：生成並緩存航點圖示的空白畫布。
- **icons_flag**：生成並緩存旗標的空白畫布。
- **icons_diamond**：生成並緩存能力提升的鑽石圖示的空白畫布。
- **compass**：包含指南針的層(標有"北")。
- **mask**：只顯示未掩蔽區域的內容的掩蔽層 (Mask Layer)。
    - **icons_player_red**：生成並緩存紅色敵方圖示的空白畫布。
    - **icons_player_blue**：生成並緩存藍色友方圖示的空白畫布。
    - **compass_glass**：在指南針左上方提供玻璃光澤的裝飾。
    - **map**：包含地形圖的層。出於演示目的,我們對地形使用一個大型點陣圖,不過,在製作情形下,這很可能是從 C++ 更新的紋理。在此情況下,更新邏輯不需要更改地圖的偏移/旋轉,這在使用掩蔽地形圖時是必需的。
- **minimapBG**：包含與地形點陣圖邊緣恰當混合的一個藍色背景。這在地形點陣圖超出視野時提供一個平穩的過渡。如果地形圖渲染為紋理並直接從 C++ 更新,就不需要該背景。

'Minimap' 符號使用 com.scaleform.Minimap 類(位於 Bin\Data\AS2 or AS3\Kits\HUD\com\scaleform 下麵)。)此最小類定義地圖圖像的路徑以及迷你地圖的構造函數的路徑,該構造函數將迷你地圖註冊到 HUDKitDemo 應用程式並載入地圖圖像。

4 HUD 視圖

C++ HUD 視圖管理內容 HUDKit.swf。其介面聲明下列類型:

- FxHUDView – 用於 HUD(特別是 HUDKit.swf)的核心管理器 (Core Manager)。根據類比狀態更新視圖和所有子 MovieClips。
- BillboardCache –用於友方/敵方看板的管理器和緩存系統(中心、螢幕指示器,不要與迷你地圖圖示混淆)。
- FxHUDLog – 用於 HUD 左邊顯示的消息日誌的管理器。跟蹤、顯示和重複使用 FxHUDMessages。
- FxHUDMessage – 消息定義,包括電影剪輯引用和消息文本。

請注意,FxHUDView 的緩存電影剪輯引用是用 MC(MovieClip 的簡寫)尾碼表示的。

在其初始化期間,FxHUDView 將一個引用註冊到運行時在 UpdateView() 中更改的每個主要 MovieClip。這對於避免在每次更新時都檢索電影剪輯引用非常重要,因而可以最大限度地減少用於更新視圖的時間。初始化還隱藏 HUDKit.fla 中定義的許多未使用的 UI 元素。

FxHUDView::UpdateView()用類比環境提供的資料更新視圖的每個元素。其邏輯分為以下幾個子方法:UpdateTeamStats()、UpdatePlayerEXP()、Update PlayerStats()、UpdateEvents()、UpdateTutorial() 和 UpdateBillboards()。一個指向類比環境的指標會被傳遞到其中每個方法,該類比環境然後用來更新特定 UI 元素。

請注意,很少在調用 UpdateView() 期間檢索電影剪輯引用。盡可能避免不必要的電影剪輯更新,因為每個幀都更新電影剪輯是一種潛在的迴圈動作浪費。

有了該版本 Scaleform Direct Access API,現在可以在 Flash 與應用程式之間高效地傳遞任何類型和形式的資料。尤其是,GFx::Value 類提供許多旨在使 Scaleform 使用者能夠更有效地控制 Flash 內容的新函數。GFx::Value::SetDisplayInfo()、GFx::Value::GetDisplayInfo() 和 GFx::Value::SetText() 用於整個 HUD 視圖的更新邏輯之中,便於直接而高效地操縱電影剪輯的顯示狀態。

