Fortza4 - progettato con elementi della classe Terza

Analisi del progetto

Scopo del gioco

Si intende realizzare un gioco tra due avversari (umani) che dispongono delle pedine colorate (rosso contro verde, nel nostro esempio) al fine di metterne quattro in fila (orizzontale o verticale o diagonale); vince chi raggiunge per primo l'obiettivo. Le pedine vengono immesse dall'alto, scegliendo la colonna in cui farle cadere e si depositano sulle altre pedine della colonna o sul fondo se la colonna è vuota.
Nel nostro esempio la matrice consisterà di 20 colonne e 10 righe, di cui la prima non è utilizzabile e serve esclusivamente per indicare la colonna in cui giocare la pedina.

Dati e codici

I dati più significativi del progetto sono:

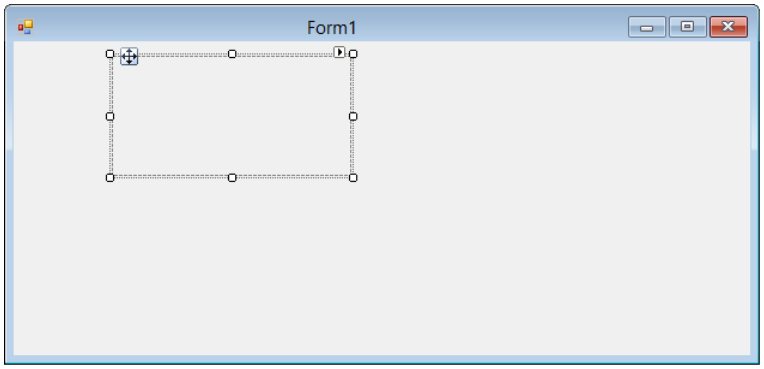
- una **matrice di interi**, di dimensione 10x20; nelle celle si userà la seguente codifica:
  - 0 per indicare una cella vuota
  - –1 per indicare una cella occupata dal verde
  - +1 per indicare una cella occupata dal rosso
- Un controllo **dataGridView**, di dimensione 10x20; nelle celle si userà la seguente codifica:
  - Azzurro per la prima riga, usata per la scelta della colonna di immissione delle pedine
  - Bianco per le celle libere
  - Rosso per le celle occupate da pedine rosse
  - Verde per le celle occupate da pedine verdi
- Le seguenti variabili globali per le informazioni generali di gioco:
  - int **nColonne** = 20; //numero colonne di gioco
  - int **nRighe** = 10; //numero righe di gioco
  - int **nPixel** = 25; //dimensione celle di gioco
  - int **turno**; //a chi tocca? -1=verde, +1=rosso
  - bool **vittoria**; //indica se qualcuno ha vinto
- Due **pannelli**, che sono contenitori di controlli per disegnare una grafica flessibile e introdurre in seguito pulsanti e etichette di informazione
  - Panel 1 - per ospitare pulsanti e altri controlli di gestione utente
  - Panel 2 - per ospitare la griglia dataGridView

Soluzione del progetto

Composizione del Form1

1) Inserire nel form1 un panel1 e impostare le sue proprietà come segue:
- Dock ← Top
- Height ← 35
- BackColor ← LightSteelBlue

2) Impostare anche le proprietà del form1 come segue:
- Text ← Allinea4



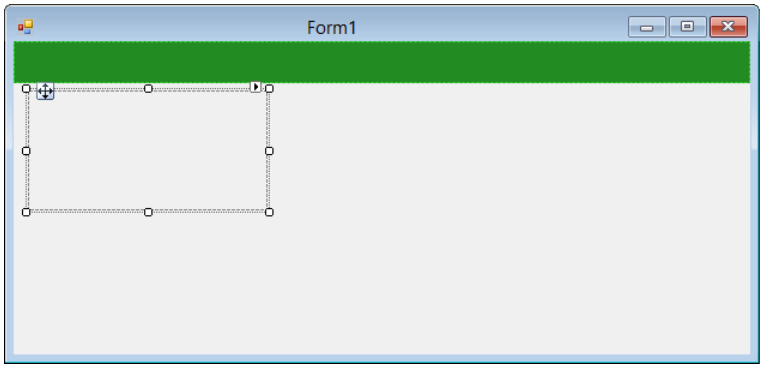
3) Doppio clic sul form1 e scrivere il seguente gestore di evento:

```
private void Form1_Load(object sender, EventArgs e)
{
    Inizio();
}
```

