21-01 Controlli al volo

01. Matrici di controlli

❖ Per iniziare, una matrice di pulsanti

Dichiarare una matrice di Button come segue:

```
Button [ , ] matrix;
matrix = new Button [8 , 8];
```

Questa matrice può essere riempita con controlli di tipo Button, ma ancora non ne contiene. Quindi ancora ciascuna cella della matrice non contiene pulsanti.

È possibile, tuttavia, assegnare un pulsante a una qualsiasi cella di matrix. Per esempio, se button1 è un pulsante presente nella Form1, allora è possibile eseguire l'istruzione seguente:

```
matrix [ 0 , 0 ] = button1;
```

si deve osservare che dopo aver eseguito l'istruzione precedente, i due "nomi" button1 e matrix[0,0] sono equivalenti, ovvero entrambi condividono lo stesso oggetto. Per esempio le istruzioni seguenti:

```
matrix [ 0 , 0 ] = button1;
matrix [ 0 , 0 ].Text = "Ciao";
```

modifica il testo del pulsante button1, anche se apparentemente non si è modificato il suo testo.

❖ Esercizio 1.

1. Creare una Form1 con 9 pulsanti, disposti in quadrato
2. Dichiarare una matrice di Button
3. Assegnare a ciascuna cella di matrice uno dei pulsanti della Form1
4. Associare un gestore di evento al pulsante button1, in modo che modifichi il testo di tutti i pulsanti con la scritta «X»
5. Associare agli altri pulsanti lo stesso gestore di evento, sfruttando la casella Eventi della finestra Proprietà

❖ Proseguiamo con la matrice di pulsanti

Dichiarare una matrice di Button come segue:

```
Button [ , ] matrix;
matrix = new Button [8 , 8];
```

Con un doppio clic sulla Form1 possiamo generare il gestore di eventi LoadForm. Adesso scriviamo nel gestore di eventi il codice seguente:

```
for (int r=0; r<8; r++)
      for (int c=0; c<8; c++)
            matrix [r , c] = new Button(); //crea un nuovo pulsante
```

Il codice scritto sopra è diverso dai precedenti: la new richiama un costruttore di pulsanti (sai usare la Random? Immagina una matrice di Random) che crea «al volo» un pulsante tutto nuovo, non ancora esistente nella Form1. Se proviamo a eseguire il programma, però, non compare nulla a video. Il motivo è che, pur esistendo i pulsanti nessuno ha specificato che debbano essere agganciati alla Form1. Modifichiamo leggermente il codice precedente come segue:

```
for (int r=0; r<8; r++)
      for (int c=0; c<8; c++)       {
            Matrix [r , c] = new Button(); //crea un nuovo pulsante
            Matrix [r , c].Text = ""; //ogni pulsante ha testo vuoto
            Matrix [r , c].Height = 25; //ogni pulsante è alto 25 pixel
            Matrix [r , c].Width = 25; //ogni pulsante è largo 25 pixel
            Matrix [r , c].Top = c*25; //posizione verticale ipotetica
            Matrix [r , c].Left = r*25; //posizione orizzontale ipotetica
      }
```


Autore: Andrea Zoccheddu

Il codice scritto sopra crea tanti pulsanti e per ciascuno imposta l testo e le dimensioni visive e la posizione a video. Ma se eseguiamo (prova) il codice ancora non si vede nulla!

Modifichiamo ancora il codice precedente come segue:

```
for (int r=0; r<8; r++)
      for (int c=0; c<8; c++)
      {
            ... //come prima
            Form1.Controls.Add(Matrix [r , c]);
      }
```

Il codice scritto finalmente aggancia i pulsanti alla finestra. Prova il progetto.

❖ Esercizio 2.

1. Dichiarare una matrice di Button e poi crearla al volo per formare il campo di gioco del tris
2. Ciascun pulsante deve mostrare un simbolo "_"

❖ Esercizio 3.

1. Dichiarare una matrice di ComboBox e poi crearla al volo per formare un campo da gioco tipo tris
2. Ciascun controllo deve offrire un elenco delle seguenti scelte: _, X, O.

❖ Agganciare gestori «al volo»

Dopo aver dichiarato la solita matrice di Button [8x8] e dopo aver previsto il metodo LoadForm per crearla a volo, aggiungiamo un altro metodo; si scriva in un punto qualsiasi del programma il seguente gestore di evento (non agganciato a nessun controllo):

```
public void MioGestore (Object sender,   EventArgs e)
{
  BackColor = Color.Red;
}
```

Adesso modifica il metodo LoadForm come segue:

```
for (int r=0; r<8; r++)
      for (int c=0; c<8; c++)
      {
            ... //come prima
            Matrix [r , c].Click += MioGestore; //senza parentesi tonde!
            Form1.Controls.Add(Matrix [r , c]);
      }
```

A questo punto dovresti avere un programma funzionante.

❖ Esercizio 4.

1. Dichiarare una matrice di Button 8x8
2. Dichiarare un gestore di evento Colorante che colora di rosso la Form1
3. Col gestore LoadForm crearla al volo per formare una scacchiera
4. Ogni pulsante deve:
   - Mostrare nel testo un numero formato da due cifre, la prima è l'indice di riga, la seconda quello di colonna
   - Avere dimensioni quadrate, da 50 pixel
   - Il gestore per l'evento Click agganciato al metodo Colorante
5. È inutile proseguire se non svolgi gli esercizi!

02. Dentro i Gestori di Evento (Event Handlers)

❖ Riconoscere il controllo chiamante

Abbiamo visto come sia possibile agganciare un gestore di evento ad un controllo.

```
Matrix [r , c].Click += MioGestore; //senza parentesi tonde!
```


Autore: Andrea Zoccheddu

L'assegnazione speciale += è indispensabile per agganciare il metodo; se usi la pura assegnazione si solleva un errore; non è l'occasione giusta per trattare i motivi di questo errore, ma puoi riflettere sul fatto che, invece di assegnare valori a variabili, stiamo assegnando pezzi di programma (detti metodi) a contenitori speciali (gestori di evento).

Abbiamo anche visto che a ciascun pulsante (ovvero a ogni cella della matrice di pulsanti) abbiamo assegnato lo stesso metodo per gestire l'evento: quindi tutti i pulsanti fanno la stessa cosa!

Apparentemente è una limitazione terribile, perché vorremmo che ciascun pulsante facesse una cosa diversa. In effetti è possibile scrivere del codice dentro il metodo per riconoscere il controllo che ha causato l'evento per esempio riconoscere il pulsante che è stato cliccato.

Torniamo al programma visto in precedenza (aprilo o ricostruiscilo da capo); modifichiamo il metodo gestore come segue:

```
public void MioGestore (Object sender, EventArgs e)
{
  (sender as Button).BackColor = Color.Red;
}
```

Come si vede la proprietà BackColor adesso non è quella della Form1, ma quella di un altro oggetto.

In particolare la proprietà BackColor è quella di uno strano controllo tra parentesi tonde:

```
  (sender as Button)
```

Ma che cosa è questa strana scrittura? In estrema sintesi la scrittura si può interpretare con la seguente frase: «considera sender come se fosse un Button». E quindi essa possiede tutte le proprietà del Button, tra cui BackColor.

Vediamo un po' meglio che cosa succede.

❖ Il parametro sender

La parola sender che appare tra parentesi tonde in ogni metodo gestore di evento rappresenta l'oggetto che ha causato l'evento; per dirla più semplicemente, ogni volta che un pulsante riceve un clic, esegue un metodo gestore che, tra parentesi ha sempre questo misterioso sender; il sender è il pulsante che è stato cliccato.

Poiché però non esistono solo pulsanti ma vari controlli, il sender è versatile ed è dichiarato come generico Object che pressappoco significa «qualsiasi cosa».

Nel nostro programma ogni cella della matrice è un pulsante; ed ogni pulsante è agganciato ad un gestore per l'evento Click; ed il gestore ha un sender che contraddistingue l'esatto pulsante che ha causato il Click. Il sender quindi è sempre un pulsante, ma il programma non lo sa e lo considera un qualsiasi controllo. Allora noi dobbiamo convincere il programma che si tratta proprio di un pulsante (Button).

❖ L'operatore as

La parola as che usiamo all'interno del gestore significa «come». Si usa spesso insieme alle parentesi tonde, per consentire l'uso di proprietà e metodi tipici del controllo desiderato. Quando si scrive:

```
  (sender as Button)
```

stiamo indicando al programma che l'oggetto sender deve essere usato «come se» fosse un Button.

Se sender è usato come un Button, allora ha a disposizione tutte le proprietà del Button, tra cui è presente anche la proprietà BackColor. Quindi è possibile scrivere l'istruzione seguente:

```
  (sender as Button).BackColor = Color.Red;
```

L'istruzione significa: considera sender come un Button e quindi usa la sua proprietà BackColor; e assegna alla proprietà BackColor il colore rosso di sistema.

Autore: Andrea Zoccheddu

❖ Esercizio 5.

1. Dichiarare una matrice di Button 8x8
2. Dichiarare un gestore di evento Messaggio che mostra in un messaggio i valori di Top e Left del sender, forzando il tipo come se fosse un Button;
3. Col gestore LoadForm crearla al volo per formare una scacchiera
4. Ogni pulsante deve:
   - Avere il testo vuoto (senza scritte)
   - Avere dimensioni quadrate, da 50 pixel
   - Il gestore per l'evento Click agganciato al metodo Messaggio
5. Esegui il programma e provalo
6. Modifica il gestore di evento Messaggio che, invece del messaggio, modifica il testo del pulsante stesso riportando le sue posizioni Top e Left
7. Esegui il programma e provalo

❖ Nascondere informazioni nella proprietà Tag

Quando costruiamo un controllo, possiamo verificare che esso possiede una particolare proprietà che si chiama Tag e che non serve a niente. La proprietà Tag permette di conservare delle informazioni (un dato di tipo string) senza che abbia una funzione specifica. Possiamo quindi usarla per scriverci dentro delle informazioni nascoste.

Per esempio in fase di creazione dei controlli possiamo scrivere:

```
for (int r=0; r<8; r++)
      for (int c=0; c<8; c++)
      {
            ... //come prima
            Matrix [r , c].Tag = Convert.ToString(r*10+c);
            Matrix [r , c].Click += MioGestore; //senza parentesi tonde!
            Form1.Controls.Add(Matrix [r , c]);
      }
```

Come risultato, avremo un numero a due cifre di cui la prima rappresenta l'indice di riga e la seconda quello di colonna.

Nel gestore di evento potremmo quindi scrivere un codice come il seguente:

```
public void MioGestore (Object sender, EventArgs e)
{
  MessageBox.Show( (sender as Button).Tag );
}
```

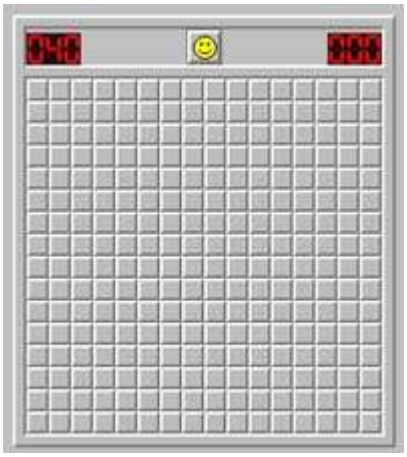
❖ Esercizio 6.

1. Dichiarare una matrice di Button 8**X**8
2. Dichiarare un gestore di evento Messaggio che copia nel testo del pulsante il suo Tag
3. Col gestore LoadForm crearla al volo per formare una scacchiera **8X8**
4. Ogni pulsante deve:
   - Avere il testo vuoto (senza scritte)
   - Avere dimensioni quadrate, da 50 pixel
   - Nel Tag un numero a due cifre che rappresenta le coordinate di matrice
   - Il gestore per l'evento Click agganciato al metodo Messaggio
5. Esegui il programma e provalo

Autore: Andrea Zoccheddu

Esercizi

❖ Esercizio 1.



- Crea un progetto visuale
- All'avvio del programma si crea «al volo» una matrice di pulsanti come nella figura qui a lato (16 x 16)
- Provare a realizzare i seguenti metodi gestore Click:
  1. Quando si esegue un clic su un qualsiasi pulsante, questo scompare!
  2. Quando si esegue un clic su un qualsiasi pulsante, questo non scompare ma si colora di verde!
  3. Quando si esegue un clic su un pulsante, si colora di rosso se la somma delle sue coordinate è pari, altrimenti si colora di verde!
  4. Quando si esegue un clic su un pulsante, questo mostra nel suo testo quante volte è stato cliccato!

http://it.wikipedia.org/wiki/Campo_minato_(videogioco)

❖ Esercizio 2.

- Crea un progetto visuale
- All'avvio del programma si crea «al volo» una matrice di pulsanti come nella figura qui a lato (8 x 8)
- Le caselle sono colorate alternativamente (chiare e scure)
- Nelle righe laterali i pulsanti devono mostrare una immagine (Gliph) presa dagli scacchi
- Nelle righe vicine alle laterali i pulsanti devono mostrare una immagine (Gliph) pedine degli scacchi
- Provare a realizzare i seguenti metodi gestore Click:
  1. Quando si esegue un clic su un qualsiasi pulsante, dice se c'è un pezzo
  2. Quando si esegue un clic su un qualsiasi pulsante, dice quale pezzo c'è

❖ Esercizio 3.

- Crea un progetto visuale
- All'avvio del programma si crea «al volo» una matrice di pulsanti come nella figura qui a lato (7 x 7)
- Le caselle non sono colorate
- Alcune caselle sono non usate (angoli)
- Altre caselle mostrano una immagine
- Provare a realizzare i seguenti metodi gestore Click:
  1. Quando si esegue un clic su un qualsiasi pulsante, dice se c'è un pezzo e memorizza la sua posizione in variabili globali;

http://sodilinux.itd.cnr.it/sdl6x3/scheda.php?stile=cl&id=5219

Gioco derivato dal classico gioco della Dama Cinese. L"obiettivo del gioco è eliminare tutti i pinguini, tranne uno, per poter passare al livello successivo. I pinguini vengono eliminati, come nel classico gioco della dama, quando è possibile "saltarli" andando a posizionarsi in una casella vuota adiacente. Il gioco implica la messa in atto di una serie di strategie che permettano di "non lasciarsi indietro" pinguini che successivamente non si potranno più eliminare, impedendo di accedere al livello successivo.
Dal punto di vista didattico il gioco risente del fatto che non è possibile, come invece in analoghe versioni informatiche della dama cinese, tornare sui propri passi e cancellare l'ultima mossa fatta: è solo possibile ricominciare da capo con lo stesso schema.

Autore: Andrea Zoccheddu

Sommario

21-01 Controlli al volo 1

01. Matrici di controlli 1

- Per iniziare, una matrice di pulsanti 1
- Esercizio 1. 1
- Proseguiamo con la matrice di pulsanti 1
- Esercizio 2. 2
- Esercizio 3. 2
- Agganciare gestori «al volo» 2
- Esercizio 4. 2

02. Dentro i Gestori di Evento (Event Handlers) 2

- Riconoscere il controllo chiamante 2
- Il parametro sender 3
- L'operatore as 3
- Esercizio 5. 4
- Nascondere informazioni nella proprietà Tag 4
- Esercizio 6. 4

Esercizi 5

- Esercizio 1. 5
- Esercizio 2. 5
- Esercizio 3. 5

Autore: Andrea Zoccheddu