4.1 回合統計資料和記分牌

teamStats MovieClip(位於 HUDKit.fla 的場景 1 (Scene 1) 中的 teamStats 層)是進行中的回合的記分牌和統計資料。它包括勝負文字方塊 (redWinning, blueWinning)、兩個文字方塊 (scoreRed, scoreBlue) 中顯示的各隊的得分以及進度條 (teamRed, teamBlue),還有回合時間時鐘文字方塊 (roundTime)。

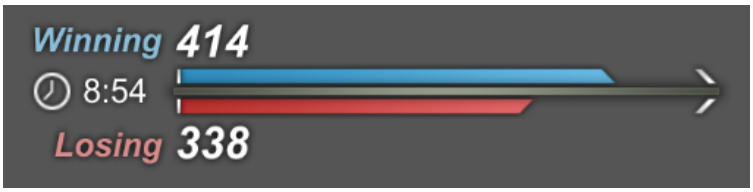

图 5：teamStats 符号

專門使用 FxHUDView::UpdateTeamStats() 來更新 teamStats。下麵的代碼更新 teamStats MovieClip 的元素。

```
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;
      }
}
```

得分存儲在一個 Scaleform::String 中,並使用 GFx::Value::SetText() 傳遞到 scoreRed 和 scoreBlue 文字方塊。teamRed 和 teamBlue 得分條的 X 尺規是使用 GFx::Value::SetDisplayInfo() 進行更新的。二者從其 0% 原始寬度開始,隨著各隊的得分向贏得相應回合所需的得分增加而逐漸伸展。

更新 blueWinning/redWinning 文字方塊之前,對比兩隊得分以免進行不必要的更新。調用 GFx::Value::SetText() 以便使用 HUDView 初始化期間定義的恒定 “Winning” 字串和 “Losing” 字串來設置文本。更新回合時間 textField 時,將最後一次調用 UpdateView() 時剩餘的時間與最新時間進行對比。如果上次更新以來過去的時間已經超過一秒,則使用 GFx::Value::SetText() 並通過格式化的時間字串來更新 roundTime textField。

4.2 Player Stats 和 Ammo

playerStats 符號(位於 HUDKit.fla 的場景 1 中的 playerStats 層)包含健康文字方塊和健康條形狀 (healthN, health)、武器圖示符號 (weapon)、HUD 的彈藥指示器以及彈藥不足指示器 (reload)。

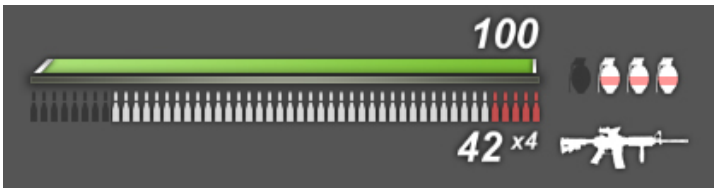

图 6：playerStats 符号

彈藥指示器分為六個電影剪輯,它們是 clipN、ammoN、ammo、ammoBG、grenade 和 grenadeBG。文字方塊 clipN 和 ammoN 以武器的彈夾和裝填的彈藥的形式顯示剩餘彈藥的數量。

ammo/healthVehicle 層包含兩個符號:ammo 和 ammoBG,每個符號分別被分隔到一個適當的子層中。兩個符號都在健康條下包含有彈藥指示器形狀。每個符號都分為 51 個 Keyframes(關鍵幀),每個關鍵幀擁有的子彈指示器都比前一個關鍵幀少一個。GFx::Value::GotoAndStop() 用來訪問這些電影剪輯的不同的關鍵幀,更改顯示的形狀的數量,以反映類比的狀態。

- *ammo* 顯示所配備武器中已裝填的子彈的數量。
- *ammoBG*顯示所配備武器的彈夾中的最大子彈數量。

grenade/armor 層與 ammo 層的設置相似,也是有兩個符號:grenade 和 grenadeBG,每個符號被分隔到各自的層中。兩個符號都包含手雷指示器形狀。就像在彈藥指示器中一樣,其中每個符號都分成 5 個關鍵幀。幀 1 有 4 個手雷形狀,而幀 5 沒有手雷形狀。GFx::Value::GotoAndStop() 用來訪問這些電影剪輯的不同的關鍵幀,更改顯示的形狀的數量,以反映類比的狀態。