4) Ignorare l'errore che si solleva e proseguire

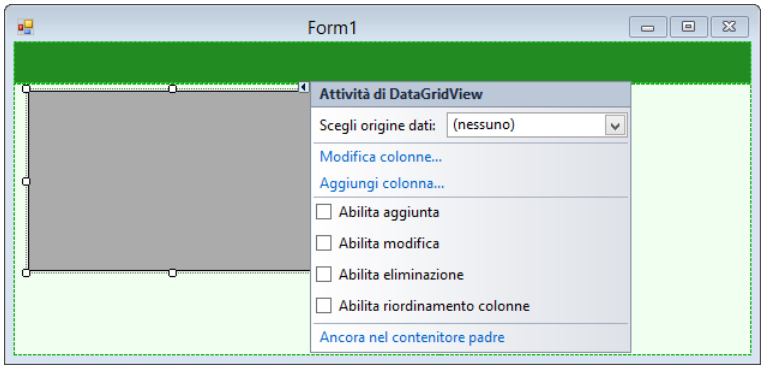
5) Inserire nel form1 un panel2 e impostare le sue proprietà come segue:

- Dock ← Fill
- BackColor ← Honeydew



6) Inserire nel panel2 un dataGridView1 e impostare le sue proprietà come segue:

- Abilita Aggiunta NO
- Abilita Modifica NO
- Abilita Eliminazione NO
- Abilita Riordina NO
- Dock ← Left

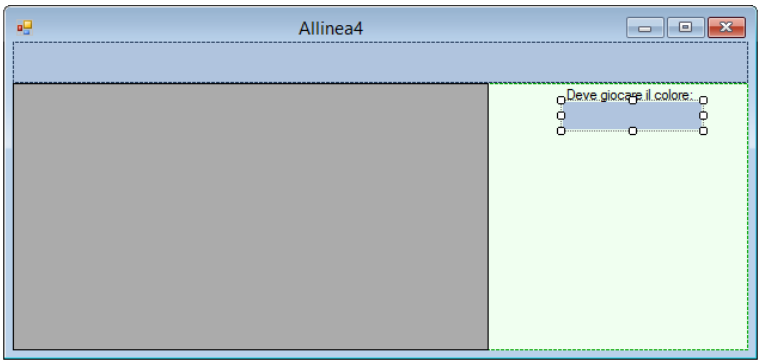


7) Inserire nel panel2 una etichetta con la scritta :

- «deve giocare il colore: »

8) Inserire nel panel2 una etichetta senza scritta ma colorata di :

- BackColor ← LightSteelBlue
- Autosize ← false



9) Visualizzare il codice e inserire un commento (**obbligatorio**) simile al seguente:

```
using System.Windows.Forms;
/* ************************************************************
 * Programma Allinea4 data della consegna 14/02/2014
 * SCUOLA: ITI G.M. ANGIOY SASSARI
 * CLASSE: III A INFORMATICA A.S. 2013/14
 * Studente realizzatore: Guido Piano
 * Docenti della materia: Prof. Andrea Zoccheddu e Prof. Remo Stanco
 ************************************************************ */
```

10) Visualizzare il codice e inserire le seguenti dichiarazioni globali (prima del metodo **`Form1()`**):

```
public partial class Form1 : Form
    {
        //variabili globali ----------------------------------------
        static int nColonne = 20; //numero colonne di gioco
        static int nRighe = 10;   //numero righe di gioco
        static int nPixel = 25;   //dimensione celle di gioco
        static int turno;         //a chi tocca? -1=verde, +1=rosso
        static bool vittoria;     //indica se qualcuno ha vinto
        static int colonna = -1;  //indica la colonna su cui si gioca
        int[,] griglia = new int[nRighe, nColonne];
        //0 libera, 1 rossa, -1 verde ***

        // ------------------------------------------------------
        public Form1()
        {
            InitializeComponent();
        }
```

OSSERVAZIONE: static è una parola chiave del linguaggio che serve per consentire l'accesso alla variabile da qualsiasi punto del programma. Non è necessario in questa fase di approfondire il termine.

