

*Tutorials e Guide
(e Traduzioni dal sito Ufficiale)*

SDL.NET

<http://www.francescomalatesta.net>
per altro materiale e articoli

di
Francesco Malatesta

Indice:

Prima Parte:

- Introduzione
- La Guida del Principiante ad SDL.NET
- Il Primo Programma: Hello World

Seconda Parte – Operazioni di Base

- Prima di Cominciare
- Eventi

- Output:
 - *Immagini;*
 - *Fonts;*
 - *Audio;*

- Input:
 - *Tastiera;*
 - *Mouse;*
 - *Joypad;*

- Il Template

Prima Parte

Introduzione

Ciao a tutti! Se vi trovate qui è perché, in un modo o nell'altro, siete interessati alla programmazione di videogiochi. Ciò che analizzeremo è una libreria usata proprio per questo scopo: creare videogiochi.

Una libreria, nella sua accezione informatica, è un insieme di funzioni di utilità più o meno comune che sono state appositamente create per l'utilizzo in un programma. L'obiettivo principale di una libreria è risparmiare tempo. Una volta un mio professore disse che *l'informatica è l'arte del riciclaggio*. Nulla di più vero.

Una libreria serve proprio a questo: evitare di reinventare ogni volta la ruota diminuendo il lavoro del programmatore.

Chiarito questo concetto è necessario che io spieghi quello che voglio fare attraverso questa serie di guide ed articoli: il mio principale obiettivo è capire al meglio il funzionamento di questa libreria, tradurre i vari articoli presenti sul wiki e, così facendo, fornire un valido aiuto a tutti i programmatori italiani interessati a questa nicchia che, per un motivo o per un altro, non masticano bene l'inglese.

Proverò passo passo tutti i passaggi, sintetizzando dove necessario ma anche spiegando linea per linea dove essenziale. Così facendo sarà facile creare un piccolo progetto ed iniziare a buttare giù qualche idea, creando qualcosa di proprio.

Con questo smetto di parlare a vanvera e inizio a fare qualcosa di utile :) Spero possa esservi d'aiuto!

La Guida del Principiante ad SDL.NET

titolo originale: “*The Absolute Newbie Guide to SDL.NET*”

Dunque, in questo primo articolo dedicato ad SDL.NET introdurremo qualche informazione relativa al suo funzionamento e qualche dato generico.

SDL.NET offre l'accesso ad una libreria multimediale e multi-piattaforma, che permette di gestire ad un livello abbastanza alto sia l'Audio, l'Input (mouse, tastiera, joystick), grafica 3D (tramite OpenGL) e grafica 2D.

Attualmente, l'home page del progetto è <http://cs-sdl.sourceforge.net/>, ed il maintainer al momento in cui scrivo è David “jendave” Hudson.

L'approccio delle API SDL.NET è molto simile al modello sul quale si basa un altro Wrapper, PyGame per il linguaggio di programmazione Python.

Dati Assembly:

- Nome: SdlDotNet.dll
- Namespace: SdlDotNet
- Note Aggiuntive:
 - Tao.Sdl contiene i binding di basso livello con le sei librerie C originarie (SDL, SDL_image, SDL_ttf, SDL_mixer, SDL_gfx, SMPEG)

Contenuti del file Zip:

- /bin – Assembly;
- /examples – Applicazioni demo basate su SDL.NET;
- /lib – Dipendenze e librerie originali, il cui uso verrà spiegato successivamente;
- /doc – files della documentazione in formato CHM e HTML;
- /source – files di informazioni, file delle soluzioni/build;
- /source/examples – codici sorgenti delle demo sopra trattate e fornite con il pacchetto;
- /source/lib – dipendenze della libreria;
- /source/src – codice sorgente di SDL.NET;
- /source/setup – script di compilazione;
- /source/tools – utility e loghi;
- /source/tests – unit tests;

Dipendenze:

Questi file sotto elencati devono essere necessariamente nella cartella dove si trova l'eseguibile (il file .exe) del vostro gioco. A volte non sono necessari tutti, ma solo quelli relativi alle funzioni da voi usate. Esempio pratico: nel vostro gioco non utilizzate filmati? Allora la libreria SMPEG non sarà un granché utile nel pacchetto finale.

Comunque sia, ecco la lista dei files che dovrebbero essere inclusi nel percorso del vostro programma:

- SDL;
- SDL_image;
- SDL_mixer;
- SDL_ttf;
- SMPEG;
- SDL_gfx;

Inoltre saranno necessarie le librerie .NET "Tao.Sdl".

Installazione del Pacchetto

Per quanto riguarda l'installazione di SDL.NET in questa guida tratterò solamente la parte relativa a Windows, in quanto attualmente uso solo questo sistema operativo ed ho modo di fare qui le prove. Per quanto riguarda Linux e Mac troverete le vostre risposte (in inglese) su: http://cs-sdl.sourceforge.net/index.php/The_absolute_newbies_guide_to_SDL.NET.

Sotto Windows potremo installare le nostre librerie tramite 3 soluzioni diverse:

- Runtime Installer, per gli utenti che non intendono modificare il codice sorgente;
- SDK Installer, per coloro che vogliono installare tutto il pacchetto completo di codice sorgente;
- Archivio Zip, un' alternativa all'installer classico, utile per gestirsi i files come si vuole.

Con il Runtime Installer:

- Dalla pagina di download del progetto, selezionare la voce SdlDotNet-x.x.x-runtime-setup.exe, dove x.x.x è l'ultima versione del progetto disponibile.
- Una volta scaricato il pacchetto, basterà eseguire il doppio click sul file e seguire le istruzioni a schermo.

Con l' SDK Installer:

- Dalla pagina di download del progetto, selezionare la voce SdlDotNet-x.x.x-sdk-setup.exe, dove x.x.x è l'ultima versione del progetto disponibile.
- Come per il caso precedente, cliccare due volte sul file appena scaricato e seguire le istruzioni.
- Per compilare i codici sorgenti basterà usare i files "build.bat".

Con l'Archivio Zip:

- Dalla pagina di download del progetto, scaricare il file SdlDotNet-x.x.x.zip.
- Copiare i file in /bin (SdlDotNet.dll e Tao.Sdl.dll) nella cartella della Global Assembly Cache (di solito è c:\winnt\assembly)
- Copiare i file in /bin/win32deps in una directory a tua scelta (di solito è consigliato C:\winnt\system32)
- Se si vuole compilare l'SDL.NET, eseguire il file build.bat.

Nei percorsi poco sopra, winnt potrebbe anche essere la cartella Windows.