- *grenade* 显示剩餘的手雷的數量。
- *grenadeBG* 顯示玩家可以攜帶的手雷的最大數量。

下麵的邏輯更新 playerStats 符號。

```
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 = UInt(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;
        }
}
```

在 ammo 中,幀 1 有 50 個子彈指示器,而幀 51 有 0 個子彈指示器。每次使其武器射擊時,都會調用 ammoMC.GotoAndStop(ammoFrames - ammoInClip) 以便訪問適當的關鍵幀,從 HUD 中刪除一個已裝填的子彈指示器。

對於 ammoBG,使用 ammoBGMC.GotoAndStop() 來訪問顯示 X 個子彈指示器背景的關鍵幀。這裡,X 基於所配備武器的彈夾大小。例如,機關槍 (Machine Gun) 的彈夾包含 50 發子彈。因此,ammoBG.GotoAndStop(1) 將顯示幀 1 中顯示的所有 50 個子彈指示器背景。只有在上次調用 FxHUDView::UpdateView() 以來所配備武器發生變化的情況下才更新此 MovieClip。

4.3 旗標捕獲指示器

flagMC 符號(位於 HUDKit.fla 的場景 1 的旗標層)是旗標捕獲指示器的容器。flagMC 包含 flag 符號,該符號分為兩部分:旗標捕獲指示器點陣圖和箭頭指示器 (arrow),後者根據旗標的 CaptureState(捕獲狀態)向左和向右移動。當玩家接近一個捕獲目標時,該指示器淡入,而當玩家不再射程內時,該指示器將淡出。

图 7：flagMC 符号

用於 flagMC 的淡入和淡出動畫是使用一個 Classic Tween 在 Flash 時間軸上完成的。flagMC 的時間軸有 4 個圖形狀態,它們分別用 4 個關鍵幀表示:Hidden(隱藏)、Visible(可見)、Fade-In(淡入)和 FadeOut(淡出)。這些狀態是使用 GotoAndStop() 和 Keyframe 的標籤/編號(1、5、“On”、“Off” 等)訪問的。

下麵複製的 FxHUDView::UpdateEvents() 使用本類比的資料來控制 flagMC 的狀態並更新 arrow(箭頭)的位置。

```
// 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;
      }
}
```

當調用 flagMC.GotoAndPlay("on") 時,flagMC 播放其淡入動畫,然後停在 Visible(可見)狀態。當調用 flagMC.GotoAndPlay("off") 時,flagMC 播放幀 11-20,即一個淡出動畫。當動畫達到幀 20 時,會自動重新開機從幀 1 重播。這是預設 Flash 行為;不需要額外的 ActionScript 或 C++。由於幀 1 包含一個 stop(); 調用,因此,電影剪輯將在播放幀 1 後停止。這是非常理想的情況,因為符號隱藏在幀 1 中,這是由於其 Alpha 設置為 0。

arrow 是使用 GFx::Value::SetDisplayInfo()橫向轉移的。此方法被傳遞來一個新的 GFx::Value::DisplayInfo,帶有根據實體的 CaptureState 的更新的 X 座標。

4.4 武器十字線

Reticule(十字線)符號(位於 HUDKit.fla 的場景 1 的準星層)是射擊十字線的容器。reticule 包含四個符號:left、right、bottom 和 top,每個符號都在自己的層內。自己的符號內擁有每個十字線使得單獨操縱成為可能。十字線與每當使用者玩家用其武器射擊時類比所啟動的 PlayerFire 事件相對應。當玩家用武器射擊時,十字線將會動態擴大。當玩家停止射擊時,十字線將會返回到其原始位置。