11) Sotto la dichiarazione del metodo standard Form1 Inserire il seguente frammento di codice:

```
public Form1()
{                   InitializeComponent();          }
//----------------------------- metodi utente -------------------------
public void Preparazione() //-----------------------------1
{
    turno = -1; //gioca il verde per primo
    PreparaMatrice();
    PreparaDataGrid();
}
public void PreparaMatrice() //-----------------------------2
{
    griglia = new int[nRighe, nColonne];
    for (int r = 0; r < nRighe; r++)
        for (int c = 0; c < nColonne; c++)
            griglia[r, c] = 0; //cella libera
}
```

OSSERVAZIONE: un metodo è un frammento di codice con un nome. Nella scrittura qui sopra si sono dichiarati due metodi denominati rispettivamente **Preparazione()** e **PreparaMatrice()**. Questa tecnica consente di scrivere pezzi di programma e di richiamarli semplicemente scrivendo il loro nome. Per esempio scrivendo **PreparaMatrice()** si esegue tutto il codice contenuto in esso (quello subito sotto).

12) Proseguendo, sotto i due metodi, scrivere il seguente:

```
public void PreparaDataGrid() //-----------------------------3
{
    dataGridView1.ColumnCount = nColonne;
    dataGridView1.RowCount = nRighe;
    dataGridView1.ReadOnly = true;
    dataGridView1.MultiSelect = false;
    dataGridView1.AllowUserToAddRows = false;
    dataGridView1.AllowUserToDeleteRows = false;
    dataGridView1.AllowUserToOrderColumns = false;
    dataGridView1.AllowUserToResizeColumns = false;
    dataGridView1.AllowUserToResizeRows = false;
    dataGridView1.EnableHeadersVisualStyles = false;
    dataGridView1.ColumnHeadersHeightSizeMode
                                = DataGridViewColumnHeadersHeightSizeMode.DisableResizing;
    dataGridView1.RowHeadersWidthSizeMode
                                    = DataGridViewRowHeadersWidthSizeMode.DisableResizing;
    dataGridView1.SelectionMode = DataGridViewSelectionMode.RowHeaderSelect;
    dataGridView1.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;
    dataGridView1.AutoSizeRowsMode = DataGridViewAutoSizeRowsMode.None;
    dataGridView1.Dock = DockStyle.Fill;
    dataGridView1.Dock = DockStyle.Left;
    dataGridView1.ColumnHeadersHeight = nPixel;
    dataGridView1.RowHeadersWidth = nPixel;
    dataGridView1.ScrollBars = ScrollBars.None;
    dataGridView1.ColumnHeadersVisible = false;
    dataGridView1.RowHeadersVisible = false;
    dataGridView1.Width = (nColonne * nPixel) - 75;
    dataGridView1.BackgroundColor = Color.Green;
    Deselezione(); Color coloreTurno = new Color();
    if (turno == -1)
        coloreTurno = Color.Green;
    else
        coloreTurno = Color.Red;
    label2.BackColor = coloreTurno;
    dataGridView1.BackgroundColor = coloreTurno;
    DisegnaGriglia();
    dataGridView1.Enabled = true;
}
```

OSSERVAZIONE: il metodo **PreparaDataGrid()** precedente serve per impostare le proprietà del controllo **dataGridView1** in modo che le celle abbiano una corretta dimensione e che il suo aspetto sia adeguato alla necessità del gioco. Alcune delle proprietà usate sono decisamente poco importanti, ma servono solo per conferire un aspetto gradevole al campo di gioco. Altre sono interessanti: per es. **MultiSelect** indica se sia possibile selezionare contemporaneamente più di una cella (e nel nostro caso lo imponiamo a false); le proprietà **ColumnCount** e **RowCount** determinano rispettivamente il numero di colonne e righe della griglia.

14) Proseguendo, sotto i due metodi, scrivere il seguente:

```
public void Deselezione()
{
    dataGridView1.Rows[0].Cells[0].Selected = true;
    dataGridView1.Rows[0].Cells[0].Selected = false;
}

public void MostraColoreTurno()
{
    Color coloreTurno = new Color();
    if (turno == -1)
        coloreTurno = Color.Red;
    else
        coloreTurno = Color.Green;

    label2.BackColor = coloreTurno;
    dataGridView1.BackgroundColor = coloreTurno;
}

public void DisegnaGriglia()
{
    for (int r = 1; r < nRighe; r++)
        for (int c = 0; c < nColonne; c++)
        switch (griglia[r, c])
        {
            case -1:
                dataGridView1.Rows[r].Cells[c].Style.BackColor
                    = Color.Green;
                break;

            case +1:
                dataGridView1.Rows[r].Cells[c].Style.BackColor
                    = Color.Red;
                break;

            default:
                dataGridView1.Rows[r].Cells[c].Style.BackColor
                    = Color.White;
                break;

        }
    for (int c = 0; c < nColonne; c++)
        dataGridView1.Rows[0].Cells[c].Style.BackColor
                    = Color.PaleTurquoise;

    Deselezione();
}
```

OSSERVAZIONE: il metodo **Deselezione()** serve per evitare che ci sia una cella selezionata. Dovrebbe esistere un metodo migliore ma non l'ho trovato.
Il metodo **MostraColoreTurno()** serve solo per indicare a video quale giocatore debba muovere.
Il metodo **DisegnaGriglia()** serve per colorare le celle del **dataGridView1**. Qui si usa la proprietà **Style** di ogni singola cella della griglia per impostare il colore di sfondo usando il bianco per le celle libere, il verde e il rosso per quelle occupate e il celeste per la prima riga.

