

DEV.

La rivista che ti insegna a programmare

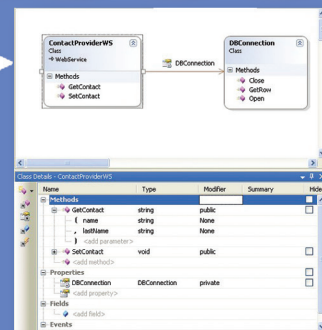
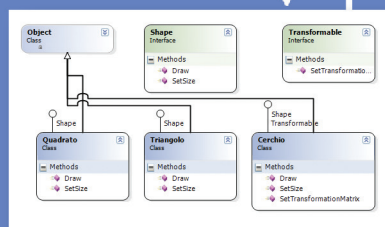
Visual Studio 2005

Panoramica, caratteristiche e novità

Programmo Subito

Il valore dell'utilizzabilità nel software

Rapid application development con Qt



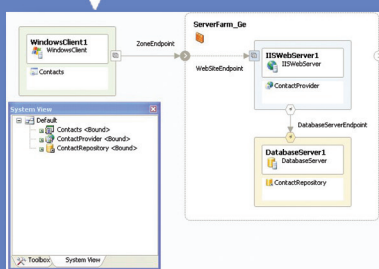
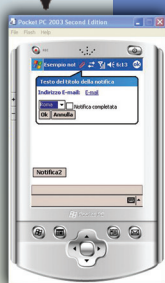
C#
Disegnare diagrammi e grafi in .NET

DB
Uno sguardo a MaxDB

JAVA
Automazione dei regression test con JUnit

ZOPE
Plone: Un Content Management System per Zope

GAME DEV
Realizziamo un videogioco con Game Maker 6



```

namespace ContactProvider
{
    [System.Web.Services.WebResource]
    public class ContactProvider : System.Web.Services.WebResource
    {
        private global::ContactProvider.ContactProvider()
        {
        }

        public void GetContact(string name, string lastnames)
        {
            Show new System.Web.Services.WebResource()
        }

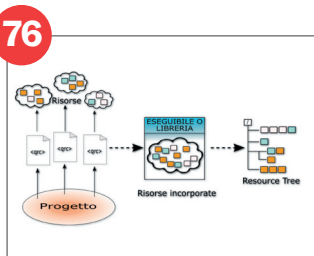
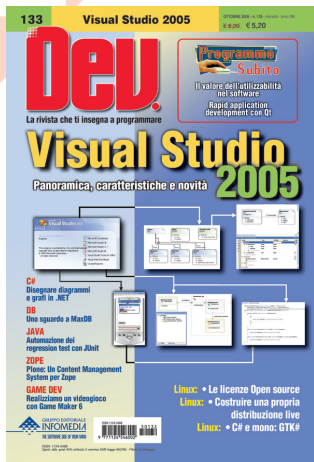
        public void SetContact(string name, string lastnames, string phoneNumbers)
        {
            Show new System.Web.Services.WebResource()
        }
    }
}

```

Linux: • Le licenze Open source

Linux: • Costruire una propria distribuzione live

Linux: • C# e mono: GTK#



Lo speciale di questo numero: Visual Studio 2005

- 14 **Visual Studio 2005 Team System: soluzione totale**
di Ferruccio Bongiani
- 20 **Visual Studio 2005 class diagram** di Gian Maria Ricci
- 25 **Unit testing in .NET 2.0** di Gian Maria Ricci
- 31 **Il controllo notification per l'ambiente pocket pc in Visual Basic .NET** di Emanuele Mattei

Programmazione

- 34 **C#** -Disegnare diagrammi e grafi in .NET di Stefano Franco
- 38 **Cellulari** -Il protocollo AT+ dei telefoni cellulari.
Comandi speciali Siemens: l'Organizer (PARTE 19)
di Vito Vessia
- 42 **Database** -Uno sguardo a MaxDB di Lorenzo Braidì
- 46 **Java** -Automazione dei regression test con JUnit di Fabio Latrofa
- 49 **Visual Objects** -Come fare per... di Enio Donci
- 51 **Zope** - Plone: Un Content Management System per Zope (ULTIMA PARTE)
di Franco Gambato

Rubriche

- 8 **Game Dev** -Realizziamo un videogioco con Game Maker 6 (ULTIMA PARTE)
a cura di Mauro Pesci
- 60 **Normative** -La tutela delle fotografie digitali di Massimo Farina
- 90 **CrackMe web edition** -CrackME web edition?
a cura di Fabio Scagliola
- 98 **Horizon** -Il re è nudo! -di Luigi Morelli

Sei agli inizi? Non riesci a risolvere i primi problemi? Niente paura! Iscriviti alle mailing-list Infomedia!

programmo_subito@infomedia.it
e linux-it@infomedia.it

I listati e i programmi di utilità che corredano gli articoli sono prelevabili gratuitamente dal sito <http://www.infomedia.it/DEV> oppure da <ftp://ftp.infomedia.it/pub/DEV/Listati/>