图 8：reticule 符号

十字線更新代碼分為兩個部分:FxHUDView::UpdateEvents() 方法和 FxHUDView::OnEvent() 方法。當捕獲到一個 PlayerFire 事件時,ReticuleHeat 和 ReticuleFiringOffset 在 FxHUDEvent 中分別遞增並設定。

```
// 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() 使用這兩個變數來切換 top、bottom、left 和 right 電影剪輯。對於每個電影剪輯,我們使用 SetDisplayInfo() 來切換電影剪輯的 X 座標或 Y 座標。方程中的第一個數位(6 或 -6)是從 0, 0 座標偏移的起始 X 或 Y。請注意,可以針對每次調用 SetDisplayInfo() 重新使用 GFx::Value::DisplayInfo 的單個實例。不過,為了確保不會發生意外重新使用 DisplayInfo 的情況,在每次調用 GFx::Value::SetDisplayInfo() 後再調用 DisplayInfo.Clear()。此方法清除了用於 DisplayInfo 實例的所有以前定義的顯示屬性。

```
// 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();
}
```

對於本演示,為了簡單起見,選擇了一個簡單的動態十字線方程,但此代碼可以容易地修改,以創建更加動態和有創意的十字線行為。

4.5 定向命中指示器

dirHit 符號(位於 HUDKit.fla 的場景 1 的 dirHit 層)是定向命中指示器的容器。dirHit 包含八個電影剪輯,它們被分成適當命名的層:tl、l、bl、b、br、r、tr 和 t。

其中每個電影剪輯都包含一個 Alpha 混合紅半圓,將作為一個指示器用於:

a) 受到損傷的使用者玩家
b) 該損傷的來源的相應方向

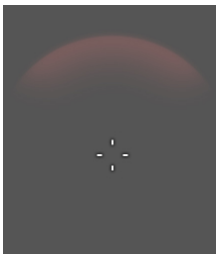

图 9：dirHit 符号

當一個玩家被命中時,就會出現命中指示器,隨後在 ~1 秒鐘後淡出。淡出動畫是 Flash 時間軸上的一個 Classic Tween Alpha 混合。通過針對任何命中指示器電影剪輯調用 GFx::Value::GotoAndPlay(“on”),相應命中指示器將會顯示,過一秒鐘後淡出。相應的電影剪輯基於損傷來源的相應方向。

```
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 等級和經驗顯示

expBar 符號(位於 HUDKit.fla 的 Scene 1 的 expBar 層)是螢幕底部的經驗條,通過玩家的當前等級來跟蹤玩家的進度。expBar 包含 exp MovieClip,後者是隨玩家通過擊斃和捕獲來獲得經驗而擴大的黃色進度條。

rank 符號(位於 HUDKit.fla 的 Scene 1 的 rank 層)顯示玩家的當前等級。rank 包含 18 個 Keyframe,每個專用於一個等級。每個 Keyframe 顯示與該等級關聯的圖示和標題。

图 10：expBar 和 rank 符号

```
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(UInt(pplayer->GetRank() + 1));
		info.SetScale(pplayer->GetLevelXP() * 100, 100);  // 伸缩 EXP 进度条
                                                          // 电影剪辑。
		ExpBarMC.SetDisplayInfo(info);
	}
}
```

與團隊得分進度條相似,當玩家的經驗發生變化時,exp 也根據玩家的當前經驗而伸縮。為了更新 rank,將 GFx::Value::GotoAndStop() 用於關鍵幀 X,其中 X 為玩家的當前等級 + 1(因為沒有 幀 0)。

4.7 文本通知

centerTextMC 符號(位於 HUDKit.fla 的場景 1 的 centerText 層)是一個動畫式文字方塊,用來向螢幕中心的使用者顯示資訊。在本演示中,centerText 用來在啟動各個回合時顯示一個簡單的教程,並宣佈該使用者玩家最近已被擊斃。該文本顯示後隨即淡出。