15) Proseguendo, scrivere il seguente metodo:

```
public void Mossa()
{
    dataGridView1.Enabled = false;
    int r = 0;
    int c = colonna;
    Deselezione();
    while ((r < nRighe) && (griglia[r, c] == 0)) //-------cella libera
        r++;
    if (r == 0) //colonna piena
    {
        MessageBox.Show("Colonna piena. Cambiala!", "Allinea4");
    }
    else
    {
        griglia[r - 1, c] = turno;
        DisegnaGriglia();
        ControlloVittoria();
        if (vittoria == false)
        {
            MostraColoreTurno();
            turno = -turno;
            dataGridView1.Enabled = true;
        }
    }
}
```

OSSERVAZIONE: il metodo **Mossa()** serve per realizzare la mossa del giocatore. La prima fase cerca la prima cella libera della colonna selezionata. Se non riesce a scendere sotto lo zero significa che la colonna è piena, altrimenti inserisce il numero del turno nella matrice di interi e poi invoca il metodo per disegnare la griglia e infine controlla se qualcuno ha vinto.

16) Proseguendo, scrivere il seguente metodo:

```
public void ControlloVittoria()
{
    vittoria = false;
    ControllaRighe();
    ControllaColonne();
    ControllaDiagonali();
    if (vittoria)
        Vittoria();
}
```

OSSERVAZIONE: il metodo **ControlloVittoria ()** serve verificare se qualcuno ha vinto. Si imposta la variabile vittoria a falsa (si suppone che nessuno abbia vinto) e poi si controllano le quadruplette rispettivamente in orizzontale, verticale e diagonale.
Se la variabile vittoria è cambiata in vera allora significa che qualcuno ha vinto e si invoca il metodo Vittoria che deve comunicare che la partita è finita.

17) Proseguendo, scrivere il seguente metodo:

```
private void ControllaRighe()
{
    for (int r = 1; r < (nRighe); r++)
        for (int c = 0; c < (nColonne - 3); c++)
            if (griglia[r, c] != 0
                && griglia[r, c] == griglia[r, c + 1]
                && griglia[r, c] == griglia[r, c + 2]
                && griglia[r, c] == griglia[r, c + 3])
                vittoria = true;
}
```

OSSERVAZIONE: il metodo **ControllaRighe ()** serve verificare se qualcuno ha vinto con 4 pedine in orizzontale. Se qualcuno ha vinto allora la variabile vittoria è cambiata in vera .

18) Proseguendo, scrivere il seguente metodo:

```
private void ControllaColonne()
{
    for (int r = 1; r < (nRighe - 3); r++)
        for (int c = 0; c < (nColonne); c++)
            if (griglia[r, c] != 0
                && griglia[r, c] == griglia[r+1, c]
                && griglia[r, c] == griglia[r+2, c]
                && griglia[r, c] == griglia[r+3, c])
                vittoria = true;
}
```

OSSERVAZIONE: il metodo **ControllaColonne ()** serve verificare se qualcuno ha vinto con 4 pedine in verticale. Se qualcuno ha vinto allora la variabile vittoria è cambiata in vera .

19) Proseguendo, scrivere il seguente metodo:

```
private void ControllaDiagonali()
{
    for (int r = 1; r < (nRighe - 3); r++)
        for (int c = 0; c < (nColonne - 3); c++)
            if (griglia[r, c] != 0
                && griglia[r, c] == griglia[r + 1, c + 1]
                && griglia[r, c] == griglia[r + 2, c + 2]
                && griglia[r, c] == griglia[r + 3, c + 3])
                vittoria = true;
    for (int r = 1; r < (nRighe - 3); r++)
        for (int c = 3; c < (nColonne); c++)
            if (griglia[r, c] != 0
                && griglia[r, c] == griglia[r + 1, c - 1]
                && griglia[r, c] == griglia[r + 2, c - 2]
                && griglia[r, c] == griglia[r + 3, c - 3])
                vittoria = true;
}
```

OSSERVAZIONE: il metodo **ControllaDiagonali ()** serve verificare se qualcuno ha vinto con 4 pedine in diagonale. Se qualcuno ha vinto allora la variabile vittoria è cambiata in vera .