Programmo Subito

Numero 86

- 64 **Livello 3** I nuovi sviluppi dei linguaggi di Front End – di Stefano Corti
- 69 **Livello 1** Il valore dell'utilizzabilità nel software – di Ioana Codoban e Manuel Breschi
- 76 **Livello 1** Rapid application development con Qt – di Fabrizio Angius

Linux Subito

Numero 65

- 81 **Livello 1** Le licenze Open source (PARTE 4) – di Alessio Saltarin
- 85 **Livello 3** Costruire una propria distribuzione live (PARTE 3) di Corrado Ignotti
- 91 **Livello 1** C# e mono: GTK# (PARTE 2) – di Gianluigi Spagnuolo

Ai lettori

Doktor Morb a cura di Luigi Morelli
Velocità felina

48

Linux 001 a cura di Remigio Darino
Linux 001

50

E-ducational a cura di Luigi Morelli
Rudi Mathematici

63

Free Reading a cura di Lorenzo Vandoni
Windows WP Media Center Edition SDK

80

Java ME World a cura di Stefano Sanna
Due importanti novità

89

www.infomedia.it – The Software Side Of Your Mind

Dev. Developing Software Solutions
Numero 133
Ottobre 2005
Anno XIII

Direttore Responsabile

Marialetizia Mari (mmari@infomedia.it)

Coordinamento Editoriale

Roberto Casarini (rcasarini@infomedia.it)

Collaboratori

Fabrizio Angius, Ferruccio Bongiani,
Lorenzo Braidì, Manuel Breschi,
Ioana Codoban, Stefano Corti
Remigio Darino, Enio Donci,
Massimo Farina, Stefano Franco,
Fabio Gambato, Corrado Ignotti,
Fabio Latrofa, Emanuele Mattei,
Luigi Morelli, Mauro Pesci,
Gian Maria Ricci, Alessio Saltarin,
Stefano Sanna, Fabio Scagliola,
Gianluigi Spagnuolo,
Lorenzo Vandoni, Vito Vessia

Grafica e Impaginazione

Manola Greco (mgreco@infomedia.it)

**GRUPPO EDITORIALE
INFOMEDIA**

Gruppo Editoriale Infomedia srl
Via Valdera P.116 56038 Ponsacco (PI)
Tel 0587 73 64 60 (r.a.)
Fax 0587 73 22 32
www.infomedia.it – red_dev@infomedia.it

Direzione

Natale Fino (nfino@infomedia.it)

Divisione Libri

Lisa Vanni (book@infomedia.it)

Marketing & Advertising

Tel. 0587 73 64 60
marketing@infomedia.it

Supporto Sistemistico

webmaster@infomedia.it

Amministrazione

Sara Mattei (amministrazione@infomedia.it)

Segreteria

Enrica Nassi (info@infomedia.it)

Stampa

Betagraf Spa – Funo di Argelato (BO)

Distribuzione

Parrini & C. (Roma)

Ufficio Abbonamenti

Tel. 0587 73 64 60
Fax 0587 73 22 32
abbonamenti@infomedia.it
online@infomedia.it

Si prega di inviare i comunicati stampa e gli inviti stampa per la redazione all'indirizzo: comunicatistampa@infomedia.it

La riproduzione con qualsiasi mezzo di illustrazioni e di articoli pubblicati sulla rivista, nonché la loro traduzione, è riservata e non può avvenire senza espressa autorizzazione dell'editore. I manoscritti e le illustrazioni inviati alla redazione non saranno restituiti anche se non pubblicati e l'editore non si assume alcuna responsabilità per il caso che si tratti di esemplari unici.

"DEV. Developing Software & Solutions"
è una rivista del Gruppo Editoriale Infomedia S.r.l.
Direzione e Amministrazione
Via Valdera P., 116 – Ponsacco
Registrazione presso il Tribunale di Pisa
n. 21 del 25.11.1999

Contenuto pubblicitario inferiore al 45%

Realizziamo un videogioco con **Game Maker 6**

ULTIMA PARTE

Siamo giunti all'ultimo appuntamento della serie di articoli dedicati a Game Maker. Questa volta affronteremo l'argomento più interessante, cioè, l'uso degli oggetti. Vedremo anche come creare gli script necessari alla realizzazione di comportamenti ed effetti molto complessi.

Nell'articolo precedente abbiamo preparato l'ambiente con tutto ciò che serve per far "vivere" il nostro videogioco. Gli *sprite* caricati nel progetto, ora, saranno in grado di muoversi ed interagire tra loro. Per una completa comprensione del presente articolo, data la complessità del progetto realizzato, consiglio vivamente ai lettori di scaricare il pacchetto presente sul sito FTP.

Animare lo shuttle

Lo Shuttle è l'*oggetto* più complesso che dobbiamo realizzare per via della varietà di comportamenti di cui dovrà essere dotato. Iniziamo quindi con l'aggiungere un nuovo *oggetto* al progetto alla solita maniera, cliccando con il tasto destro sul menu ad albero alla voce *Objects* e selezionando *Add Object*. In questo modo si aprirà la finestra *Object Properties* che, aiutati dall'immagine in **Figura 1**, andremo subito a descrivere. Essa è divisa in tre sezioni; nella prima, iniziando dall'alto, è presente la solita casella di testo per l'inserimento del nome dell'*oggetto* dove scriveremo *objShuttle*. Appena sotto, nel pannello chiamato *Sprite*, verrà specificato, cliccando sull'icona a destra della casella dove è scritto *<no sprite>*, quale *sprite* associare all'*oggetto*.