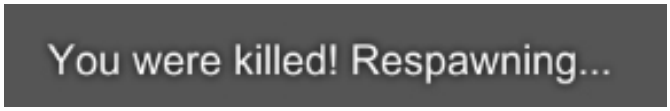

图 5：centerTextMC 符号

centerTextMC 包含 centerText 電影剪輯,後者包含實際的文字方塊 textField。如此排列電影剪輯的目的是便於製作動畫。centerTextMC 的時間軸包含兩個帶有標籤的關鍵幀 “on5”(在播放一個淡出動畫前顯示文本 5 秒鐘)以及 “on”(在播放一個淡出動畫之前顯示文本 3 秒鐘)。時間軸的末尾重新開機電影剪輯在幀 1 重播,在這裡,centerTextMC 的 Alpha 設置為 0,將其隱藏直到重新調用 “on” 或 “on5”。

```
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.
```

在此,對 centerTextMC 的 textField 進行了適當設置,並使用 GotoAndPlay(“on”) 顯示電影剪輯。電影剪輯將會由於時間軸的 Classic Tween 而在 ~3 秒鐘後自行淡出。

4.8 彈出消息和通知

popup 符號(位於 HUDKit.fla 的場景 1 的 popup(彈出消息)層)是動畫式事件指示器。在本演示中,每當使用者玩家達到 3 次以上連續擊斃(連殺)時,就會播放 popup。它將淡入並在顯示時播放一個動畫,而在 3-5 秒鐘後根據 C++ 調用而淡出。

图 6：popup 符号

Popup 是彈出消息的得分容器。其子元素的所有動畫都是使用 Classic Tweens 在 popup 的時間軸上完成的。popup 包含 4 層:actions、popupNumber、popupText 和 popupBG。popupBG 是在數位後面 “爆炸”的動畫黑色形狀。popupText 和 popupNumber 包含顯示彈出消息的文本和次數的 textField 符號。此符號層次是在時間軸上製作文字方塊動畫所必需的。

此符號有一個 “on” 關鍵幀,可以使用 GotoAndPlay() 來調用該關鍵幀以啟動動畫。一旦完成初始動畫,彈出消息就會淡出,而且重播將返回到幀 1 並停止播放,通過將其 Alpha 設置為 0 隱藏起來。

```
// 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); // 设置 PopUp 的文本框。
```

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

這裡,FxHUDEvent 作為一個 KillStreak 事件被丟棄。然後從玩家檢索連續擊斃的次數並相應地格式化。通過 C++ 設置預緩存的 popupText's textField 符號 PTextFieldMC 的文本,而 GotoAndPlay("on") 用來觸發 popup 的顯示動畫。

4.9 消息和事件日誌

log 符號(位於 HUDKit.fla 的 Scene 1 的 log(日誌)層)是 HUD 左下邊的消息和事件日誌的容器。在本演示中,日誌顯示與最近事件(包括擊斃、能力提升、旗標捕獲和級別提升)相關的文本消息。日誌底部添加新消息,就會向上推送舊消息。日誌消息在顯示 ~3 秒鐘後淡出。

图 13：log 和 logMessage 符号

日誌消息是 logMessage 符號的實例。在淡出動畫開始之前,添加到日誌的消息顯示大約 3 秒鐘。此動畫是在 logMessage 符號的時間軸上定義的。消息的文本從 C++ 發送,並使用 htmlText 在 textField(文字方塊)中顯示。

下麵的代碼來自 FxHUDMessage::Movieclip* FxHUDLog::GetUnusedMessageMovieclip(),後者創建一個新的 LogMessage MovieClip 並將其附加到 Log 日誌電影剪輯,即用於所有日誌消息的畫布。

```
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;
}
```

消息的內容從 C++ 按事件類型設置。下麵是 C++ 代碼,用來處理 Level Up 事件,並為將在日誌中顯示的“You are now a [Rank Icon] [Rank Name]”(您現在是 [等級圖示] [等級名稱])新消息設置 htmlText。

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

下麵是用來管理日誌的 HUD Update 邏輯。它更新用於日誌消息引用的佈局和動畫。

```
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;
        }
}
```