20) Proseguendo, scrivere il seguente metodo:

```
public void Vittoria()
{
    if (turno == 1)
        MessageBox.Show(
        "Ha vinto il rosso!",
        "Allinea4");
    if (turno == -1)
        MessageBox.Show(
        "Ha vinto il verde!",
        "Allinea4");
    DialogResult risposta = MessageBox.Show(
                            "Vuoi giocare ancora?\nSe scegli No esce dal programma!",
                            "Allinea4", MessageBoxButtons.YesNo);
    if (risposta == System.Windows.Forms.DialogResult.No)
        Close();
    else
        Preparazione();
}
```

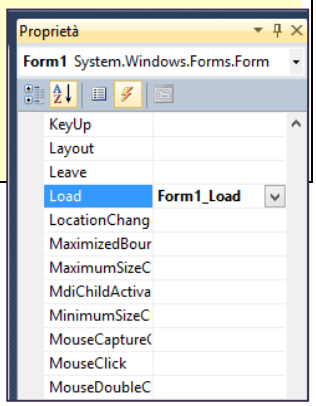
OSSERVAZIONE: il metodo **Vittoria ()** è preposto alla comunicazione della vittoria e alla eventuale preparazione di una nuova partita.

21) Adesso la dichiarazione dei metodi è terminata; non ci resta che collegare le invocazioni dei metodi dai gestori di eventi; i gestori di eventi sono quei particolari metodi che il programma crea automaticamente e collega, per esempio, ai pulsanti in risposta al clic.

22) Il metodo **Form1_Load** è associato al caricamento del **Form1** e serve per compiere alcune istruzioni prima di visualizzare la finestra iniziale. Per creare il gestore di evento è consigliabile visualizzare la

finestra Proprietà, selezionare il Form1, fare clic sul simbolo del fulmine che visualizza gli Eventi e individuare tra i suoi eventi il gestore **Load** e fare doppio clic sullo spazio a fianco.

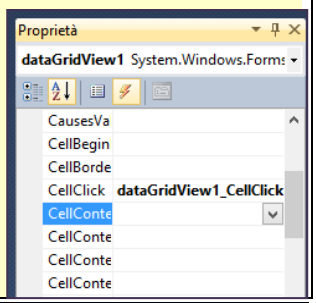
```
//----------------------------gestori di evento -------------------------

private void Form1_Load(object sender, EventArgs e)
{
    Width = 600;
    Height = 300;
    Text = "Allinea4";
    Preparazione();
}
```



OSSERVAZIONE: il metodo **Form1_Load ()** si può generare in due modi diversi: il primo modo è di fare un doppio clic sul Form1 in fase di progettazione; il secondo è di individuare tra i suoi eventi (finestra Proprietà) il gestore e fare doppio clic sullo spazio a fianco. Nella figura qui sotto è proposta una illustrazione della finestra Proprietà, col metodo creato.

23) Il metodo **dataGridView1_CellClick** serve per rispondere al clic sul **dataGridView1** e eseguire le istruzioni corrispondenti. Per creare il gestore di evento è consigliabile visualizzare la finestra Proprietà, selezionare il **dataGridView1**, fare clic sul simbolo del fulmine che visualizza gli Eventi e individuare tra i suoi eventi il gestore **CellClick** e fare doppio clic sullo spazio a fianco.

```
private void dataGridView1_CellClick(object sender, DataGridViewCellEventArgs e)
{
    if (e.RowIndex > 0)
    {
        Deselezione();
        MessageBox.Show ("devi fare clic sulla prima riga
                                          della colonna scelta");
        return;
    }
    colonna = e.ColumnIndex;
    Mossa();
}
```



OSSERVAZIONE: il metodo **dataGridView1_CellClick ()** è il cuore del programma poiché si accorge dei clic dei giocatori, verifica per sicurezza che si sia fatto clic sulla prima colonna (altrimenti si deve ripetere la giocata) e, impostando la colonna scelta e invoca il metodo **Mossa()** che provvede alla mossa del giocatore di turno.

24) Il progetto è pronto per essere avviato ma si lascia allo studente la possibilità di arricchirlo con pulsanti e elementi informativi, le istruzioni del gioco e il salvataggio della partita. Il progetto finale dovrebbe assomigliare alla seguente illustrazione: