

8. Approfondimenti

8.1. Aggiungere eventi da oggetti standard

Combinando classi personalizzate, che usano oggetti standard (ossia una o più proprietà della classe sono variabili di tipo oggetto), ed eventi è possibile ottenere dei risultati sorprendenti e al contempo molto utili.

Supponiamo, ad esempio, di aver la necessità di dover modificare molto spesso la scala dell'asse verticale di un grafico creato in un foglio Excel. Per evitare di dover operare manualmente, sarebbe utile associare ad un evento (ad esempio il click del mouse sul grafico) una procedura che provvede automaticamente a modificare la scala.

Nulla di più facile: l'oggetto chart espone degli eventi ai quali possiamo associare tutte le procedure che vogliamo. Per farlo è sufficiente:

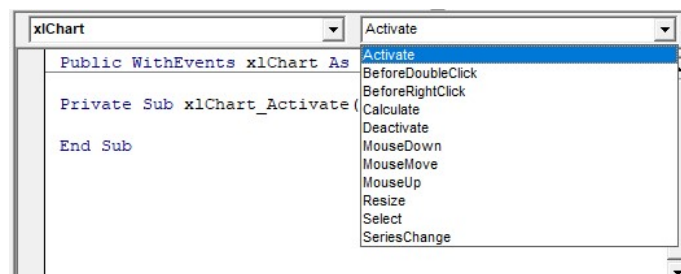
- Creare un modulo di classe contenente un oggetto chart;
- Dichiarare tale oggetto con il prefisso WithEvents, di modo da poter accedere ai suoi eventi;
- Scrivere le procedure associate agli eventi di nostro interesse.

In pratica si tratta di creare un nostro tipo di grafico personalizzato che è in toto uguale a quello standard, ma che esegue un'azione specifica (ridimensionamento della scala) ogni volta che si clicca su di lui.

Vediamo nel dettaglio come procedere. Come detto iniziamo a creare un modulo di classe, rinominiamolo Cls_Chart e definiamo la seguente variabile pubblica di tipo oggetto:

```
Public WithEvents xlChart As Chart
```

Fatto questo, possiamo vedere subito gli eventi esposti da tale oggetto. Basta infatti selezionarlo nel menu a tendina di destra (posto sopra al foglio di sviluppo) e tutti i suoi eventi appaiono nel menu a tendina di sinistra, come visibile nella figura sottostante.



Eventi dell'oggetto Chart

A questo punto selezioniamo l'evento MouseDown che ci permetterà di associare del codice sia all'evento legato al click sul tasto destro del mouse, sia al click su quello sinistro. Selezionandolo, VBA crea la seguente ossatura del codice:

```
Private Sub xlChart_MouseDown(ByVal Button As Long, ByVal Shift As Long, _  
    ByVal x As Long, ByVal y As Long)  
[...]  
End Sub
```

Come si vede, anche in questo caso la procedura associata all'evento ha come nome il nome dell'oggetto seguito dal nome dell'evento (separato dall'underscore).

I parametri rilasciati dall'evento sono: Button che vale 1 nel caso in cui sia stato premuto il tasto sinistro, 2 per il tasto destro, Shift che ci dice se è stato premuto il tasto Shift, x e y che ci danno le coordinate in cui si trovava il cursore del mouse al momento del click.

A questo punto possiamo scrivere il codice che ci permette di modificare la scala:

```
Private Sub xlChart_MouseDown(ByVal Button As Long, ByVal Shift As Long, _
    ByVal x As Long, ByVal y As Long)

    Sc = Me.xlChart.Axes(xlValue).MaximumScale 'Axes(xlValue) restituisce l'asse vertical, MaximumScale la scala

    If Button = 1 Then 'Left Button
        Me.xlChart.Axes(xlValue).MaximumScale = Sc * 0.75
    End If

    If Button = 2 Then 'Right Button
        Me.xlChart.Axes(xlValue).MaximumScale = Sc * 1.25
    End If
End Sub
```

Brevemente, si registra la scala corrente nella variabile Sc e, in seguito, la si incrementa o riduce del 25% in funzione del pulsante cliccato dall'utente.