Log 的更新邏輯通過檢查某個 FxHUDMessage 是否設置為 0 來確定日誌中目前是否顯示該 FxHUDMessage。如果沒有顯示,就將其從 Log 中刪除,並添加回到未用 FxHUDMessage MovieClips 的 MessageMCs 清單。如果添加了一個新的 FxHUDMessage,就會使用 GFx::Value::DisplayInfo.SetY() 和 GFx::Value::SetDisplayInfo() 將所有 MessageMC 相應升檔。

4.10 看板

billboard_container(位於 HUDKit.fla 的 Scene 1 的 billboard 層)是玩家看板的容器。每個看板顯示一個跟隨玩家位置的箭頭和玩家名稱。看板經常在遊戲中顯示 3D 物件的附加詳細資訊,而 3D 物件包含在使用者玩家的視圖平截體 (Frustum) 內。在本演示中,玩家看板出現在使用者玩家的一定距離處。友方玩家始終顯示玩家名稱,而敵方看板只顯示箭頭,直到目標接近視圖中心。

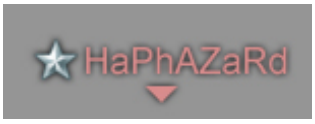

图 7：billboard_enemy 符号

本演示中的看板實現在 UI 中創建了一個 2D 看板電影剪輯,通過切換和伸縮並使用 GFx::Value::SetDisplayInfo() 來跟蹤玩家的位置。

每個看板 MovieClip (billboard_friendly, billboard_enemy) 都是從位於文本層的一個 billboard_label_(friendly/enemy) 以及位於箭頭層的一個箭頭圖像構造的。billboard_label_* 包含一個用來顯示目標玩家名稱的 textField。billboard_friendly's "Show" Keyframe 使用 billboard_friendly 的時間軸上的 Classic Tween 半透明混合在啟動淡入動畫。一旦播放 "Show"(顯示),看板就保持可見,直到其被隱藏或刪除為止。

將看板電影剪輯附加到畫布的 C++ 邏輯在 FxHUDView::BillboardCache::GetUnusedBillboardMovieclip 內:

```
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);

}
```

每個 BillboardCache 都包含一個對電影剪輯的引用,它將電影剪輯附加到看板命名的 CanvasMC。初始化時,此引用被傳遞到 BillboardCache。上面的邏輯將 billboard_enemy/billboard_friendly 符號的一個看板電影剪輯附加到其位於 CanvasMC.AttachMovie() 調用的預訂義畫布,並保留一個對它的引用,以便於重新使用。

Billboard 更新邏輯是在下列方法中定義的:UpdateBillboards()、BeginProcessing()、EndProcessing() 和 GetUnusedBillboardMovieclip()。在更新一組友方看板或敵方看板的開始和結尾調用 BeginProcessing() 和 EndProcessing()。這些方法管理已使用的和未使用的看板清單。