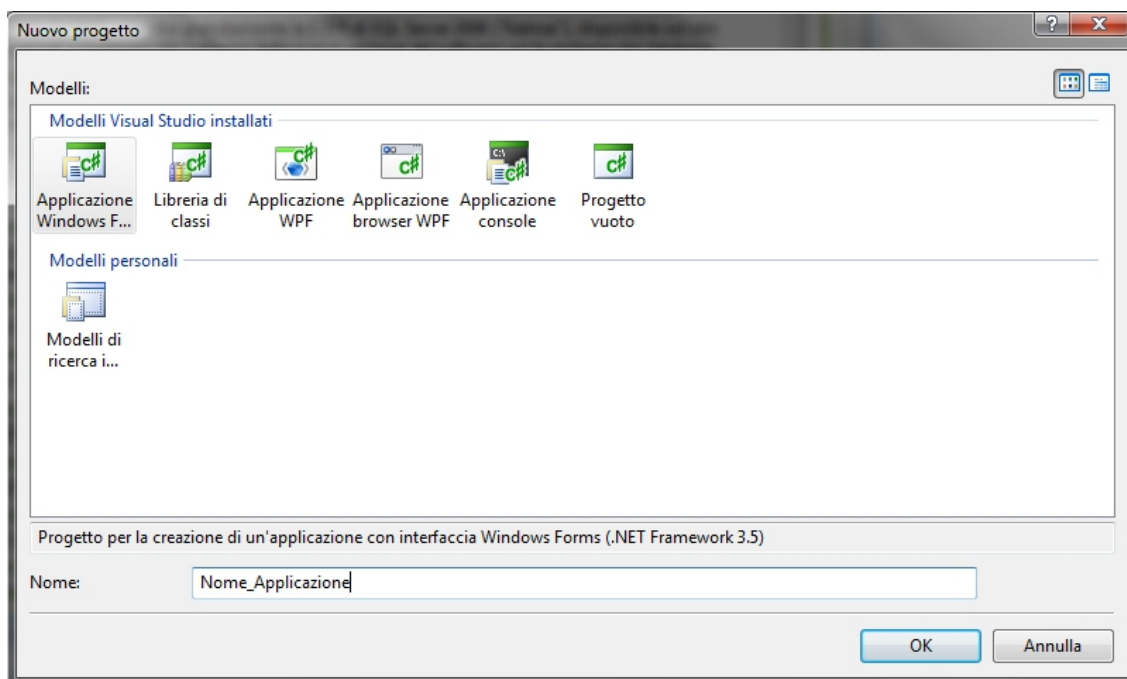
Se non sono stati riscontrati problemi, è tutto regolare e il nostro SDK è bello ed installato! Ora dobbiamo solamente programmarci (solamente eh -.-").

Il Primo Programma: Hello World

titolo originale: "Hello World"

Ok, dopo aver installato il nostro SDK è proprio ora di fare qualcosa di concreto. Perlomeno, ci proviamo. Per prima cosa apriamo il nostro Visual C# Express (al momento della scrittura uso la versione 2008, liberamente scaricabile dall'indirizzo <http://www.microsoft.com/express/vcsharp/>).

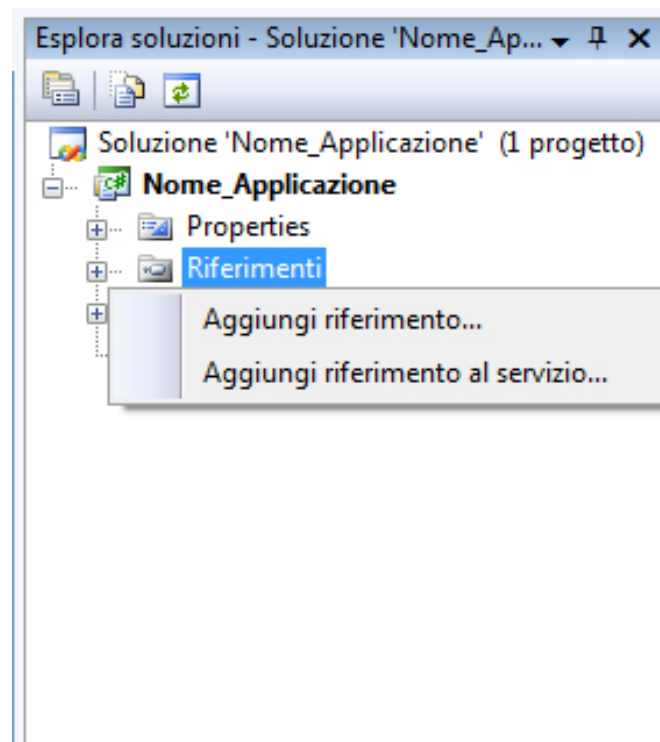
Dal menù File, in alto a sinistra, creiamo un nuovo progetto: ci si aprirà una nuova finestra dove dovremo decidere cosa fare. Selezioniamo "Applicazione Windows Form", scegliamo il nome per il progetto e clicchiamo su ok, come indicato nella finestra qui di seguito.



Una volta creato il nostro progetto si aprirà in automatico tutto l'ambiente di sviluppo, dove dovremo inserire il nostro codice. Prima di iniziare a scrivere selvaggiamente, però, ci sono altre piccole operazioni da effettuare.

- Aggiunta Referenze:

Nel Solution Explorer dovremo cliccare con il pulsante destro del mouse su Riferimenti, dopodichè scegliere "Aggiungi Riferimento" (immagine poco più giù). Dalla lista che ci comparirà selezionare "SdlDotNet" e cliccare su Ok.



- Cancellare il Form1:

Il file Form1.cs viene creato automaticamente, alla creazione del progetto. A noi non serve, per cui provvediamo immediatamente a cancellarlo selezionando il file con il pulsante destro del mouse e cliccando su Elimina.

- Cancellare il codice dal file Program.cs

Niente di più semplice, dovete cancellare il codice del file in questione e sostituirlo con il codice riportato nella pagina seguente.

Codice da incollare:

```
using System;

using SdlDotNet.Core;
using SdlDotNet.Graphics;

namespace SdlDotNetExamples.SmallDemos
{
    public class HelloWorld
    {
        [STAThread]
        public static void Main()
        {
            HelloWorld app = new HelloWorld();
            app.Go();
        }

        public HelloWorld()
        {
            Video.SetVideoMode(400, 300);
            Video.WindowCaption = "Hello World!";
        }

        public void Go()
        {
            Events.Quit += new EventHandler<QuitEventArgs>(this.Quit);
            Events.Run();
        }

        private void Quit(object sender, QuitEventArgs e)
        {
            Events.QuitApplication();
        }
    }
}
```

Fatto questo, non vi rimane che salvare il progetto e premere F5 per avviare la nostra prima applicazione con SDL.NET! Lo so, non è niente di eccitante.