Abbiamo quasi finito, resta un'ultima cosa. Come visto, l'evento MouseDown considera sia l'evento "click sul tasto destro", sia l'evento "click sul tasto sinistro". Ora il primo di tali eventi è un evento di per sé stante e, tale evento si verifica anche nel caso di doppio click. Risulta allora necessario disabilitare tali eventi con il codice seguente:

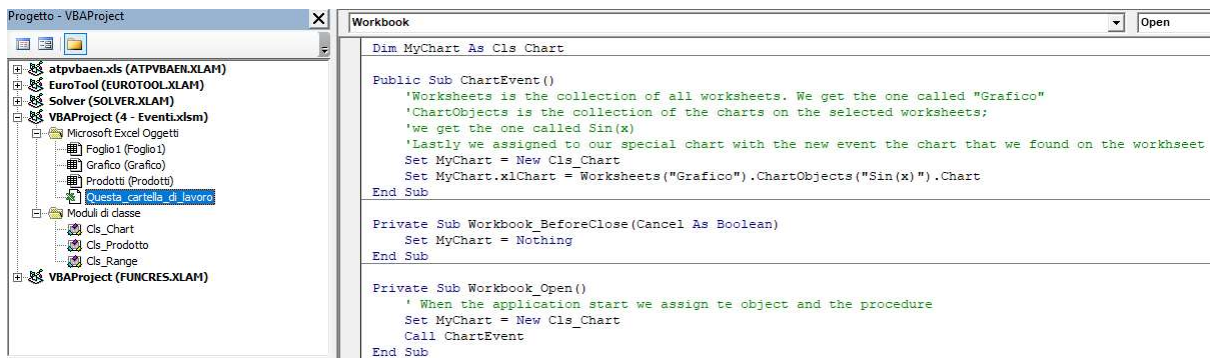
```
Private Sub xlChart_BeforeDoubleClick(ByVal ElementID As Long, _
    ByVal Arg1 As Long, ByVal Arg2 As Long, Cancel As Boolean)
    Cancel = True 'Deactivate Double Click
End Sub

Private Sub xlChart_BeforeRightClick(Cancel As Boolean)
    Cancel = True 'Deactivate Right Click
End Sub
```

Il nostro grafico modificato è pronto, si tratta di utilizzarlo¹¹. Supponiamo ora che il grafico di nostro interesse si chiami Sin(x) e che sia posto nel foglio di lavoro "Grafico". Per fare in modo che tale grafico si comporti come il nostro oggetto modificato Cls_chart dobbiamo creare un oggetto di tipo Cls_chart e di assegnare alla sua proprietà xlChart (la variabile di tipo oggetto chart) il grafico "Sin(x)" posto nel foglio di lavoro "Grafico".

Possibilmente tale assegnazione andrebbe fatta nello spazio di lavoro relativo alla cartella di lavoro (vedi figura sottostante).

¹¹ Abbiamo sempre detto che il codice legato ad un evento è esterno alla classe che rilascia l'evento. In questo caso sembra che non sia così. Abbiamo infatti scritto il codice che verrà attivato dall'evento MouseDown internamente alla definizione della nostra classe. Per quanto possa sembrare strano, questo è assolutamente concorde con quanto detto sino ad ora. L'evento che stiamo utilizzando è infatti relativo all'oggetto Chart, oggetto che stiamo utilizzando all'interno della nostra classe Cls_Chart, ma che non abbiamo ridefinito. Proprio qui sta il punto, utilizzando un oggetto all'interno di un altro oggetto (oggetto contenitore) possiamo creare dei metodi legati all'oggetto contenitore basati sugli eventi rilasciati dall'oggetto contenuto. In pratica, quindi, il codice che opera sull'evento è effettivamente esterno all'oggetto Chart, ma interno al nostro oggetto personalizzato Cls_chart



Il codice posto nello spazio di lavoro della cartella di lavoro

```

Dim MyChart As Cls_Chart 'La variabile oggetto

Public Sub ChartEvent() 'La procedura che effettua l'assegnamento
    Set MyChart.xlChart = Worksheets("Grafico").ChartObjects("Sin(x)").Chart
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean) 'Evento che si scatena alla chiusura del file
    Set MyChart = Nothing
End Sub

Private Sub Workbook_Open() 'Evento che si scatena all'apertura del file
    ' When the application start we assign te object and the procedure
    Set MyChart = New Cls_Chart
    Call ChartEvent
End Sub

```

Come si vede:

- Abbiamo creato una variabile MyChart di tipo Cls_Chart
- Abbiamo creato una procedura (ChartEvent) che assegna il grafico Sin(x) posto sul foglio "Grafico" alla proprietà (variabile di tipo oggetto) .xlChart della variabile MyChart. Questo crea il collegamento e, pertanto, l'evento MouseDown associato al grafico Sin(X) produrrà l'effetto voluto. Si noti che Worksheets è la collection di tutti i fogli di lavoro del file excel corrente. Similmente, ChartObjects è la collection di tutti i grafici aperti sul foglio corrente. In pratica quindi, la scrittura `Worksheets("Grafico").ChartObjects("Sin(x)").Chart` corrisponde a dire: "Tra tutti i fogli di lavoro prendi il foglio di nome "Grafico"; tra i suoi grafici prendi quello di nome "Sin(x)" e restituiscilo proprio come oggetto di tipo Chart".
- Abbiamo anche associato due righe di codice all'evento Workbook_Open, evento che si attiva all'apertura del file. In pratica, appena il file si apre, ci preoccupiamo di creare la variabile MyChart e di chiamare la procedura ChartEvent che completa l'accoppiamento con il grafico Sin(x).

8.2. Ancora sul come sfruttare eventi di oggetti predefiniti

Vediamo un secondo esempio simile al precedente, che mostra come sia possibile creare oggetti personalizzati aggiungono eventi ad oggetti standard di Excel (la stessa cosa vale per qualsiasi oggetto predefinito, specialmente se di tipo grafico).

Questa volta vogliamo ottenere il seguente comportamento personalizzato: modificando il contenuto della cella A1 del Foglio 1, vogliamo che tale cella si colori e che nelle celle ad essa contigue vengano visualizzate le informazioni inerenti colore e contenuto della suddetta cella A1. Premettiamo che per fare questo sarebbe sufficiente lavorare sugli eventi esposti dal Foglio 1¹², ma a scopi didattici, complichiamo un po' la procedura per mostrare come sia possibile creare oggetti che ampliano lo spettro degli eventi di cui sono dotati gli oggetti standard di Excel (in questo caso l'oggetto standard è l'oggetto Range, ossia un insieme di celle del foglio di lavoro).

Come prima creiamo un modulo di classe chiamato Cls_Range e dotiamolo di una proprietà (variabile) pubblica, di due variabili private e di un evento, come mostrato di seguito:

```
Private Rng As Range 'L'oggetto Range
Private Col As Integer 'Il colore
Private MyName As String 'Il nome
Public Event Select(Cell As Range) 'L'evento che restituisce un range
```

Aggiungiamo le routine proprietà relative alle variabili private come mostrato di seguito:

```
Public Property Set My_Range(objRng As Range)
    Set Rng = objRng
    RaiseEvent Select(Rng) 'Quando si assegnerà il range si genererà l'evento che restituisce un range
End Property

Public Property Get My_Range() As Range
    Set My_Range = Rng
End Property

Property Let Nome(nom As String)
    Nm = nom
End Property

Property Get Nome() As String
    Nome = Nm
End Property

Property Let Colore(C As Integer)
    Col = C
End Property

Property Get Colore() As Integer
    Colore = Col
End Property
```

¹² Ecco un altro esercizio lasciato al lettore

Si noti l'uso della property Set necessaria a creare l'oggetto Range e l'istruzione RaiseEvent all'interno di questa routine proprietà. In questo modo, nel momento stesso in cui si assegnerà un range al nostro oggetto, si scatterà l'evento Select.

Aggiungiamo infine due metodi, che sfruttano i metodi predefiniti dell'oggetto Range per assegnargli il nome ed il colore. Questo è mostrato di seguito:

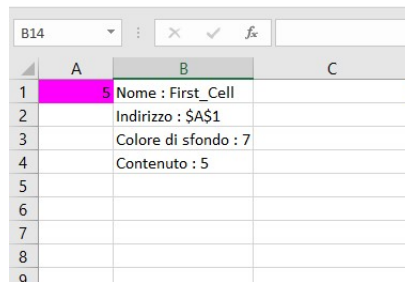
```
Public Sub Ass_Colore()  
    Rng.Interior.ColorIndex = Col 'Assegna il colore  
End Sub  
  
Public Sub Ass_Nome()  
    Rng.Name = MyName 'Assegna il nome  
End Sub
```