下麵的代碼是來自 UpdateBillboards() 的友方看板的更新邏輯。

```
// 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() 根據一個從使用者玩家視圖查詢檢索到的實體清單創建和更新看板。它使用 GotoAndPlay(“show”) 顯示看板,使用 GFx::Value::SetDisplayInfo() 設置 MovieClip’s X、Y 和尺規,並使用 GFx::Value::SetText() 更改顯示的文本。

textField 使用 htmlText 填充和顯示看板。該文字方塊包含玩家等級圖像和玩家名稱。等級圖示通過 src=’rank{0}’ HTML 定義。在上面的示例中,它與友方玩家等級 + 1 (friendlies[i].pPlayer->GetRank()+1)

相對應。其他 img 選項定義該圖示圖像的高度、寬度、對齊方式和 vspace。玩家的名稱通過 {1} 定義。在上面的示例中,它與友方玩家的名稱 (friendlies[i].pPlayer->GetName()) 相對應。

5 迷你地圖視圖

迷你地圖 (Minimap) 的 C++ 方面包含以下檔(位於 Apps\Kits\HUD\ 之中):

- **FxMinimap.h**：聲明遊戲環境和應用程式為插到迷你地圖演示核心中而可以實現的介面。
- **FxMinimapView.h/.cpp**：提供迷你地圖視圖使用的類型。

5.1 迷你地圖視圖

迷你地圖視圖介面聲明下列類型:

- **FxMinimapIconType**：圖示類型定義,包括類型 ID 以及用於額外專業化的額外子類型值。
- **FxMinimapIcon**：與視圖中某個元素連結的圖示資源。
- **FxMinimapIconCache**：存儲特定類型圖示集的圖示緩存。根據需要由該緩存來管理圖示視圖狀態,而且該緩存還支援進入/退出可探測範圍的圖示的淡入/淡出。該緩存使用一個工廠來生成新圖示。工廠的基實現是一個範本化的類,稱為 MinimapIconFactoryBase。各個工廠處理根據需要把圖示的電影剪輯附加到其適當畫布電影剪輯以及在迷你地圖被毀壞時刪除這些 MovieClips 方面的事情。定義了多個工廠類型來支援下列不同的圖示類型。各個圖示類型都包含一個 Update() 方法,該方法實現針對圖示類型的更新邏輯,例如:翻譯、旋轉和 textField 更改:
    - *PlayerIcon*：*同時代表友方和敵方圖示資源,因為兩種圖示類型共用相同的邏輯。它們只在圖示創建方法方面存在區別。*
    - *FlagIcon*
    - *ObjectiveIcon*
    - *WaypointIcon*
- **FxMinimapView**：迷你地圖視圖的主控制器。應用程式調用其 UpdateView() 方法來刷新視圖。

下麵的代碼是 PlayerIcon::Update() 方法(FxMinimapView.cpp;第 116 行)。它顯示用來利用 Direct Access API 更新視圖中的玩家圖示的邏輯。MovieClip 成員包含對臺上的圖示 MovieClip 的引用。SetDisplayInfo 方法直接修改電影剪輯的顯示屬性(顯示矩陣、可視性旗標或 Alpha 值),而不是通過較慢的 SetMember() 方法。請注意,GFx::Value::SetMember 仍然比 GFx::Movie::SetVariable 快:

```
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 性能分析

應用程式顯示的統計資料說明更新 HUD 視圖所花費的時間、更新迷你地圖所花費的時間、已更新的迷你地圖物件的數量、Scaleform 內容的總顯示時間以及推進 SWF 所花費的時間。

物件數量是各種類型的實體(包括友方/敵方玩家、能力提升和目標)的累計數量。HUD 的更新時間包括更新所有 UI 指示器,例如:看板、日誌消息、回合記分牌以及玩家統計資料。迷你地圖的更新時間包括更新前面提到的迷你地圖物件以及掩蔽地形、指南針和玩家圖示所花費的時間。更新這些視圖包括執行控制電影剪輯(如 x/y 位置、標籤、當前幀、旋轉、縮放和轉換矩陣)視覺屬性的所有邏輯。

6.1 性能統計資料

以前的 Scaleform 實現僅限於使用 C++ GFx::Movie::SetVariable 和 GFx::Movie::Invoke 方法更新 Movie 的視圖狀態。令人遺憾的是,由於許多因素(如解析 MovieClip 路徑)所致,這產生極大的性能損失。相比之下,Direct Access API 提供便於執行同樣的功能的更快的路徑,因此,使得使用 Scaleform 3.1 和更新版本進行複雜 HUD 更新高效得多。

下圖顯示採用 Direct Access API 方法與採用 SetVariable/Invoke 方法相比迷你地圖性能方面的顯著增強(縮短了更新時間)。性能增強為 10-25 倍:

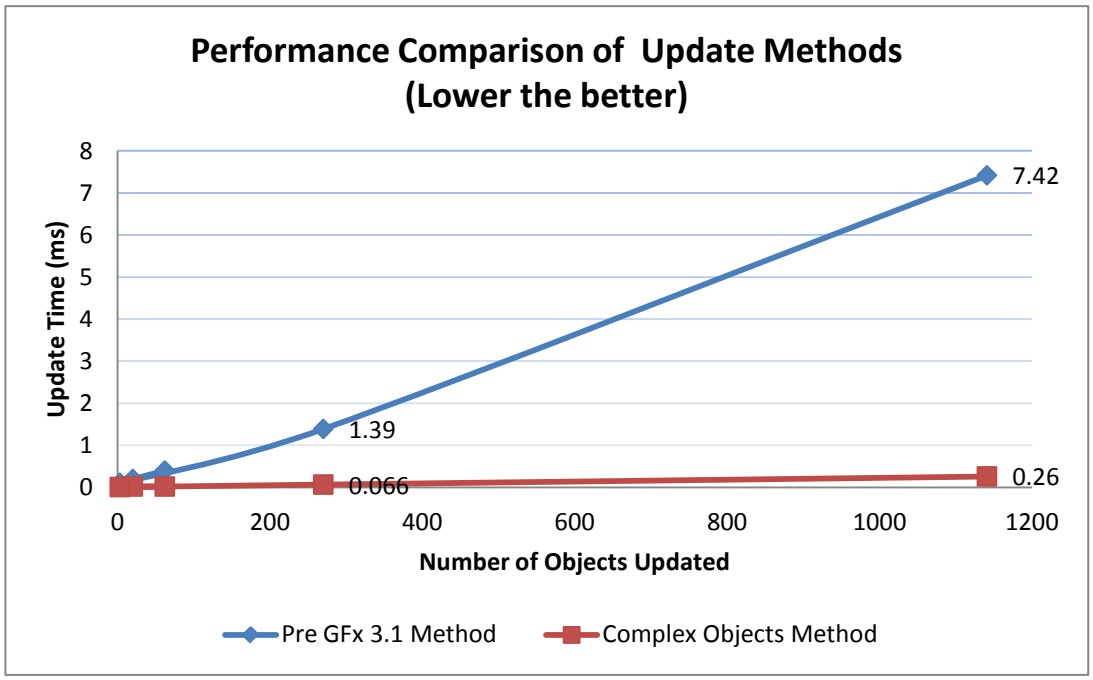

图 8：Direct Access API 与 SetVariable/Invoke 之间的迷你地图更新方法性能比较

如上圖所示,Direct Access 方法的更新時間對於各種工作環境來說都是最佳的,因為更新時間遠遠低於 1 毫秒的閾值 – HUD UI 的分配的平均處理時間。

下麵按平臺顯示一次 15 分鐘類比回合期間整個 HUD 工具箱的一般性能統計資料。這些數位提供 Scaleform 驅動的全功能 Flash HUD 的性能估計:

平台	FPS	更新（毫秒）	显示（毫秒）	推进（毫秒）	总计（毫秒）
Windows Vista 在 MacBook Pro 上	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

表 1：Scaleform HUD 工具箱每个平台的平均性能统计数据

如表 1 所示,HUD 的更新時間對於各種工作環境來說都是最佳的,因為更新時間遠遠低於 1 毫秒的閾值 – HUD UI 的分配的平均處理時間。

6.2 記憶體明細表

下麵是 HUD 工具箱演示載入的所有未壓縮內容的一個記憶體明細表。內容是使用 GFxExport 的預設設置匯出的。全部記憶體佔用包括 .GFX 檔、圖像、元件和匯出的字體。

1. 文件格式： **GFX 标准导出、未压缩的图像**

總記憶體： *1858kb*

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

未壓縮的圖像

- HUDKit.gfx
  - 核心 UI 元件 - 603kb
  - 等級圖示 - 72kb
  - 武器圖示 - 51kb
    - **注意：**本版 HUD 工具箱中,有 43kb 的武器圖示沒有使用。
- Minimap.gfx
  - 元件 - 644kb
- Map.jpg - 259kb