Se invece il programma vi da degli errori, beh... vi ricordate cosa ho detto nell'articolo precedente, riguardo le dipendenze? Dovete avere il file SDL.dll nella stessa cartella del vostro file eseguibile, oltre alle librerie fornite con il package di installazione. Guardate l'immagine di seguito, in modo da capire quali sono i file da includere nella cartella:

zlib1.dll	21/07/2007 18.22
Test1.exe	09/11/2009 12.46
Tao.Sdl.dll	01/05/2008 19.00
smpeg.dll	21/07/2007 7.42
SdlDotNet.dll	01/05/2008 23.00
SDL_ttf.dll	21/07/2007 18.17
SDL_net.dll	21/07/2007 9.02
SDL_mixer.dll	21/07/2007 7.42
SDL_image.dll	21/07/2007 6.15
SDL_gfx.dll	25/07/2007 18.43
SDL.dll	31/12/2007 1.09
libvorbisfile-3.dll	21/07/2007 7.43
libvorbis-0.dll	21/07/2007 7.43
libtiff-3.dll	21/07/2007 6.13
libpng12-0.dll	21/07/2007 6.13
libogg-0.dll	21/07/2007 7.43
libfreetype-6.dll	21/07/2007 18.19
jpeg.dll	21/07/2007 6.13

Un consiglio: come potete vedere in questa cartella vi sono un'infinità di files: se volete, provate a togliere quelli che secondo voi non servono e fate tanti test. In questo modo potrete capire quali sono le dipendenze necessarie per ogni caso. Se invece volete andare sul sicuro, lasciate tutti i files nella cartella e non vi preoccupate ;)

Prima di proseguire, spieghiamo un po' quello che abbiamo fatto:

Tramite le direttive using abbiamo incluso i namespace delle "sezioni" della libreria che ci servivano: nel nostro caso abbiamo incluso "Core", ovvero la base essenziale, e "Graphics", che come suggerisce il nome serve per la gestione degli aspetti grafici.

Abbiamo creato quindi una piccola classe HelloWorld con i metodi essenziali alla creazione della nostra base: il costruttore HelloWorld, che si occupa di inizializzare il video ad una risoluzione di 400 x 300 e cambiare il titolo della finestra (WindowCaption) il metodo Go(), che assegna i vari EventHandler agli eventi previsti dalla libreria ed infine il metodo Quit, che regola l'uscita dal programma.

Spero mi perdonerete per questa mia sintetica spiegazione del codice: per ora è tutto quello che dovete sapere per darvi un'idea di base. Nelle prossime lezioni, tuttavia, spiegherò come capire il sistema degli eventi e sarete quindi in grado di comprendere e spiegare riga per riga tutto il codice di questo file.

Per ora, invece, è tutto!

Seconda Parte

Prima di Iniziare

Bene, la prima parte è andata. Abbiamo creato il nostro “Hello World”. La finestrella nera si è aperta, come prova della corretta impostazione dell'ambiente di sviluppo. Ovviamente, siamo ancora all'inizio. Continuiamo quindi quest'avventura alla scoperta dell'SDL.NET.

In questa seconda parte della guida parleremo degli aspetti più importanti della libreria, legati essenzialmente all'input (tastiera, mouse e joypad) ed all'output (grafica, audio, gestione dei fonts TrueType).

Ma non è tutto: prima di iniziare a lavorare con tutti questi aspetti è importantissimo spiegare cosa sono gli Eventi, come funzionano e a cosa servono. Sulla Wiki ufficiale del progetto SDL.NET gli eventi vengono spostati avanti di una lezione rispetto agli elementi grafici: nonostante ciò ho preferito studiare (e quindi riportare su questa guida) prima gli eventi e poi occuparmi di tutti gli altri ambiti. Il motivo è presto spiegato: gli eventi sono il cuore del funzionamento di un programma realizzato con SDL.NET.

Per questo motivo quindi la seconda parte della guida provvederà prima a spiegare gli Eventi, poi la gestione dell'Output, poi dell'Input. Infine prenderemo in considerazione un template da usare per le proprie applicazioni, ovvero una sorta di codice “standard” pronto da usare per una buona quantità di programmi in SDL.NET.

Bene, proseguiamo!

Eventi

Ok, eccoci arrivati agli eventi: per poter comprendere al meglio quest'importante concetto, riprendiamo il codice che avevamo usato nel nostro Hello World. Per non farvi perdere tempo tornando indietro, ve lo ripropongo qui:

```
--  
using System;  
  
using SdlDotNet.Core;  
using SdlDotNet.Graphics;  
  
namespace SdlDotNetExamples.SmallDemos  
{  
    public class HelloWorld  
    {  
        [STAThread]  
        public static void Main()  
        {  
            HelloWorld app = new HelloWorld();  
            app.Go();  
        }  
  
        public HelloWorld()  
        {  
            Video.SetVideoMode(400, 300);  
            Video.WindowCaption = "Hello World!";  
        }  
  
        public void Go()  
        {  
            Events.Quit += new EventHandler<QuitEventArgs>(this.Quit);  
            Events.Run();  
        }  
  
        private void Quit(object sender, QuitEventArgs e)  
        {  
            Events.QuitApplication();  
        }  
    }  
}  
--
```

Per prima cosa, analizziamo il metodo Go. All'interno di esso, analizziamo il primo comando, ovvero:

```
Events.Quit += new EventHandler<QuitEventArgs>(this.Quit);
```

Che cosa abbiamo fatto? Semplice, abbiamo appena gestito un evento del programma. Ma cosa vuol dire tutto questo? Quando noi creiamo un gioco con SDL.NET, esistono degli "Eventi" che vengono eseguiti in determinate condizioni: per esempio il OnKeyboardDown, che viene lanciato tutte le volte che premiamo un pulsante sulla tastiera.

Oppure l'evento Tick, che viene lanciato ad ogni fotogramma. O ancora, l'evento Quit, che abbiamo visto nel nostro codice, che viene eseguito prima di uscire dal programma. In questo modo il lavoro che il software fa viene alleggerito di tanto.

Ecco quindi spiegata la riga di codice: in poche parole noi associamo un metodo da noi creato (il Quit) ad un evento previsto dal sistema (Events.Quit). Nel nostro caso il metodo Quit è un semplice

metodo che consente di uscire dall'applicazione attraverso il metodo `Events.QuitApplication()`. Scritto in questo modo, potrebbe non significare molto.

Ma nel caso si vogliano cancellare delle variabili, allora possiamo mettere nel nostro metodo `Quit()` ciò che serve. Spero di essere stato chiaro e vi consiglio di fare molte prove per capire bene il funzionamento del sistema ad eventi.

Sulla Wiki ufficiale del progetto c'è scritto che il sistema degli eventi è molto utile ed intuitivo in fase di gestione dell'input. Senza dubbio vero, ma non è di certo tutto: c'è una grande agevolazione anche nella gestione di musica (per esempio l'evento `MusicFinished`, che come suggerisce il nome viene lanciato solo quando termina l'esecuzione di un brano musicale), delle finestre (`VideoResize`, indovinate che fa? ;)) e di altri aspetti che in seguito analizzeremo meglio.

Tra i comandi scritti nel metodo `Go`, inoltre, troviamo anche `Run`. `Run` è un metodo importantissimo per il gioco, in quanto è proprio esso che avvia il loop principale del programma, il ciclo che verrà eseguito fin quando non decideremo di uscire.

Oltre a `Run` ecco altri metodi della classe `Events`:

- `CloseAudio`, `CloseMixer`, `CloseJoystick` e così via, che chiudono i sottosistemi SDL indicati;
- `Close`, che chiude tutti i sottosistemi dell'SDL.
- `QuitApplication`, che permette l'uscita dal programma.