Dopo il click comparirà l'alberatura degli elementi presenti nella categoria degli

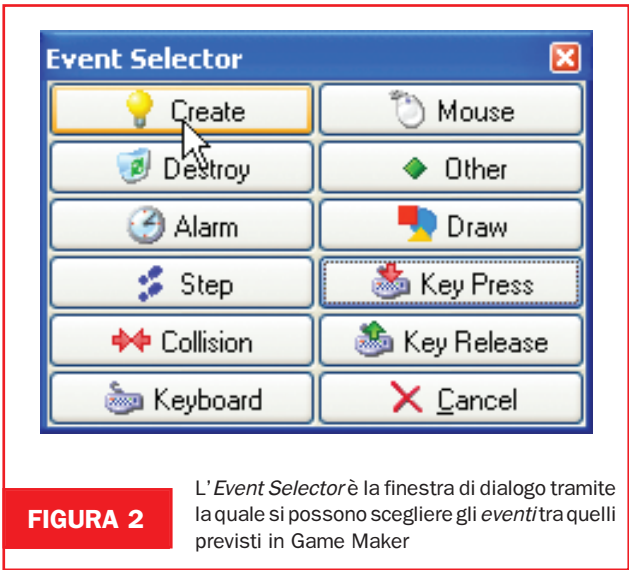
sprite. Ora è necessario puntare con il mouse alla voce *Partita* poi *Shuttle* ed infine cliccare su *sprShuttle*. Abbiamo, così, associato l'immagine dello Shuttle all'*oggetto*.

Accertiamoci che il segno di spunta sia presente sulle checkbox chiamate *Visibile* e *Solid* le quali servono, rispettivamente, a rendere visibile e solido l'*oggetto* nella stanza. Per quanto la prima proprietà risulti ovvia non può essere detto altrettanto della seconda. Per determinare le collisioni, come abbiamo detto nel precedente articolo, viene lanciato un evento quando gli *sprite* di due *oggetti* si sovrappongono. Cosa può accadere però quando l'esecuzione del programma prosegue al passo successivo? Che gli *sprite* potrebbero trovarsi ancora sovrapposti, di conseguenza Game Maker lancerà un altro evento di collisione.

In molti casi non si vuole che ciò avvenga; immaginiamo ad esempio che un meteorite entri in collisione con lo Shuttle e che quest'ultimo a differenza del primo sia privo di moto. Ad ogni passo il meteorite continuerebbe il suo percorso e, sovrapponendosi continuamente allo *sprite* dello Shuttle, il gioco lancerebbe continuamente eventi di collisione. Impostando, invece, l'opzione *Solid* ogni *oggetto* che viene a contatto con lo Shuttle, al passo successivo, sarà riportato indietro alla posizione che occupava prima di causare una collisione, evitando così l'inconveniente di cui abbiamo appena parlato.

Proseguendo con la finestra delle proprietà troviamo la casella *Depth*. Questa indica l'ordine in cui gli *oggetti* devono





essere disegnati. Ad esempio, con un valore grande e positivo l'oggetto verrà disegnato prima di altri in modo tale che, sullo schermo, risulti posto dietro. Un valore grande e negativo permette all'oggetto di apparire in primo piano poiché viene disegnato per ultimo. A noi interessa che si verifichi questa condizione, quindi inseriremo il valore -1000.

La voce *Persistent* indica se l'oggetto creato deve esistere dopo il passaggio da una stanza all'altra. Per i nostri scopi, dovrà rimanere deselezionato.

Non abbiamo bisogno di impostare nemmeno la casella *Parent* che implementa una sorta di ereditarietà tra gli oggetti con comportamenti e caratteristiche simili. Anche *Mask* deve rimanere vuota poiché indica se le collisioni per l'oggetto devono essere calcolate in base alla forma di uno *sprite* diverso da quello assegnato.

Fatto questo passiamo alla seconda sezione della finestra dove è possibile specificare a quali eventi deve rispondere l'oggetto. Premere sul tasto *Add Event* e cliccare la voce *Create* nella finestrella di dialogo che compare. Vedere **Figura 2**.

L'evento *Create* viene lanciato ogni volta, e solo una volta, al momento della creazione dell'oggetto all'interno della stanza.

In corrispondenza di questo è utile effettuare tutte le operazioni preliminari di cui si ha bisogno.

Il nostro *Shuttle*, per esempio, deve creare ed inizializzare le proprie variabili interne. Tramite

uno *script* faremo in modo che al momento della creazione dell'oggetto queste assumano i valori corretti. Nella sezione *Scripts* sotto la cartella *Partita/Shuttle* creiamo un nuovo *script* come quello riportato nel **Listato 1** chiamandolo *scriptImpostazioneParametriInizialiShuttle*.

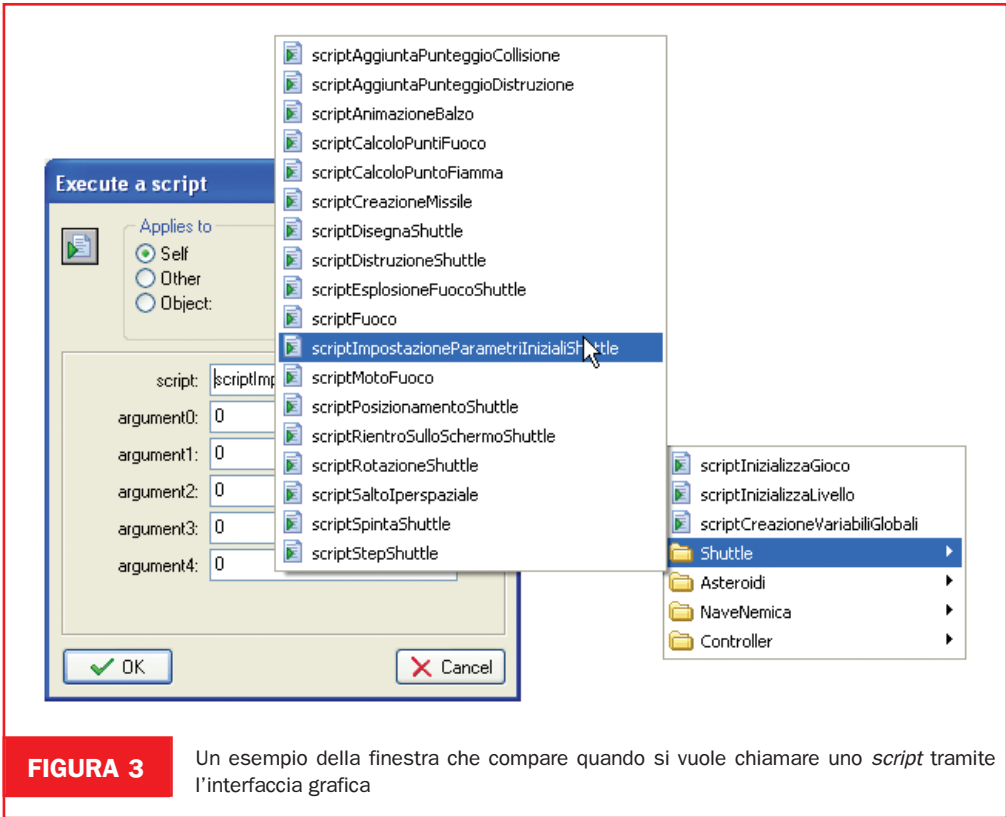
È importante chiarire che ogni *oggetto* possiede una certa quantità di variabili interne che determinano la sua posizione, la direzione e velocità di moto, la velocità di visualizzazione dei fotogrammi dello *sprite* ecc.

Nello *script* precedente abbiamo impostato a zero la velocità di visualizzazione dei fotogrammi dello *sprite*, poiché in condizioni normali essi verrebbero visualizzati in sequenza e continuamente. Abbiamo impostato a zero la sua direzione e velocità, mentre l'energia vitale *health* al massimo. Le altre variabili ci serviranno in seguito. Per fare in modo che lo *script* venga eseguito bisogna selezionare la cartella delle azioni chiamata *Control* e trascinare nella colonna *Actions* l'icona con la freccia verde dalla sezione *Code*. Comparirà una finestra come in **Figura 3** dove è possibile specificare lo *script* da associare all'azione.

Ora faremo in modo che lo *Shuttle* ruoti verso destra o sinistra secondo la pressione dei tasti freccia.

Aggiungiamo, quindi, due nuovi eventi cliccando su *Add Event/Key Press/Left* e poi *Add Event/Key Press/Right*.

Aggiungiamo un altro *script* che serve a cambiare la direzione dello *Shuttle* ogni volta che si premono i tasti freccia. Nel **Listato 2** è riportato quanto dobbiamo scrivere nello *scriptRotazioneShuttle*. Avrete notato che lo *script* non effettua modifiche alle variabili interne dell'oggetto, ma a quelle custom. Le costanti *SHUTTLE_ROT_A* e *SHUTTLE_ROT_B* vengono impostate nella finestra di dialogo *Global Game Settings*, raggiungibile dal menu ad albero, alla voce



LISTATO 1

Questo script imposta tutti i parametri relativi allo Shuttle con valori di default e verrà lanciato ogni qualvolta un'istanza di questo oggetto viene creata

```
{
  image_speed = 0;
  direction = 0;
  speed = 0;
  health = 100;

  motion_set(direction, speed);
  io_clear();

  global.SHUTTLE_VelocitaDiRotazione = 2;
  global.SHUTTLE_FattoreAccelerazione = 0.5;
  global.SHUTTLE_FattoreDecelerazione = 0.1;
  global.SHUTTLE_FlagAccelerazione = FALSE;
  global.SHUTTLE_Direzione = 0;
  global.SHUTTLE_VelocitaFuoco = 15;
  global.SHUTTLE_RaggioCollisione = sprite_get_width(sprShuttle) * 3;
  global.SHUTTLE_LivelloBalzo = 100;
  global.SHUTTLE_LivelloReattore = 100;
  global.SHUTTLE_Missili = SHUTTLE_NUMERO_MISSILI;
  global.SHUTTLE_ContatoreInizio = 3 * room_speed;
}
```

Constants. Seguiamo i soliti passi per agganciare lo script agli eventi Key Press/Left e Key Press/Right in più, questa volta, nella casella argument0 della finestra dei parametri inseriamo i valori SHUTTLE_ROT_A per Left e SHUTTLE_ROT_B per Right. A questo punto però lo Shuttle non è ancora in grado di ruotare su se stesso e questo compito verrà svolto da uno script che agganceremo in un altro evento. Lo Step è un evento che viene eseguito ad ogni passo del gioco; qui è possibile piazzare tutte le azioni che devono essere eseguite con una cadenza temporale regolare. Tale script è presentato nel Listato 3. In pratica quello

LISTATO 2

Lo script riceve come parametro una costante che indica verso quale direzione lo Shuttle deve essere ruotato. Vengono modificate opportunamente le variabili interne dell'oggetto

```
/* PARAMETRI:
argument0 : Direzione di rotazione dello Shuttle
SHUTTLE_ROT_A : Rotazione in senso antiorario
SHUTTLE_ROT_O : Rotazione in senso orario */
{
  // Cambio la direzione di spostamento dello Shuttle
  if (argument0 == SHUTTLE_ROT_A)
  {
    global.SHUTTLE_Direzione += global.SHUTTLE_VelocitaDiRotazione;
  }
  else if (argument0 == SHUTTLE_ROT_O)
  {
    global.SHUTTLE_Direzione -= global.SHUTTLE_VelocitaDiRotazione;
  }
}
```

che il codice fa non è altro che associare, ad ogni passo, il fotogramma corretto dello sprite Shuttle in funzione del valore contenuto nella variabile

```
global.SHUTTLE_Direzione
```

Il perché di questa scelta è semplice; l'intenzione è quella di evitare la duplicazione del codice per i due eventi di rotazione. La prima istruzione del listato setta la variabile image_index. Questa rappresenta l'indice del fotogramma dello sprite da visualizzare, ottenuto tramite l'espressione successiva, in base al numero di fotogrammi presenti nello sprite, ricavati dalla funzione

```
sprite_get_number(sprShuttle)
```

e alla direzione che deve assumere lo Shuttle. La chiamata

```
motion_set(direction, speed)
```

imposta la direzione e la velocità di moto, mentre la riga

```
if (health <= 0) instance_create(x,y,objEsplosioneShuttle)
```

crea un'istanza dell'oggetto objEsplosioneShuttle, alle coordinate attuali x e y dello Shuttle, quando termina la propria energia vitale. Facciamo ora in modo che lo Shuttle possa avanzare, perciò aggiungiamo l'evento Key Press/Up. In Figura 4 viene mostrato come è possibile ottenere tramite le azioni predefinite, una sorta di flusso di controllo, mentre nel Riquadro 1 è presente una descrizione verbale delle azioni eseguite. Per il momento abbiamo indicato semplicemente che il tasto UP è stato premuto richiamando lo scriptSpintaShuttle del Listato 4 il quale cambia i valori della variabile interna speed. I cambiamenti sono resi effettivi al prossimo passo dalla funzione motion_set all'interno dello scriptStepShuttle che abbiamo visto poco fa. Ora lo Shuttle è realmente autonomo ed in grado di muoversi in tutte

LISTATO 3

Qui vengono effettuate le operazioni periodiche come l'assegnazione del fotogramma dello sprite e del moto

```
{
  // Cambio il fotogramma dello sprite visualizzato in base alla
  // direzione e al numero complessivo di fotogrammi
  image_index = (sprite_get_number(sprShuttle) / 360)
                (global.SHUTTLE_Direzione - 90);

  // Comando all'oggetto di muoversi nella direzione e
  // alla velocità specificata
  motion_set(direction, speed);

  // Controllo dell'energia
  if (health <= 0) instance_create(x,y,objEsplosioneShuttle);
}
```

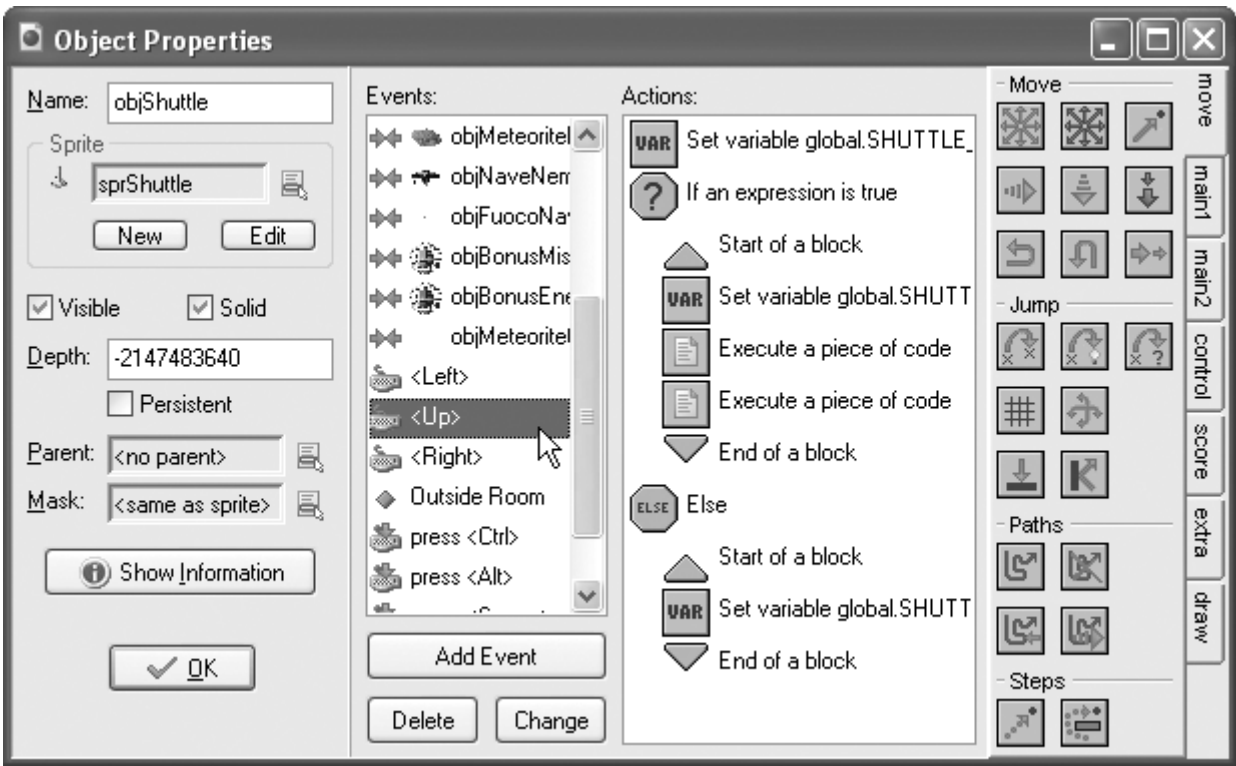


FIGURA 4 Questa immagine, ripresa dal progetto finito, mostra come è possibile creare flussi di controllo direttamente dell'interfaccia grafica

le direzioni all'interno dello schermo. Dato che la stanza è impostata in modo da essere più grande dello schermo visibile è necessario stabilire quale *oggetto* seguire negli spostamenti.
Nella tabella *views* delle proprietà della stanza *roomLivello* troviamo la voce *Object Following*. Impostandola su *objShuttle* siamo certi che questo verrà inquadrato sempre

al centro dello schermo. Ovviamente c'è molto altro da dire sullo Shuttle, ma credo che sia opportuno esplorare da soli il progetto e studiare come vengono effettuate le

LISTATO 4

Viene calcolata ed applicata la spinta di accelerazione/decelerazione allo Shuttle in base al parametro passato come argomento

```
{
  if (global.SHUTTLE_FlagAccelerazione == TRUE)
  {
    speed += (global.SHUTTLE_FattoreAccelerazione * (global.SHUTTLE_LivelloReattore/100));

    // Impedisco che il valore di spinta superi il massimo consentito
    if (speed > SHUTTLE_MAX_SPINTA) speed = SHUTTLE_MAX_SPINTA;
  }
  else if (global.SHUTTLE_FlagAccelerazione == FALSE)
  {
    speed -= global.SHUTTLE_FattoreDecelerazione;

    // Impedisco che il valore di spinta sia negativo
    if (speed < 0) speed = 0;
  }
}
```

LISTATO 5

Questo script gestisce la diminuzione di energia vitale ogni qualvolta un asteroide viene colpito

```
/* PARAMETRI
   argument0 : Oggetto con il quale è avvenuta la collisione
   argument1 : ID dell'istanza del meteorite
   argument2 : Rapporto di detrazione dell'energia */

{
  // Determino la quantità di energia da detrarre
  // in funzione dell'oggetto contro il quale è
  // avvenuta la collisione
  if (argument0 == objShuttle)
    detrazioneEnergia = 20;
  else if (argument0 == objFuocoShuttle)
    detrazioneEnergia = 5;
  else if (argument0 == objEsplosioneMissile)
    detrazioneEnergia = SHUTTLE_POTERE_MISSILE;

  // Recupero il livello di energia di questo oggetto,
  // ne detraggo la quantità opportuna e memorizzo il valore
  // nella mappa
  ds_map_replace(global.mappaEnergia, argument1,
    ds_map_find_value(global.mappaEnergia, argument1) -
    (detrazioneEnergia * argument2));
}
```

RIQUADRO 1

Una breve descrizione dei passi eseguiti dal flusso che gestisce la pressione della freccia in alto

```
Keyboard Event for <Up> Key:
set variable global.SHUTTLE_LivelloReattore relative to -SHUTTLE_
DIMINUZIONE_REATTORE
if expression global.SHUTTLE_LivelloReattore > 0 is true
  set variable global.SHUTTLE_FlagAccelerazione to TRUE
  direction = global.SHUTTLE_Direzione
  effect_create_above(ef_smoke,global.SHUTTLE_PuntoFiammaX,global.
SHUTTLE_PuntoFiammaY,0,c_lgray);
  effect_create_above(ef_smoke,global.SHUTTLE_PuntoFiammaX,global.
SHUTTLE_PuntoFiammaY,0,c_white);
else
  set variable global.SHUTTLE_LivelloReattore to 0
```

altre operazioni come lo sparo, il lancio dei missili, il controllo dei livelli di energia ecc.
Una volta completata la creazione delle azioni e script è possibile aggiungere l'oggetto nella stanza roomLivello posizionandolo a piacimento all'interno di essa.

Animare gli asteroidi

Gli asteroidi sono oggetti dai comportamenti molto semplici, essi infatti, possiedono un moto determinato da una direzione ed una velocità di spostamento. Hanno a disposizione una quantità di energia vitale che diminuisce ad ogni collisione con lo Shuttle. Quando l'energia vitale termina essi esplodono e, i più grandi, si frammentano in pezzi più piccoli.
Iniziamo dai meteoriti piccolissimi e ne creiamo uno nella cartella Asteroidi/Piccolissimi chiamato objMeteoritePiccolissimo1, con lo sprite sprMeteoritePiccolissimo1. Nell'evento Create inseriamo l'azione che si trova nella tabella delle azioni move rappresentata dalle otto frecce blu.
Nella casella direction, della finestra dei parametri, inseriamo la stringa direction, mentre nella casella speed inseriamo random(10).
Questo sta ad indicare che quando l'oggetto viene creato si deve muovere nella direzione indicata dalla sua variabile interna e ad una velocità casuale compresa tra 0 e 10 pixel per passo. Aggiungiamo ora gli eventi necessari a gestire le collisioni con lo Shuttle, i proiettili e i missili.
Premere su Add Event/Collision/Partita/Shuttle/objShuttle. la stessa cosa deve essere effettuata con gli oggetti objFuocoShuttle e objEsplosioneMissile. Nel Listato 5 è mostrato il codice dello scriptGestioneEnergiaAsteroidi da aggiungere ad ogni evento di collisione appena creato. Come parametri lo script riceve

LISTATO 6

Questo script interno serve a distruggere l'istanza di un oggetto quando questo termina la propria energia

```
{
  if (ds_map_find_value(global.mappaEnergia, self.id) <= 0 &&
global.SHUTTLE_ContatoreInizio <= 0)
  {
    // Cancello l'elemento dalla mappa
    ds_map_delete(global.mappaEnergia, self.id);

    // Effetto anello su collisione con astronave
    effect_create_above(ef_ring,x,y,0,c_lgray);

    // Genero l'effetto esplosione
    scriptEsplosioneAsteroidi(0);

    // Distruggo l'istanza dell'oggetto
    instance_destroy();
  }
}
```

- L'index dell'oggetto con il quale è avvenuta la collisione (objShuttle, objFuocoShuttle, ecc.)
- L'id dell'istanza del meteorite chiamante, ottenibile tramite il comando Self.id
- Il rapporto moltiplicativo di detrazione dell'energia vitale

A questo punto l'energia degli asteroidi diminuisce ad ogni collisione, ma cosa deve succedere quando questa termina? Osserviamo prima il Listato 6. Qui viene controllato il livello di energia e se risulta minore o uguale a zero viene generato l'effetto speciale di esplosione tramite lo scriptEsplosioneAste

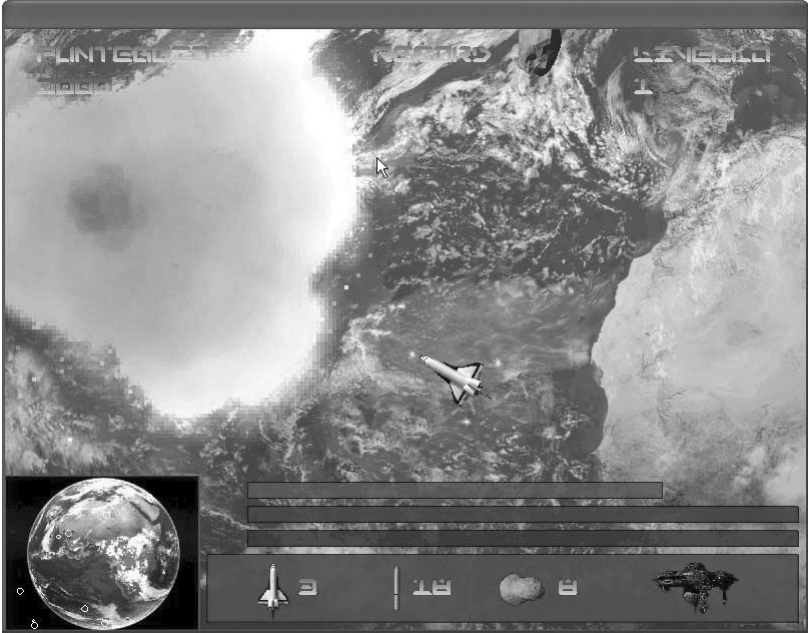


FIGURA 5

Un'immagine ripresa durante l'azione del gioco completo

LISTATO 7

Questo script viene lanciato al momento della creazione della nave nemica e si occupa di inizializzare i parametri

```
{
  global.NAVE_PosizioneX = random(room_width - sprite_get_
    width(sprNaveNemica) * 2) + sprite_get_width(sprNaveNemica);
  global.NAVE_PosizioneY = random(room_height - sprite_get_
    height(sprNaveNemica) * 2) + sprite_get_height(sprNaveNemica);
  global.NAVE_TimerFuoco = 0;
  global.NAVE_TimerSalto = 0;

  x = global.NAVE_PosizioneX;
  y = global.NAVE_PosizioneY;
  image_speed = 0.5;
}
```

roide e, alla fine, avviene la distruzione definitiva dell'istanza dell'oggetto. Per fare in modo che gli asteroidi più grandi si dividano è necessario aggiungere la linea

```
// Lancio lo script che genera gli
// asteroidi figli
scriptDividiAsteroide(objMeteoritePiccolo1);
```

al codice del Listato 6. Lo scriptDividiAsteroide che per questioni di spazio non riportiamo non fa altro che generare un numero casuale di nuove istanze di asteroidi più piccoli.

La nave nemica

Realizziamo ora l'oggetto di nome objNaveNemica. La nave nemica non possiede moto ma è in grado di effettuare balzi iperspaziali in continuazione e fare fuoco in direzione dello

LISTATO 8

Questo script viene lanciato quando la nave nemica viene distrutta e crea un effetto di esplosione

```
{
  psx = 0;
  psy = 0;

  // Crea gli effetti vari
  for (contatore = 0; contatore <= 5; contatore+=1)
  {
    psx = random(50)+x-25;
    psy = random(50)+y-25;
    effect_create_above(ef_explosion,psx,psy,2,make_color_rgb(0,
      random(127)+128, random(127)+128));
    effect_create_above(ef_ring,psx,psy,2,c_white);
  }

  if (round(random(2)) mod 2 == 0)
    instance_create(x,y,objBonusEnergia);
  else
    instance_create(x,y,objBonusMissili);

  // Eseguo il suono associato
  sound_play(sndEsplosioneNaveNemica);
}
```

Shuttle. Per questo oggetto devono essere creati gli eventi Create, Step, Collision/objShuttle, Collision/objFuocoShuttle e Collision/objEsplosioneMissile. Nell'evento Create immettere lo scriptInizializzaNaveNemica il cui codice è quello del Listato 7. Questo fa in modo da posizionare l'astronave nemica in una posizione casuale all'interno della stanza. Mentre la variabile image_index dimezza la velocità di visualizzazione dell'animazione. Nell'evento step sono inseriti due script che temporizzano i salti iperspaziali e il fuoco verso lo Shuttle. Negli altri eventi è presente lo scriptEsplosioneNaveNemica, dato che essa deve essere distrutta ad ogni collisione. Il Listato 8 mostra come si comporta questo script il quale crea alcuni effetti grafici, istanzia gli oggetti Bonus scelti a caso ed esegue un effetto sonoro tramite la funzione

snd_play

Il controller

Il controller è un oggetto come gli altri al quale viene delegato il compito di gestire ed organizzare le varie fasi di gioco, inizializzare le variabili globali, creare le istanze dei meteoriti ed altro. Gli eventi principali ai quali deve rispondere sono solitamente Create e Step.

Conclusioni

In Figura 5 possiamo vedere un'immagine del gioco finale. Purtroppo negli articoli non c'è abbastanza spazio per descrivere a fondo tutti i particolari necessari. Rimango però a disposizione per qualunque chiarimento risulti necessario, vi confesso che gradisco enormemente le vostre mail e soprattutto richieste di possibili implementazioni di videogiochi. Sul sito FTP potrete trovare anche un altro progetto, un clone del gioco Snake che vi darà ulteriori informazioni sull'utilizzo di Game Maker. Nei prossimi numeri tratteremo altri strumenti utili per sviluppare videogiochi ma per adesso non posso anticiparvi nulla.

Bibliografia

[1] Mark Overmars, "Designing Games with Game Maker", www.gamemaker.nl

Download del progetto

Scarica il progetto completo di questo articolo all'indirizzo: <ftp://ftp.infomedia.it/pub/DEV/Listati/Dev133/gamedev.zip>

Mauro Pesci

Si occupa di Analisi e Sviluppo del software in vari ambienti e piattaforme. Ha lavorato per diverse aziende nel settore, mentre ora è impegnato in un progetto presso la HP di Roma. Attualmente studia Informatica presso l'università di Tor Vergata (Roma).