Bene, il nostro oggetto è pronto. Ora si tratta di utilizzarlo. Per farlo possiamo scrivere il seguente codice nello spazio di lavoro dedicato al foglio 1.

```
Private WithEvents Rng As Cls_Range  
  
Private Sub Rng_Sel(Cell As Range) 'Il codice associato all'evento  
    'Assegniamo un nome fisso e un colore random  
    Rng.Nome = "First_Cell"  
    Rng.Colore = Application.WorksheetFunction.RandBetween(0, 100)  
    If Rng.Colore < 1 Or Rng.Colore > 56 Then  
        MsgBox "Errore! Inserire un valore valido per ColorIndex compreso tra 1 e 56"  
        Exit Sub  
    End If  
    'Assegniamo nome e colore al range  
    Rng.Ass_Colore  
    Rng.Ass_Nome  
    i = Rng.Color  
    Rng.My_Range.Select 'Si seleziona il range questo consente di usare la funzione Offset, equivalente di Scarto  
    'Qui scriviamo tutte le informazioni nelle celle adiacenti  
    Selection.Offset(0, 1).value = "Nome : " & Rng.Nome  
    Selection.Offset(1, 1).value = "Indirizzo : " & Selection.Address  
    Selection.Offset(2, 1).value = "Colore di sfondo : " & i  
    Selection.Offset(3, 1).value = "Contenuto : " & Selection.value  
End Sub  
  
Private Sub Worksheet_Change(ByVal Target As Range) 'Evento attivato dal cambio del valore in una cella  
On Error GoTo ErrorHandler  
    Set Rng = New Cls_Range  
    If Target.Address = Range("A1").Address Then 'Target è la cella che ha cambiato contenuto. Consideriamo solo A1  
        Set Rng.My_Range = Target 'si assegna la cella modificata al nostro oggetto rng; l'evento viene scatenato!  
    Else  
        Exit Sub  
    End If  
ErrorHandler:  
    Application.EnableEvents = True  
End Sub
```

Come si vede (ci si aiuti con i commenti in verde), si sfrutta l'evento Change del foglio di lavoro per creare un oggetto personalizzato che agisce su un range. In particolare, tale evento restituisce l'oggetto Target, oggetto corrispondente alla cella che ha subito una modifica. Nel codice usiamo tale fatto a nostro vantaggio e, infatti, proseguiamo se e solo se Target (ossia la cella modificata) è la cella A1 (si noti che il controllo viene fatto tramite la proprietà Address).

Nel momento in cui si assegna Target a Rng.My_range si scatena l'evento e, conseguentemente si attiva la procedura Rng_Sel che genera un colore casuale per la cella A1, la colora e scrive nelle celle adiacenti una descrizione di A1, come mostrato nella figura sottostante:



	A	B	C
1	5	Nome : First_Cell	
2		Indirizzo : \$A\$1	
3		Colore di sfondo : 7	
4		Contenuto : 5	
5			
6			
7			
8			
9			

Il risultato dell'evento

8.3. Il problema delle 8 Regine – Classi, ricorsione e backtracking

Concludiamo questo rapido excursus di programmazione con un classico problema combinatoriale, il problema delle 8 regine, che ci darà modo di rivedere l'utilizzo di una classe, del concetto di ricorsione e di backtracking.

Iniziamo enunciando il problema. Consideriamo una scacchiera 8x8 (generalizzabile al caso NxN) e cerchiamo tutti i possibili modi in cui possiamo porre 8 regine (o N nella versione generalizzata) sulla scacchiera di modo che nessuna regina possa essere mangiata dalle altre e che, al contempo, nessuna regina possa muoversi senza esporsi all'attacco delle altre regine. Si tratta, in pratica, di trovare tutte le configurazioni di stallo. Risolvere il problema in via iterativa non è particolarmente difficile: con una scacchiera 8x8 occorrerebbero 8 cicli for concatenati tramite cui posizionare la prima regina in una cella (a partire dalla cella (1,1)), provare a posizionare la seconda nelle rimanenti celle non minacciate dalla prima regina e così via sino a posizionare l'ottava. Tuttavia, se le dimensioni della scacchiera sono ignote, tale approccio diventa impossibile. Serve allora pensare ad un approccio ricorsivo.

Prima di procedere oltre ricordiamo che la regina si sposta sia come una torre (può muoversi in verticale e in orizzontale di quante caselle vuole), sia come un alfiere (può muoversi lungo le diagonali della scacchiera di quante caselle vuole). Per tale motivo, risulta subito ovvio che qualsiasi soluzione fattibile deve prevedere che su ogni riga e su ogni colonna della scacchiera ci sia una e una sola regina. Quest'osservazione semplifica la codifica della soluzione, dato che, invece che una matrice NxN, la nostra soluzione può essere sintetizzata con un vettore B , il cui generico elemento $B[i]$ contiene il valore della colonna su cui è collocata la regina assegnata alla riga i -esima. Ad esempio, il vettore (1,2,3,4) indica una disposizione diagonale, ossia: la prima regina è posta nella prima riga e nella prima colonna, la seconda nella seconda riga e nella seconda colonna e così via per le altre.

A fronte di ciò, se indichiamo con $F(B)$ la funzione ricorsiva che riceve in input il vettore B contenente una soluzione parziale del problema, il funzionamento di F sarà qualcosa del genere:

- Si parte da una soluzione vuota $B = []$ e si chiama la funzione $F(B)$;
- Si colloca una regina nella riga $(Q+1)$ dove $Q = \text{Ubound}(B)$ è il numero delle regine che sono già state posizionate sulla scacchiera. Ovviamente alla prima chiamata $Q = 0$ e la prima regina sarà posizionata nella prima riga; alla seconda chiamata la seconda regina sarà posizionata nella seconda riga e così via.
- Per stabilire in che colonna posizionare la regina si esegue un ciclo progressivo su tutti i valori di ' i ', per ' i ' che va da 1 a N , dove N è il numero di colonne della scacchiera. Se si trova una cella sulla riga $(Q+1)$ e sulla colonna ' i ' che non è sotto attacco dalle precedenti regine, si piazza qui la nuova regina, si aggiorna B e si effettua una nuova chiamata a $F(B)$. Se tutto va bene, le chiamate successive porteranno a generare una soluzione.
- Viceversa, se non si trova una cella "sicura", la configurazione sinora generata e codificata nel vettore B non può portare a nessuna soluzione valida. Prima di procedere oltre bisogna allora ricondursi all'ultima configurazione generata, in questo senso si parla di *backtracking*. Facciamo un esempio. Supponiamo che la terza regina sia stata collocata sulla colonna x_3 della terza riga, ossia $B = (x_1 \ x_2 \ x_3)$. Se la chiamata $F(B)$ non trova nessuna collocazione per la quarta regina, significa che la configurazione parziale $B = (x_1 \ x_2 \ x_3)$ non può generare una soluzione. È inutile andare avanti e, anzi, bisogna tornare sui propri passi aggiornando il vettore B . A tal fine si modifica l'ultimo inserimento fatto, ossia si modifica la posizione assegnata all'ultima regina; tale regina verrà posizionata in una colonna sicuramente maggiore di x_3 (dato che le colonne precedenti saranno già state valutate), ma non necessariamente la $(x_3 + 1)$ perché quella potrebbe essere sotto attacco. Se si trova una cella sicura, detta \tilde{x}_3 la regina viene

posizionata lì, B diventa $B = [x_1, x_2, \tilde{x}_3]$ e la funzione F viene nuovamente invocata. Viceversa, se non si trova una cella valida, occorre fare un secondo step indietro cercando di modificare la posizione x_2 assegnata alla seconda regina. E così via.

Osserviamo che, per poter trovare tutte le possibili soluzioni, il backtracking viene eseguito anche nel caso in cui si sia trovata una soluzione. In pratica, trovata una soluzione, si prova a ripartire modificando la posizione assegnata all'ultima regina e così via. In questo modo vengono generate esaustivamente tutte le soluzioni (siano esse valide siano esse non valide), procedendo in maniera ricorsiva.