Un saggio uso degli eventi, insomma, aiuta tantissimo nello sviluppo della propria applicazione, oltre a garantire una facile lettura del codice e manutenzione. Che altro potrei dire? In realtà ben poco, ho già parlato abbastanza. Andiamo avanti ed iniziamo a vedere come si può caricare un'immagine e visualizzarla su schermo.

E' tempo di mostrare qualcosa sullo schermo, vero? Sono d'accordo con voi, iniziamo a parlare delle Surface, i primi oggetti che ci accompagneranno nella rappresentazione su schermo dei nostri file grafici. Volendo tradurre alla lettera dalla Wiki ufficiale, possiamo ottenere la spiegazione più esatta per questa classe:

"Le Surface in SDL.NET rappresentano, genericamente, delle informazioni grafiche. E' possibile crearle da zero (e quindi disegnarci ciò che si vuole) oppure caricando dei files dal disco fisso o da un array di bytes. Ogni Surface ha una propria larghezza ed altezza e possiedono delle funzionalità di disegno sulla superficie stessa."

Le cose non sono ancora chiare? Tranquilli, ecco un bel programma di prova, che verrà prontamente spiegato.

```
using System;

using SdlDotNet.Core;
using SdlDotNet.Graphics;

namespace Demo
{
    public class Game
    {
        Surface schermata = Video.SetVideoMode(800, 600);
        Surface immagine = new Surface("immagine.jpg");

        [STAThread]
        public static void Main()
        {
            Game app = new Game();
            app.Go();
        }

        public Game()
        {
            Events.Fps = 60;
            Video.WindowCaption = "Gestione Surface - SDL.NET";
        }

        public void Go()
        {
            Events.Quit += new EventHandler<QuitEventArgs>(this.Quit);
            Events.Tick += new EventHandler<TickEventArgs>(Events_Tick);
            Events.Run();
        }

        void Events_Tick(object sender, TickEventArgs e)
        {
            schermata.Fill(System.Drawing.Color.Black);

            schermata.Blit(immagine);

            schermata.Update();
        }

        private void Quit(object sender, QuitEventArgs e)
        {
            Events.QuitApplication();
        }
    }
}
```

```
}
```

Partiamo dalle prime righe:

```
using System;  
  
using SdlDotNet.Core;  
using SdlDotNet.Graphics;
```

Qua dovremmo esserci: includiamo i namespace che ci servono per i nostri scopi, attraverso la direttiva using.

```
Surface schermata = Video.SetVideoMode(800, 600);
```

Ecco la prima delle nostre Surface. Come detto precedentemente, queste possono essere create in vari modi: l'oggetto "schermata" che stiamo dichiarando è speciale: rappresenta infatti la finestra di gioco che noi utilizzeremo.

Il tutto viene reso possibile grazie al metodo SetVideoMode della classe Video, che restituisce appunto una Surface. In questo modo abbiamo deciso che la nostra finestra avrà una dimensione di 800 pixel per 600 pixel.

Ora abbiamo:

```
Surface immagine = new Surface("immagine.jpg");
```

Questa surface invece viene creata passando come argomento del metodo costruttore il nome del file da caricare. In questo caso caricheremo il file "immagine.jpg", situato nella stessa cartella del nostro file exe compilato. Una volta che il nostro oggetto viene creato è pronto per essere mostrato. Continuiamo però con il codice:

```
public Game()  
{  
    Events.Fps = 60;  
    Video.WindowCaption = "Gestione Surface - SDL.NET";  
}
```

Qui abbiamo il metodo costruttore della nostra classe Game, che svolge essenzialmente due operazioni: imposta il limite dei frame per secondo a 60 utilizzando la proprietà Fps della classe Events e cambia il titolo alla finestra del programma, stavolta utilizzando WindowCaption di Video. Molto semplice come concetto. Adesso:

```
public void Go()  
{  
    Events.Quit += new EventHandler<QuitEventArgs>(this.Quit);  
    Events.Tick += new EventHandler<TickEventArgs>(Events_Tick);  
    Events.Run();  
}
```

Il metodo Go aggiunge gli Handler per gli eventi che vogliamo gestire: in questo caso ne abbiamo due: Quit, già visto in precedenza, e Tick. Tick permette di gestire ciò che succede ad ogni frame, risultando particolarmente utile nell'aggiornamento dei valori delle variabili in gioco. Stando al limite dato precedentemente, la nostra funzione Events_Tick viene eseguita 60 volte al secondo. Quello che facciamo nel metodo Events_Tick è spiegato nelle righe successive:

```
void Events_Tick(object sender, TickEventArgs e)
```

```

{
    schermata.Fill(System.Drawing.Color.Black);

    schermata.Blit(immagine);

    schermata.Update();
}

```

Ed ecco l'utilizzo effettivo delle nostre Surface! Come si può vedere, in questo metodo eseguiamo tre operazioni:

1. Tramite la funzione Fill riempiamo tutta la schermata di nero. Può apparire insensato, ma pensate ad un file immagine con delle trasparenze (per esempio un file PNG) che si sposta nel tempo. Senza riempire lo schermo di nero tutte le volte avremmo un effetto fastidioso dovuto alla “non pulizia” della surface, se così la vogliamo chiamare.
2. Tramite il metodo Blit invece disegniamo la nostra immagine. Blit è un metodo che ha ben 8 overload: ciò vuol dire che ci sono ben 9 modalità diverse con cui disegnare sulle surface. Nel nostro caso abbiamo passato solo il nome della Surface da disegnare: questo vuol dire che le coordinate in cui verrà disegnata la nostra Surface saranno (0,0) rispetto alla “schermata”.
3. Il metodo Update si occupa infine di “aggiornare” il nostro frame e fare in modo che tutte le eventuali modifiche del sistema siano mostrate a schermo. Non dimenticatelo alla fine della fase di disegno, è importante!

Ma torniamo al punto 2: perchè queste coordinate sono (0,0)? E' semplice da spiegare: la Surface “schermata” rappresenta la finestra stessa del programma. Di conseguenza, il punto situato più in alto a sinistra avrà delle coordinate pari a 0 sull'asse x ed altrettanto sulla y.

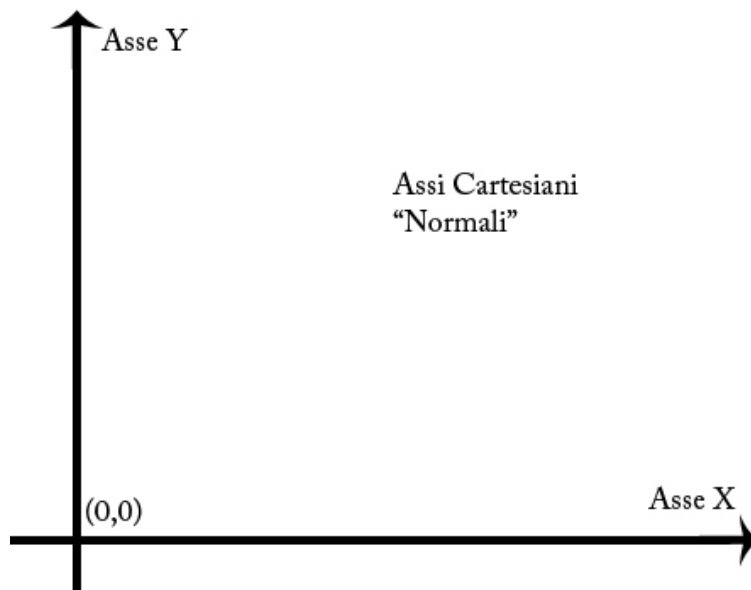
Per disegnare la nostra immagine in una posizione differente, la sintassi sarà la seguente:

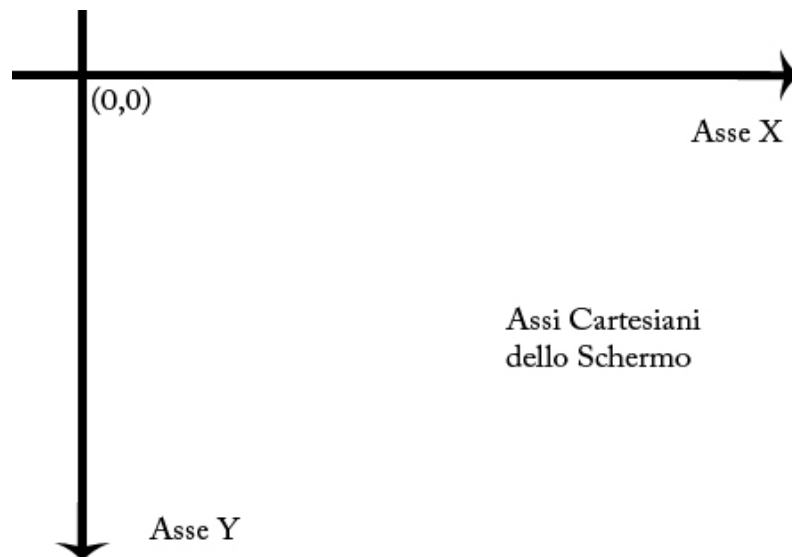
```

schermata.Blit(immagine, new System.Drawing.Point(20, 20));

```

Stavolta disegneremo la nostra immagine alle coordinate (20,20), e non più (0,0), sui nostri assi cartesiani. A tal proposito devo ricordarvi una cosa molto importante: gli assi cartesiani sullo schermo del pc sono ribaltati sull'asse Y. In poche parole, ecco due immagini per chiarire le idee.





Ciò che abbiamo dopo il metodo `Events_Tick` è il codice del metodo `Quit`, che ci consente di uscire dal programma.

Tornando a parlare genericamente, l'oggetto `Surface` supporta una buona quantità di formati grafici:

- BMP
- PNM
- XPM
- LBM
- PCX
- GIF
- JPEG
- PNG
- TGA

Trasparenza

Ovviamente non è solo questo quello che dobbiamo dire riguardo alle `Surface` e al loro utilizzo. Un'altra feature molto utile è quella della gestione della trasparenza, che avviene tramite due proprietà:

- `TransparentColor`
- `Transparent`

Facciamo un esempio. Abbiamo un'immagine in un file PNG che vogliamo utilizzare come cursore del mouse nel nostro gioco. Caricandola e mostrandola sullo schermo tramite la procedura sopra mostrata, tuttavia, non avremmo il risultato desiderato: oltre a mostrare il cursore, infatti, verrebbe mostrato anche il quadrato in cui è contenuta l'immagine, ovvero tutto il file immagine, per intero. Come facciamo ad evitare questo problema?

Per prima cosa, dobbiamo specificare il colore di trasparenza del nostro file. Ecco un veloce esempio, usando sempre la nostra `surface` "immagine":

```
immagine.TransparentColor = System.Drawing.Color.White;
```

In questo modo diciamo al sistema che, quando mostrerà a schermo la surface “immagine” dovrà ignorare tutti i pixel di colore bianco. Dopo questo, usiamo anche la proprietà `Transparent`, semplicemente impostandola su `True`. Ricordate che questo step è necessario.

```
immagine.Transparent = true;
```

Adesso il nostro cursore sarà bello e pronto da mostrare senza problemi ;). Ovviamente queste istruzioni è bene non metterle nel loop senza condizioni: io per esempio le metto nel metodo costruttore della mia classe. Ecco un esempio completo, per mostrare dove ho posizionato le istruzioni.

```
public Game()
{
    immagine.TransparentColor = System.Drawing.Color.White;
    immagine.Transparent = true;

    Events.Fps = 60;
    Video.WindowCaption = "Gestione Surface - SDL.NET";
}
```

Alpha Blending

Anche l'Alpha Blending è decisamente semplice da utilizzare. Come per la trasparenza, dovremo scrivere solamente due righe di codice, utilizzando due proprietà:

- Alpha
- AlphaBlending

Alpha è di tipo byte e va da 0 a 255. Impostandolo su 0 avremo la totale trasparenza della surface. A 255 invece la surface sarà totalmente visibile. Dopo aver impostato l'Alpha procediamo quindi ad attivare l'AlphaBlending impostando la proprietà su `true`, proprio come per la trasparenza. Se non effettuiamo quest'ultimo passo la proprietà Alpha verrà ignorata.

Ecco un esempio:

```
immagine.Alpha = Convert.ToByte(255);
immagine.Alpha = Convert.ToByte(0);
```

```
immagine.AlphaBlending = true;
```

Come suggerisce anche la Wiki ufficiale, un uso saggio delle surface può essere anche l'uso delle `SurfaceCollections`, ovvero strutture dinamiche che permettono di gestire altrettanto dinamicamente questo tipo di oggetto. Tuttavia ne parleremo più in là, magari quando riuscirò a scrivere una sorta di sezione “avanzata” di questa guida. Per ora, per quanto riguarda le surface, è tutto!

Output – Fonts

Come potrete facilmente immaginare, le immagini non sono l'unica componente grafica di un gioco. Molto spesso, infatti, anche nel gioco più semplice dobbiamo fare uso dei font: si pensi alla creazione delle GUI, delle varie interfacce o anche al punteggio del più semplicistico clone di pong.

Insomma, per poter fare qualcosa di decente abbiamo bisogno di uno strumento versatile che ci consenta di gestire i font e l'output testuale senza troppi problemi. In nostro aiuto, per quanto riguarda l'SDL.NET, arriva la classe Font. E sarà proprio nelle prossime righe che cercheremo di analizzare, almeno nelle sue basi, questo utile strumento.

Per prima cosa vediamo come iniziare ad usare un oggetto di questo tipo.

```
Font f1 = new Font("font1.ttf", 14);
```

f1 è un oggetto di tipo Font. Nel nostro caso, al costruttore passiamo una stringa con il nome del font utilizzato ed un intero che indica la grandezza del testo. Nulla di più semplice ed intuitivo. Dopo questa dichiarazione, in memoria abbiamo già il nostro font pronto ad essere utilizzato.

Adesso però dobbiamo scrivere qualcosa: ecco, qui di seguito, il codice che ci serve.

```
Surface testo = f1.Render("testo da scrivere!", System.Drawing.Color.Black);  
Surface testo = f1.Render("testo da scrivere!", System.Drawing.Color.Black, System.Drawing.Color.White);
```

Volendo spiegare in parole povere il funzionamento del meccanismo, pensate alla surface testo come un “foglio”, e all'oggetto f1 come una “stampante”. Nella prima riga di codice che ho riportato stiamo dicendo ad f1 di “stampare” sulla Surface "testo" la stringa “testo da scrivere!”, in colore nero. La surface, successivamente, sarà pronta per essere utilizzata. Il metodo Render ha svariati overload, tra cui l'altro che ho riportato: oltre al testo e il colore, nel metodo passiamo come terzo argomento anche il colore di sfondo da utilizzare.

La personalizzazione del testo, tuttavia, non finisce qui: la classe Font mette a disposizione svariate proprietà di customizzazione e formattazione, per fornire la massima flessibilità. Di seguito qualche esempio.

- **Proprietà Bold:** permette di decidere se il testo da renderizzare sarà in grassetto oppure no. Di default, questa proprietà è settata su False.
- **Proprietà Italic:** in questo caso possiamo decidere se il nostro testo sarà in corsivo oppure no. La proprietà è impostata su False di default.
- **Proprietà Underline:** Impostata su False normalmente, questa proprietà consente di decidere se il testo da renderizzare sarà sottolineato.

Oltre a queste proprietà c'è un'altra che riassume tutte e tre contemporaneamente, che si chiama Style. I valori che può assumere sono compresi nell'enumerazione SdlDotNet.Graphics.Styles. Ecco un esempio pratico qui di seguito.

```
f1.Style = Styles.Underline;
```

Insomma, la gestione del testo, almeno a livello base, è veramente molto semplice e facile da capire. Ovviamente, le cose si possono complicare se ci sono necessità riguardanti, per esempio, testi multilinea: a tutto però c'è una soluzione. Ma per adesso rimarremo sul semplice.

Come avete potuto notare dagli esempi (e se non l'avete notato ci penso io ;) i colori usati nei testi e negli sfondi sono quelli compresi nella struct `System.Drawing.Colors`. In questo modo avremo già pronti un sacco di colori.

Ma cosa fare se abbiamo necessità di utilizzare un colore di cui conosciamo perfettamente i valori RGB? Non c'è problema, ecco la sintassi!

```
f1.Render("testo da scrivere!", System.Drawing.Color.FromArgb(0, 0, 0));
```

La struttura `Color` infatti presenta il metodo `FromArgb`, dove `rgb` sta per `Red`, `Green` e `Blue`. Un attimo.. ma la `A`? `FromArgb` presenta anche due overload, ed uno sguardo veloce all'intellisense permette di capire subito di cosa si tratta: parliamo di `Alpha`, ed il valore che può assumere va da 0 a 255.

Un'altro overload di `FromArgb` consente invece di fornire come parametro un `int` a 32 bit nel quale specificare il numero `ARGB` che rappresenta il nostro colore. Come potete vedere c'è l'imbarazzo della scelta anche in questo caso.

Prima di concludere questa parte, dovete sapere che esiste anche un altro strumento per gestire i testi: parlo di `TextSprite`, contenuto in `SdlDotNet.Graphics.Sprites`. Stavolta, però, ci si ferma alla gestione base dei fonts e riprenderemo il discorso successivamente.

Adesso, infatti, lasciamo perdere per un po' il mondo della vista, per addentrarci in quello dell'udito :) Nel prossimo articolo parleremo di `Audio` per i nostri programmi.

Output – Audio

Nell'ambito della gestione dell'audio ci sono svariate classi interessanti: in questa parte della guida prenderò in considerazione quelle che reputo le due "base" per il settore. Parlo delle classi Sound e Music. L'SDL.NET supporta svariati formati di files Audio: Microsoft Wave, Ogg Vorbis, MIDI. Attualmente, tuttavia, gli mp3 non sono supportati, per cui se dovete usare dei files di questo formato dovrete guardare altrove (anche di questo parleremo più avanti).

Come già avvenuto per le Surface e per i Fonts, il caricamento in memoria di un file audio è veramente semplice:

```
Sound bang = new Sound("bang.wav");
```

In questo modo all'oggetto bang sarà associato il suono "bang.wav" residente in memoria. Anche la riproduzione del suono è enormemente agevolata grazie al metodo Play(). Ecco due esempi per comprendere quanto è facile:

```
bang.Play();    // Il suono viene riprodotto una volta.  
bang.Play(4);  // Il suono viene riprodotto quattro volte.
```

Essendo questa una classe base, ovviamente, non ci sarà la possibilità di agire molto sulla personalizzazione del suono e la sua modifica. Tra gli altri metodi interessanti che possiamo trovare, tuttavia, abbiamo:

- Il metodo FadeIn che permette di riprodurre il suono in fade in. Gli passiamo come parametro un intero che indichi il tempo di fade in millisecondi.
- Il metodo FadeOut, analogamente alla precedente, permette di stoppare il suono in fade out. Anche in questo caso gli passiamo il numero di millisecondi sottoforma di un intero.
- Il metodo Stop, ovviamente, non può mancare. Non ha bisogno di presentazioni ;)

Tra le proprietà degne di nota, invece, riporto solamente la proprietà "Volume", che permette di regolare il volume.

Per la musica c'è un'altra classe a parte, come detto precedentemente: Music. Funziona nello stesso identico modo di Sound per quanto riguarda il caricamento dei suoni. La proprietà Volume, però, stavolta non è disponibile.

L'ho già detto precedentemente e mi ripeto: come potete vedere le classi che ho trattato in questa piccola parte della guida sono molto semplici e non sono molto potenti. Permettono di fare operazioni base ma niente di eccezionale. Tuttavia, una volta che si prenderà dimestichezza con entità quali la classe Mixer e la classe MusicPlayer (di cui parlerò in un contesto più avanzato) le cose cambieranno radicalmente.

Date un'occhiata all'esempio di riferimento, in modo tale da capire meglio un pò il funzionamento e fare qualche prova. Per ora è tutto: iniziamo a parlare un pò di input ;)

Alla prossima!

Input – Tastiera e Mouse

Ora cambiamo un po' argomento. Abbiamo visto tutto (o quasi) ciò che poteva riguardare l'output: abbiamo imparato a caricare un'immagine sullo schermo, come renderla trasparente, come caricare un font e scriverci qualcosa ed anche aprire un file audio per usare un effetto sonoro nei nostri programmi.

Sapete bene però che un gioco non è solo Output. Un gioco è interazione, un gioco è anche Input. Per cui, adesso inizieremo a guardare le varie alternative che ci vengono proposte per gestire al meglio il modo in cui l'utente interagisce con il nostro gioco. Vi ricordate il sistema degli eventi, che avevo descritto ed introdotto qualche tempo fa? Inizieremo proprio da lì.