Vediamo come sia possibile implementare tale strategia con VBA. Per farlo iniziamo a creare una classe `Cls_Board` che: (i) aggiunge una regina nella prima riga vuota, (ii) toglie l'ultima regina, (iii) verifica se una cella è sicura, (iv) mostra a video la soluzione (rappresenta graficamente lo stato della scacchiera). Il codice è il seguente:

```
Private State As Collection 'Usiamo una collection anziché un vettore B per definire una sotto-soluzione
Public Size As Integer 'La dimensione della scacchiera

Private Sub Class_Initialize()
    Set State = New Collection
End Sub

Public Function N_Queens()
    N_Queens = State.Count
End Function

Public Sub Next_Row(Column As Integer) 'Aggiunge il valore della colonna in cui piazzare la nuova regina
    State.Add Column
End Sub

Public Sub Remove_Row(Column As Integer) 'Toglie l'ultima regina piazzata
    Dim Pos As Integer
    Pos = State.Count
    State.Remove Pos
End Sub

Public Function IS_Safe(Column) As Boolean 'La funzione che verifica se una cella è sicura o sotto attacco
    Dim Row As Integer, r As Integer, C As Integer

    IS_Safe = False
    Row = N_Queens + 1
    'Controlla la riga
    For r = 1 To State.Count
        C = State(r)
        If C = Column Then Exit Function
    'Controlla la diagonale
    If Abs(C - Column) = Abs(r - Row) Then Exit Function
    Next r
```



```

IS_Safe = True
End Function

Public Sub Show() ' Mostra la scacchiera con una stringa del tipo -- Q --
Dim r As Integer, C As Integer
Dim Brd As String
For r = 1 To Size
    For C = 1 To Size
        If Get_Queen(r, C) Then
            Brd = Brd & " Q "
        Else
            Brd = Brd & " - "
        End If
    Next C
    Brd = Brd & vbNewLine
Next r
Debug.Print Brd
Debug.Print St
End Sub

Private Function Get_Queen(r As Integer, C As Integer) As Boolean ' Verifica che ci sia una regina in (r,c)
On Error GoTo Err:
    Get_Queen = False
    If State(r) = C Then
        Get_Queen = True
        Exit Function
    End If
Err:
'Do nothing
End Function

Private Function St() As String ' Codifica la scacchiera come [(r,c), (r,c), ...] posizione di ogni regina
Dim V As Variant
Dim r As Integer
Dim Tuple As String

r = 1
St = "State Vector ["

For Each V In State
    Tuple = "(" & r & ", " & V & "),"
    St = St & Tuple
    r = r + 1
Next V
St = Left(St, Len(St) - 2) & "]"
End Function

```

Si noti solo il modo con cui vengono controllate le diagonali. A tal fine si usa la seguente espressione:

```
If Abs(C - Column) = Abs(R - Row)
```

In cui C ed R indicano la Colonna e la riga in cui è posizionata una regina, mentre Column e Row indicano la colonna e la riga in cui vorremmo posizionare la nuova regina. Se le due celle giacciono sulla stessa diagonale, allora tale condizione è vera, dato che la variazione di ordinata è pari alla variazione d'ascissa.

A questo punto si tratta di scrivere la funzione ricorsiva che genera tutte le soluzioni del nostro problema. Una possibile codifica è mostrata di seguito, in cui tutto è delegato all funzione privata denominata Place_Queens. In pratica tale funzione, oltre a rilasciare in output il numero di soluzioni trovate, provvede a stampare ogni soluzione completa. Usando i commenti e tenendo presente quanto detto precedentemente, non dovrebbe essere difficile comprendere il funzionamento di tale funzione.

```
Public Sub Show_All(Optional Size As Integer = 8)
Dim C_Board As Cls_Board
Set C_Board = New Cls_Board
    C_Board.Size = Size
    Debug.Print Place_Queens(C_Board)
End Sub

Private Function Place_Queens(C_Board As Cls_Board) As Long
'Restituisce il numero totale di soluzioni trovate
Dim Sz As Integer
Dim N_Sol As Long
Dim Column As Integer

    Sz = C_Board.Size
' Caso base (chiusura): se il numero di regine della sotto soluzione è pari al numero di colonne abbiamo finito
' si plotta la soluzione.
If Sz = C_Board.N_Queens Then
    C_Board.Show
    Place_Queens = 1
    Exit Function
End If

    N_Sol = 0

' Proviamo a mettere una regina in una colonna della riga successiva della scacchiera
For Column = 1 To Sz
    If C_Board.IS_Safe(Column) Then
        C_Board.Next_Row (Column)
        N_Sol = N_Sol + Place_Queens(C_Board) ' La chiamata ricorsiva
        C_Board.Remove_Row (Column) ' Backtracking, si torna al punto precedente e proviamo ad andare avanti
    End If
Next Column
Place_Queens = N_Sol
End Function
```