Riprendiamo l'esempio dell'Hello World, apportando però qualche modifica al codice:

```
using System;

using SdlDotNet.Core;
using SdlDotNet.Graphics;

namespace SdlDotNetExamples.SmallDemos
{
    public class HelloWorld
    {
        [STAThread]
        public static void Main()
        {
            HelloWorld app = new HelloWorld();
            app.Go();
        }

        public HelloWorld()
        {
            Video.SetVideoMode(400, 300);
            Video.WindowCaption = "Hello World!";
        }

        public void Go()
        {
            Events.KeyboardDown += new
            EventHandler<SdlDotNet.Input.KeyboardEventArgs>(Events_KeyboardDown);
            Events.Run();
        }

        void Events_KeyboardDown(object sender, SdlDotNet.Input.KeyboardEventArgs e)
        {
            if (e.Key == SdlDotNet.Input.Key.Escape) Events.QuitApplication();
        }
    }
}
```

Dunque, cosa abbiamo di diverso? Sicuramente l'istruzione

```
Events.KeyboardDown += new EventHandler<SdlDotNet.Input.KeyboardEventArgs>(Events_KeyboardDown);
```

E... cosa vorrebbe dire?

Semplicemente, che abbiamo gestito l'evento che si verificherà alla pressione di un tasto sulla nostra tastiera.

Ehm... cioè? :)

Spieghiamoci meglio: ogni volta che il nostro giocatore premerà un pulsante sulla tastiera, verrà eseguito il metodo `Events_KeyboardDown`. Ecco quindi che troviamo il codice del metodo:

```
void Events_KeyboardDown(object sender, SdlDotNet.Input.KeyboardEventArgs e)
{
    if (e.Key == SdlDotNet.Input.Key.Escape) Events.QuitApplication();
}
```

Come parametri in ingresso passeremo un oggetto `sender` e un `KeyboardEventArgs` chiamato "e". Soprattutto questo sarà importantissimo, come vedremo tra poco. Ed infatti ecco l'istruzione chiave che spiega questo nuovo concetto:

```
if (e.Key == SdlDotNet.Input.Key.Escape) Events.QuitApplication();
```

Non è molto difficile da capire: in poche parole qui si dice al nostro caro programma che ogni volta che l'utente premerà il tasto `Escape` (ESC) il programma dovrà terminare, tramite la chiamata al metodo `QuitApplication`, già menzionato precedentemente.

Avviando il nostro programma di prova premendo `F5`, potremo subito verificare i risultati. Premendo `Esc`, infatti, l'applicazione terminerà facendoci tornare alla schermata del nostro IDE (ho scritto `F5` perchè do per scontato l'uso del `Visual C# Express`, che agevola di molto le cose).

L'oggetto "e" che passiamo, inoltre, offre svariate possibilità grazie alle sue proprietà: la booleana `Down`, che restituisce `true` se un tasto è stato premuto. Oppure ancora `Unicode` ed `UnicodeCharacter`, che forniscono informazioni sul carattere Unicode rappresentato dal tasto appena premuto. Ciò, quindi, lascia spazio per un buon numero di possibilità e personalizzazioni.

Passiamo ora a dare un'occhiata al `Mouse`: anche in questo caso abbiamo un buon numero di eventi da gestire e metodi pronti da usare per facilitare il nostro lavoro. Un esempio pratico è rappresentato dal codice di seguito.

```
using System;

using SdlDotNet.Core;
using SdlDotNet.Graphics;

namespace SdlDotNetExamples.SmallDemos
{
    public class HelloWorld
    {
        Font fl = new Font("Calibri.ttf", 12);

        string posizione = "";

        [STAThread]
        public static void Main()
        {
            HelloWorld app = new HelloWorld();
            app.Go();
        }

        public HelloWorld()
        {
            Video.SetVideoMode(400, 300);
            Video.WindowCaption = "Hello World!";
        }
    }
}
```

```

public void Go()
{
    Events.KeyboardDown += new
EventHandler<SdlDotNet.Input.KeyboardEventArgs>(Events_KeyboardDown);
    Events.MouseMotion += new
EventHandler<SdlDotNet.Input.MouseMotionEventArgs>(Events_MouseMotion);
    Events.Tick += new EventHandler<TickEventArgs>(Events_Tick);
    Events.Run();
}

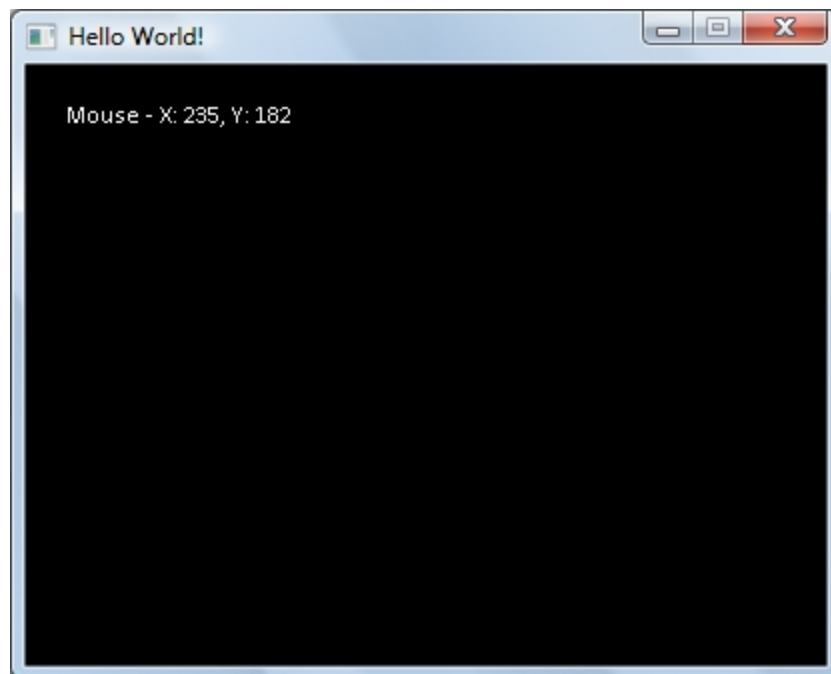
void Events_Tick(object sender, TickEventArgs e)
{
    Video.Screen.Fill(System.Drawing.Color.Black);
    Video.Screen.Blit(fl.Render(posizione, System.Drawing.Color.White), new System.Drawing.Point(20, 20));
    Video.Screen.Update();
}

void Events_MouseMotion(object sender, SdlDotNet.Input.MouseMotionEventArgs e)
{
    posizione = "Mouse - X: " + e.Y.ToString() + ", Y: " + e.X.ToString();
}

void Events_KeyboardDown(object sender, SdlDotNet.Input.KeyboardEventArgs e)
{
    if (e.Key == SdlDotNet.Input.Key.Escape) Events.QuitApplication();
}
}
}

```

Qui cosa abbiamo fatto? Andiamo con ordine. L'obiettivo di questo programma non è niente di eccezionale: leggendo ogni spostamento del mouse, riportiamo a schermo le sue coordinate. Queste saranno memorizzate in una stringa che verrà utilizzata con un font.



Ecco che dichiariamo la nostra stringa (inizialmente vuota) “posizione” e il nostro font fl (ricordate che il file del carattere da utilizzare dovrà essere ubicato nella stessa cartella dell'eseguibile che compileremo):

```
Font f1 = new Font("Calibri.ttf", 12);
string posizione = "";
```

Adesso andiamo a gestire l'evento relativo al movimento del mouse, che si chiama MouseMotion:

```
public void Go()
{
    Events.KeyboardDown += new
EventHandler<SdlDotNet.Input.KeyboardEventArgs>(Events_KeyboardDown);
    Events.MouseMotion += new
EventHandler<SdlDotNet.Input.MouseMotionEventArgs>(Events_MouseMotion);
    Events.Tick += new EventHandler<TickEventArgs>(Events_Tick);
    Events.Run();
}
```

Il metodo che lanceremo per gestire la procedura sarà Events_MouseMotion. Il nome è creato automaticamente dal sistema, ma potete crearne uno direttamente voi. L'ho fatto per comodità e per il fatto che viene generato al volo dall'Intellisense del Visual Studio, alla semplice pressione del tasto Tab. Comunque, ecco il nostro codice:

```
void Events_MouseMotion(object sender, SdlDotNet.Input.MouseMotionEventArgs e)
{
    posizione = "Mouse - X: " + e.X.ToString() + ", Y: " + e.Y.ToString();
}
```

Anche stavolta abbiamo un oggetto sender ed uno “e”, stavolta però di tipo MouseMotionEventArgs. Quindi, cosa fa questo metodo? Semplicemente, ogni volta che il sistema rileva uno spostamento del mouse assegneremo alla stringa “posizione” i valori di X ed Y che il mouse assume in quel momento. Finchè il mouse non verrà mosso, i valori non cambieranno ulteriormente.

Occorre anche in questo caso fare una bella scarrellata di proprietà che ci vengono messe a disposizione dall'oggetto “e”. Non le riporto tutte ovviamente, ma solo alcune:

- X ed Y: due proprietà che indicano la posizione sull'asse X ed Y del mouse allo stato attuale del sistema.
- Button: restituisce il nome del pulsante del mouse attualmente premuto usando come valori possibili l'enumerazione Input.MouseButton.
- ButtonPressed: booleana che restituisce True nel caso un pulsante sia stato premuto.

Questi strumenti permettono di gestire al meglio l'input da parte dell'utente, sfruttando le potenzialità del classico metodo “mouse più tastiera” tipico della grande maggioranza di giochi in circolazione negli ultimi anni.

Spero di essere stato chiaro in queste spiegazioni. Vi consiglio assolutamente di fare molte prove e, come già detto precedentemente, sperimentare per bene il sistema ad eventi se non lo si è capito bene. Specialmente adesso, forti di queste nuove piccole nozioni. Con queste basi, molto presto saremo pronti ad affrontare la creazione di un piccolo gioco vero e proprio.

Alla prossima!

Il Template

Appena ho visto il file Template.Cs, messo a disposizione da Paul Aspinall sul sito ufficiale di SDL.NET, ho subito trovato questa risorsa decisamente interessante. Dove molto spesso c'è confusione nell'approccio iniziale, questo file mette subito in chiaro tutti i metodi tendenzialmente usati e permette di avere il vantaggio di uno "scheletro" base già pronto.

Ho detto base perchè, volendo partire senza troppe pretese da questo file di certo non realizzeremo Assassin's Creed 2. Comunque sia, riporto qui di seguito il codice del file per intero commentato, fornendo anche la disponibilità per il download. Buona lettura!

```
//Public Domain
//Original version written by Paul Aspinall.
//Versione commentata da Francesco Malatesta in Italiano :)

// Direttive using per l'inclusione dei namespace interessati.
using System;
using System.Drawing;

using SdlDotNet.Core;
using SdlDotNet.Graphics;
using SdlDotNet.Input;

namespace Template_per_SDL.NET1
{
    public class Template
    {
        /*
         * Questo metodo base è statico ed è il punto d'ingresso nella nostra applicazione. Da qui, infatti,
         * dichiareremo una nuova variabile di tipo Template ed eseguiremo il metodo Go.
         */
        [STAThread]
        public static void Main()
        {
            Template game = new Template();
            game.Go();
        }

        /*
         * Il costruttore Template attualmente è vuoto ma generalmente è molto utile per inizializzare variabili
         * che verranno successivamente usate nel gioco.
         */
        public Template()
        {
        }

        /*
         * Il metodo Go, in questo caso, definisce la risoluzione della finestra di gioco (800 x 600) e avvia il metodo
         * AddHandlers, che analizzeremo tra poco. In ultimo, inoltre, lancia il metodo Events.Run(), facendo partire
         * in questo modo il Loop principale del gioco.
         */
        public void Go()
        {
            Video.SetVideoMode(800, 600);
            this.AddHandlers();
            Events.Run();
        }

        /*
         * AddHandlers è un metodo che si occupa di gestire il sistema ad eventi, definendo con facilità tutti quelli
         * che dovranno essere contemplati dal nostro programma. In questo caso abbiamo aggiunto degli Handler per

```

```

    * l'evento Quit (uscita dal programma) e Tick (evento che si presenta ad ogni frame).
    */
private void AddHandlers()
{
    Events.Quit += new EventHandler<QuitEventArgs>(this.Events_Quit);
    Events.Tick += new EventHandler<TickEventArgs>(this.Events_Tick);
}

/*
 * RemoveHandlers() toglie quello che AddHandlers() aveva aggiunto: vengono rimossi gli Handler ai vari eventi
 * e, come vedremo successivamente, questo metodo viene utilizzato in fase di uscita dal gioco.
 */
private void RemoveHandlers()
{
    Events.Quit -= new EventHandler<QuitEventArgs>(this.Events_Quit);
    Events.Tick -= new EventHandler<TickEventArgs>(this.Events_Tick);
}

/*
 * Events_Tick è il metodo contemplato per l'evento Tick: quello che fa è semplice. Inizialmente riempie
 * la finestra di blue, tramite il metodo Fill della Surface Screen. Successivamente aggiorna il tutto tramite
 * il metodo Update(), provvedendo in questo modo a mandare sullo schermo i risultati dell'elaborazione.
 */
private void Events_Tick(object sender, TickEventArgs e)
{
    Video.Screen.Fill(System.Drawing.Color.Blue);
    Video.Screen.Update();
}

/*
 * Come già detto precedentemente, Events_Quit provvede a rimuovere i vari EventHandlers tramite il metodo
 * RemoveHandlers() e successivamente esce dal programma, tramite il metodo Events.QuitApplication().
 */
private void Events_Quit(object sender, QuitEventArgs e)
{
    RemoveHandlers();
    Events.QuitApplication();
}
}
}

```