



**Eötvös Loránd Tudományegyetem  
Informatikai Kar**

## **Eseményvezérelt alkalmazások**

---

### **8. előadás**

## **Windows Forms dinamikus felhasználói felület, elemi grafika**

---

**Cserép Máté**

**[mcserep@inf.elte.hu](mailto:mcserep@inf.elte.hu)**

**<https://mcserep.web.elte.hu>**

# Windows Forms dinamikus felhasználói felület

## Vezérlők dinamikus kezelése

---

- Vezérlőket *dinamikusan* is létrehozhatunk, az alkalmazás futása közben
  - a kódban létrehozott vezérlők tulajdonságait (pozíció, méret, felirat, ...), valamint az eseménykezelő-társításokat ugyanúgy be tudjuk állítani
  - a vezérlő csak akkor jelenik meg az ablakon, ha annak **Controls** listájában szerepel, ezért oda is fel kell vennünk, illetve törölnünk kell, ha le akarjuk venni az ablakról
    - pl.:  
`this.Controls.Add(myLabel);`
    - esetlegesen manuálisan is megsemmisíthető a vezérlő a **Dispose (...)** művelettel, de ez csak ritkán szükséges

# Windows Forms dinamikus felhasználói felület

## Méretezés, elrendezések

---

- Annak érdekében, hogy a felület alkalmazkodjon az ablak méretéhez, vehetjük méreteit automatikusra (**AutoSize**, **AutoSizeMode**), továbbá lehetőségünk van, hogy különböző módon dokkoljuk őket (**Dock**) a tartalmazó vezérlőhöz
- A csoportosan létrehozott vezérlők elhelyezhető különböző elrendező elemek (pl. **FlowLayoutPanel**, **TableLayoutPanel**) segítségével
  - ekkor a vezérlőt nem az ablak, hanem az elrendező gyerekelemeként helyezzük el
  - Az elrendezők speciális módon szabályozhatóak (pl. **TableLayoutPanel** esetén megadható a sorok, illetve oszlopok méretezésének módja egyenként)

# Windows Forms dinamikus felhasználói felület

## Méretezés, elrendezések

---

- Pl.:

```
this.AutoSize = true; // automatikus méret
this.AutoSizeMode =
    AutoSizeMode.GrowAndShrink; // csökkenhet is
...
Button myButton = new Button();
myButton.Dock = DockStyle.Fill; // kitöltés

FlowLayoutPanel myPanel = new FlowLayoutPanel();
    // folyamatos elrendező elem
myPanel.FlowDirection = FlowDirection.BottomUp;
    // alulról felfele elrendezés
myPanel.Controls.Add(myButton);
    // a gombot az elrendezőre vesszük fel
```

# Windows Forms dinamikus felhasználói felület

## Példa

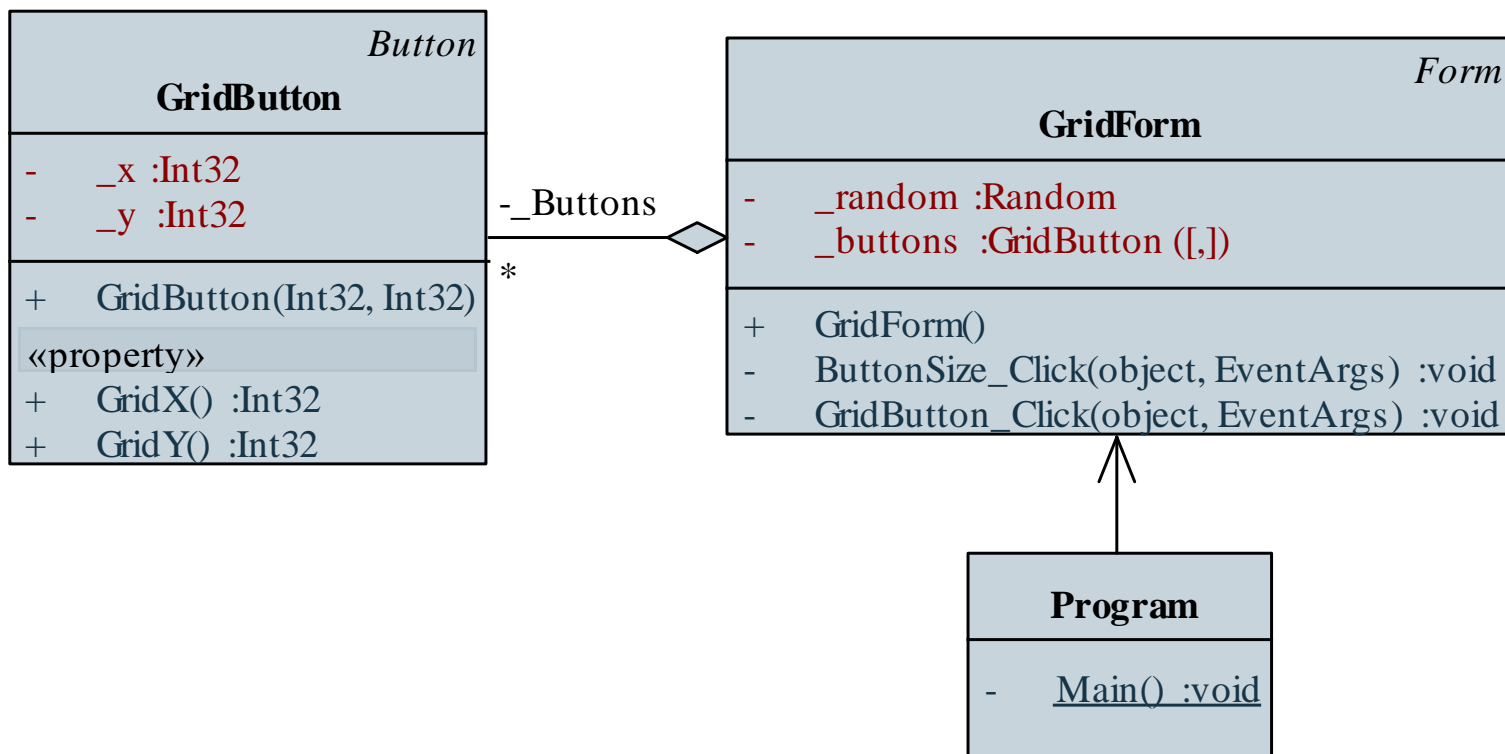
*Feladat:* Készítsünk egy dinamikusan méretezhető táblát, amely véletlenszerű színre állítja a kattintott gombot, valamint a vele egy sorban és oszlopban lévőket.

- táblaelrendezést (**TableLayoutPanel**) használunk, amely tartalmazni fogja a gombrácsot, ügyeljünk arra, hogy a stílusokat is szabályoznunk kell a sorokban és oszlopokban
- a rács méretét külön szabályozhatjuk (**NumericUpDown**), mindig új, üres rácsot generálunk (a régit töröljük)
- a gombokat specializáljuk egy új típusba (**GridButton**), amely eltárolja annak rácsbeli koordinátáját is (**GridX, GridY**)

# Windows Forms dinamikus felhasználói felület

## Példa

Tervezés:



# Windows Forms dinamikus felhasználói felület

## Példa

*Megvalósítás (GridButton.cs):*

```
class GridButton : Button {
    // rácsgomb típusa, speciális gomb
    private Int32 _x;
    private Int32 _y;

    public Int32 GridX { get { return _x; } }
    public Int32 GridY { get { return _y; } }
    // lekérdezzhetjük a rácsbeli pozíciót

    public GridButton(Int32 x, Int32 y) {
        _x = x; _y = y;
    }
}
```

# Windows Forms dinamikus felhasználói felület

## Példa

*Megvalósítás (GridForm.cs):*

```
void ButtonSize_Click(object sender, EventArgs e) {  
    ...  
    _buttons[i, j] = new GridButton(i, j);  
    _buttons[i, j].BackColor = Color.White;  
    _buttons[i, j].Dock = DockStyle.Fill;  
    // kitöltésre állítjuk  
    _buttons[i, j].Click +=  
        new EventHandler(GridButton_Click);  
    // eseménykezelő társítás  
    _tableLayoutGrid.Controls.Add(  
        _buttons[i, j], j, i);  
    // hozzáadjuk a táblapanel vezérlőihez  
    ...  
}
```



# Windows Forms dinamikus felhasználói felület

## Példa

*Feladat:* Készítsünk egy Tic-Tac-Toe programot, amelyben két játékos küzdhet egymás ellen.

- a programban lehetőséget adunk új játék kezdésére, valamint lépésre (felváltva)
- a programban ,X' és ,0' jelekkel ábrázoljuk a két játékost
- a program automatikusan jelez, ha vége a játéknak (előugró üzenetben), majd automatikusan új játékot kezd
- lehetőséget adunk, hogy a felhasználó bármikor új játékot indítson
- az alkalmazás felületét gombok segítségével valósítjuk meg (9 játékgomb, valamint új játék kezdése)

# Windows Forms dinamikus felhasználói felület

## Példa

---

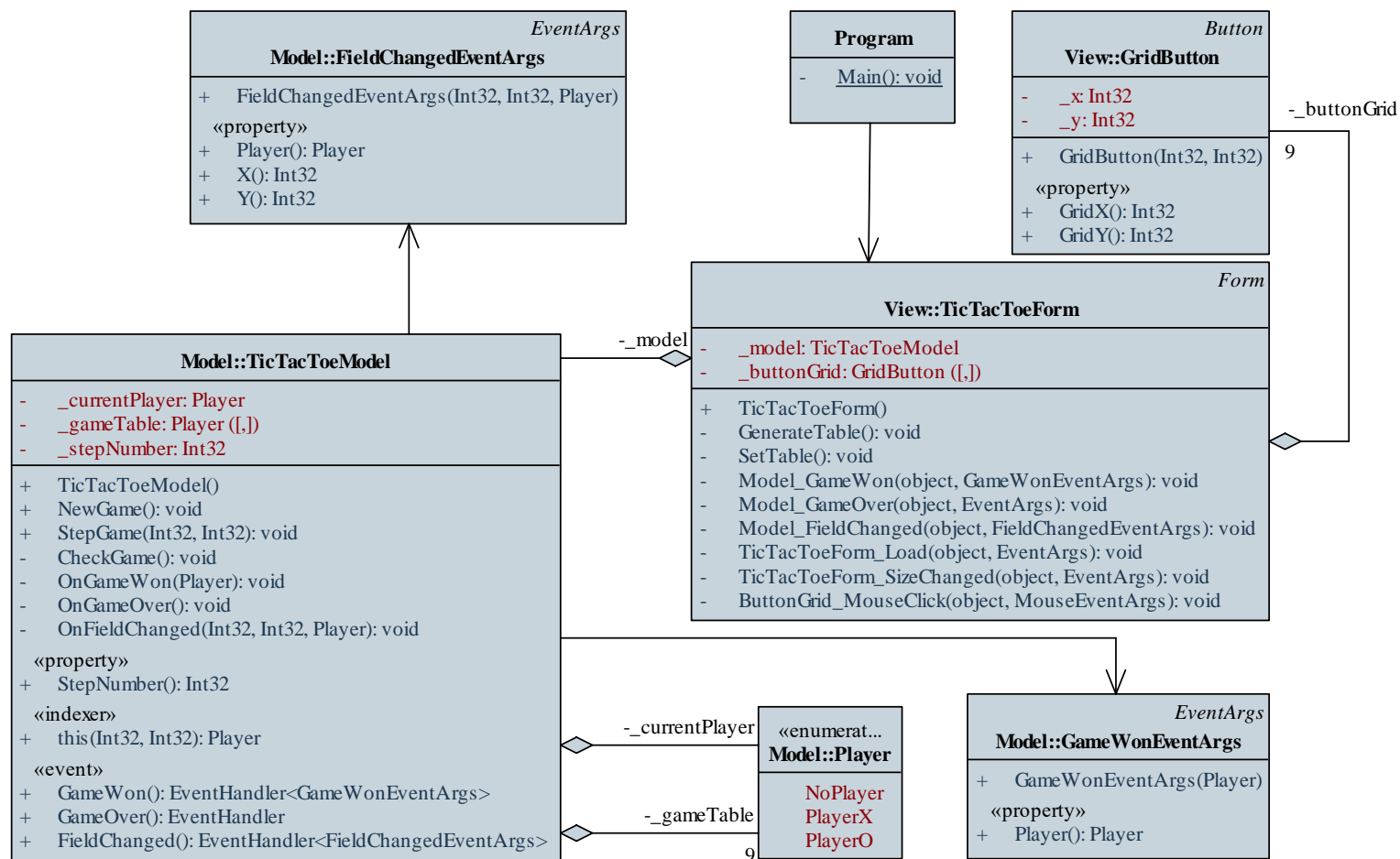
### *Tervezés:*

- az alkalmazást kétrétegű architektúrában valósítjuk meg
- a modell (**TicTacToeModel**) egy mátrixban tárolja el a mezők állásait, a következő játékost és a lépésszámot
- felhasználunk egy felsorolási típust a mezők értékeire (**Player**)
- eseménnyel jelezzük a mező változását, játék végét és a győzelmet, és felhasználunk két speciális eseményargumentum típust (**FieldChangedEventArgs**, **GameWonEventArgs**), amelyek plusz információkat biztosítanak

# Windows Forms dinamikus felhasználói felület

## Példa

Tervezés:



# Windows Forms dinamikus felhasználói felület

## Példa

*Megvalósítás (TicTacToeModel.cs):*

```
public void StepGame(Int32 x, Int32 y) {
    _gameTable[x, y] = _currentPlayer;
    // pozíció rögzítése
    OnFieldChanged(x, y, _currentPlayer);
    // jelezzük egy eseménykiváltással, hogy
    // változott a mező
    _stepNumber++;
    _currentPlayer =
        _currentPlayer == Player.PlayerO ?
        Player.PlayerX : Player.PlayerO;
    // beállítjuk a következő játékost
    CheckGame();
}
```

# Windows Forms dinamikus felhasználói felület

## Példa

*Megvalósítás (TicTacToeModel.cs):*

```
private void OnGameWon(Player player) {
    if (GameWon != null)
        GameWon(this, new GameWonEventArgs(player));
}
private void OnGameOver() {
    if (GameOver != null)
        GameOver(this, EventArgs.Empty);
}
private void OnFieldChanged(Int32 x, Int32 y, ...) {
    if (FieldChanged != null)
        FieldChanged(this,
            new FieldChangedEventArgs(x, y, player));
}
```

# Windows Forms dinamikus felhasználói felület

## Képek megjelenítése

---

- A képek kezelését a **System.Drawing**, illetve **System.Drawing.Imaging** névterek biztosítják
  - támogatott képformátumok: BMP, GIF, JPEG, PNG, TIFF
- Az **Image** osztály az alapvető funkciókat biztosítja, pl.:
  - megnyitás (**Image.FromFile (...)**, **Image.FromStream (...)**), mentés (**Save (...)**),
  - egyszerű manipulációk (**RotateFlip (...)**), miniatűrkép lekérdezés (**GetThumbnailImage (...)**)
  - dimenziók lekérdezése (**Width**, **Height**, **PixelFormat**, **Palette**, ...)

# Windows Forms dinamikus felhasználói felület

## Képek megjelenítése

---

- Ennél bővebb funkcionalitást biztosít a `Bitmap` osztály, pl.
  - pixelszintű lekérdezést, írást (`GetPixel (...)`, `SetPixel (...)`)
  - kép létrehozása méret, fájlnev, illetve másik kép alapján (átméretezéssel is)
- A képek több vezérlőn is megjeleníthetők, pl. egyszerű címkén:

```
Label myLabel = new Label();  
Bitmap myBitmap = new Bitmap(...); // kép betöltése  
myLabel.Size = new Size(myBitmap.Width,  
                        myBitmap.Height);  
    // címke átméretezése  
myLabel.Image = myBitmap; // kép beállítása
```

# Windows Forms dinamikus felhasználói felület

## Képek megjelenítése

---

- Alapvetően a képek megjelenítésére a **PictureBox** vezérlő szolgál, amely számos kényelmi funkciót biztosít, pl.:
  - méretezés módja (**SizeMode**)
  - kép betöltése lokális vagy távoli útvonalról (**ImageLocation**)
  - hibakép (**ErrorImage**)

- Pl.:

```
PictureBox myBox = new PictureBox();
```

```
...
```

```
myBox.Image = myBitmap; // kép beállítása
```

```
myBox.SizeMode = PictureBoxSizeMode.StretchImage;
```

```
    // kép elnyújtása a vezérlő méreteinek
```

```
    // megfelelően
```



# Windows Forms dinamikus felhasználói felület

## Erőforrások kezelése

---

- A programban felhasznált erőforrásokat (képek, hangok, ...) célszerű a projekthez rögzíteni
  - bármilyen fájl hozzáadható a projekthez
    - tartalomként (*content*), és átmásolható a kimentti könyvtárba (az elem tulajdonságait szabályozva; vagy
    - beágyazott erőforrásként (*embedded resource*)
  - az erőforrásfájlok (*resource file*) lehetővé teszik erőforrások (szöveg, kép, ikon) csoportos kezelését és programkódban történő elérését
    - az így hozzáadott erőforrások is beágyazottan elhelyeződnek az alkalmazásban
    - a **MyResource.resx** fájl hozzáadásával a tartalmakat **MyResource.ResourceManager** útvonalon érjük el

# Windows Forms dinamikus felhasználói felület

## Fájlrendszer-kezelés

---

- A fájlokkal és fájlrendszerrel kapcsolatos műveletek a **System.IO** névtérben helyezkednek el
  - fájlműveleteket a **File**, könyvtárműveleteket a **Directory** osztály statikus műveleteivel hajthatunk végre, pl.:

```
Directory.CreateDirectory(@"c:\Data");  
    // könyvtár létrehozása  
String[] paths = Directory.GetFiles(@"c:\Data");  
    // könyvtár listázása  
File.Copy(@"c:\data.txt", @"c:\Data\data.txt");  
    // fájl másolása
```
  - az elérési útvonallal kapcsolatos műveletek a **Path** osztályban találhatóak, pl.:

```
Path.GetParent(@"c:\Data"); // szülő lekérdezése
```

# Windows Forms dinamikus felhasználói felület

## Példa

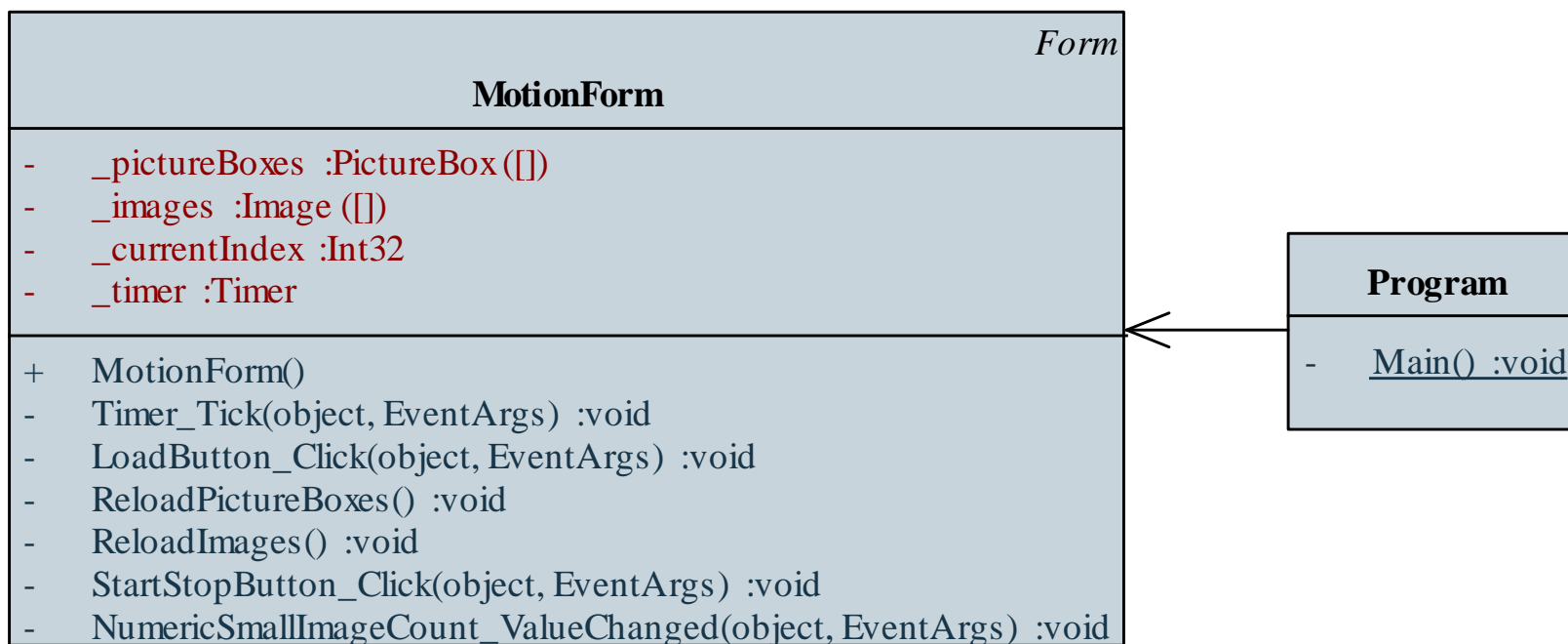
*Feladat:* Készítsünk egy mozgókép megjelenítő alkalmazást, amelyben képek sorozatát tudjuk betölteni (mint képkockákat), és megjeleníteni azt animációként. Lehesen szabályozni az animáció sebességét, valamint lehesen látni, hogy a következő 1 másodpercben milyen képkockák jelennek meg.

- a felületnek lesz statikus, valamint dinamikus része (egy másodpercnek megfelelő képek)
- a képek megnyitásához könyvtárböngésző dialógust (**FolderBrowserDialog**) használunk
- eltároljuk a betöltött képeket (**\_images**), valamint a generált címkéket (**\_pictureBoxes**), és időzítő segítségével fogjuk periodikusan cserélni őket

# Windows Forms dinamikus felhasználói felület

## Példa

*Tervezés:*



# Windows Forms dinamikus felhasználói felület

## Példa

*Megvalósítás (MotionForm.cs):*

```
void LoadButton_Click(object sender, EventArgs e) {
    if (_folderBrowserDialog.ShowDialog() ==
        DialogResult.OK) {
        // ha OK-val zárták le a dialógusablakot
        String[] files = Directory.GetFiles(
            _folderBrowserDialog.SelectedPath,
            "*.jpg");
        // könyvtár jpg kiterjesztésű fájljainak
        // listázása

        _images = new Image[files.Length];
        // a képek száma megegyezik a fájlok
        // számával
    }
}
```

# Windows Forms dinamikus felhasználói felület

## Példa

*Megvalósítás (MotionForm.cs):*

```
    for (Int32 i = 0; i < files.Length; i++) {
        try
        {
            _images[i] = Image.FromFile(files[i]);
            // kép betöltése
        }
        catch (ArgumentException) {
            // ha a fájl nem kép
            _images[i] = null;
        }
    }
    ...
}
```

# Windows Forms dinamikus felhasználói felület

## Példa

*Megvalósítás (MotionForm.cs):*

```
private void ReloadPictureBoxes() {  
    // kis képeket tartalmazó képmegjelenítők  
    // cseréje  
    _pictureBoxes[i] = new PictureBox();  
    ...  
    _pictureBoxes[i].BorderStyle =  
        BorderStyle.FixedSingle; // keret  
    _pictureBoxes[i].SizeMode =  
        PictureBoxSizeMode.StretchImage; // nyújtás  
  
    Controls.Add(_pictureBoxes[i]);  
    // vezérlő felvétele  
    ...  
}
```

# Windows Forms elemi grafika

## Rajzolási lehetőségek

---

- A grafikus felület lehetőséget biztosít 2D rajzolás végrehajtására, amelynek keretében egyszerű alakzatokat (vonal, kör, szöveg, ...), vagy képeket rajzolhatunk bármely felületre az ablakunkban
- A rajzolással kapcsolatos tevékenységek a **System.Drawing** névtérben találhatóak
  - a rajzolást a **Graphics** osztály metódusai biztosítják
  - minden vezérlő (**Control**), valamint kép (**Image**) rajzolható
  - a rajzolás az adott vezérlő koordinátarendezésben történik logikai koordináták szerint (élsimítással korrigálható)
  - a rajzolásnál műveletenként adjuk meg a tulajdonságokat



# Windows Forms elemi grafika

## Rajzeszközök

---

- A **Graphics** osztály példányosításával megadjuk a rajzfelületet, majd a következő módon rajzolhatunk:
  - a **DrawLine**, **DrawRectangle**, **DrawArc**, ... műveletek az alakzatok körvonalát rajzolják meg toll (**Pen**) segítségével
  - a **FillRectangle**, **FillEllipse**, **FillPath**, ... műveletek az alakzatok kitöltését rajzolják meg ecset (**Brush**) segítségével
  - a **DrawString** művelettel rajzolhatunk szöveget a megadott betűtípussal (**Font**) és tollal
  - a **DrawImage** művelettel rajzolhatunk képet
  - a **Clear** művelettel törölhetjük a rajzfelületet

# Windows Forms elemi grafika

## Rajzfelület

---

- A rajzoló felület lehet:
  - a direkt rajzolásra készített **Panel** típus
    - rendelkezik egy **Paint** eseménnyel, amelynek eseményargumentumából lekérdezhető a rajzobjektum
    - a panel frissítésével (**Refresh (...)**) újra kiváltódik az esemény
    - ugyanígy lekérhető az objektum a **CreateGraphics ()** utasítással is
  - bármely egyéb vezérlő a **Graphics.FromHwnd (...)** utasítással, amely paraméterben egy **Control** objektum **Handle** tulajdonságát kapja meg, pl.:  
**Graphics g = Graphics.FromHwnd(myButton.Handle) ;**

# Windows Forms elemi grafika

## Rajzfelület

- kép (**Image**), így lehetőségünk van a rajzolás háttérben való elvégzésére és kimentésére, ehhez a **Graphics.FromImage (...)** műveletet kell használnunk
- Pl.:

```
Panel myPanel = new Panel(); // rajzpanel
...
myPanel.Paint +=
    new PaintEventHandler(Panel_Paint);
...
void Panel_Paint(object sender, PaintEventArgs e) {
    Graphics gr = e.Graphics;
    // vagy myPanel.CreateGraphics();
    ...
}
```

# Windows Forms elemi grafika

## Színek, ecsetek, tollak

---

- A színezést a **Color** típus biztosítja alapértelmezett értékekkel (pl. **Color.Blue**), illetve tetszőleges, akár áttetsző szín létrehozásával (**Color.FromArgb(...)**)
  - a **SystemColors** típus tartalmazza a rendszerszíneket
- A toll (**Pen**) a színen definiál vastagságot, stílust (pl. szaggatott, pöttyözött), valamint végpont típust (pl. lekerekített, nyíl)
  - a **Pens** osztály tartalmazza az egyszerű tollakat
- Az ecset (**Brush**) a szín mellett speciális átmenettel, textúrával tudja ellátni a felületet, így különböző ecsettípusokat használhatunk (**SolidColorBrush**, **TextureBrush**, ...)
  - a **Brushes** osztály tartalmazza az egyszerű kitöltéseket

# Windows Forms elemi grafika

## Színek, ecsetek, tollak

---

- Pl.:

```
gr.FillRectangle(Brushes.Yellow, 0, 0, 200, 100);  
    // narancs színű téglalap kitöltés  
Pen myPen = new Pen(Color.Red, 2);  
    // 2 vastag piros toll  
myPen.DashStyle = DashStyle.Dot; // pontozott  
gr.DrawRectangle(myPen, 0, 0, 200, 100);  
    // szegély megrajzolása  
  
Brush myBrush = new LinearGradientBrush(  
    new Point(0, 0), new Point(100, 100),  
    Color.LightBlue, Color.LightRed);  
    // átmenetes ecset  
gr.FillPolygon(myBrush, ...); // sokszög kitöltés
```

# Windows Forms elemi grafika

## Rajzeszköz beállítások

---

- A **Graphics** osztály további beállításokat biztosít:
  - élsimítás a **SmoothingMode** tulajdonsággal (**Default**, **HighSpeed**, **AntiAlias**, ...)
  - koordinátarendszer módosítás a **TranslateTransform(...)**, **ScaleTransform(...)**, **RotateTransform(...)** műveletekkel (az összes utána lévő utasításra hat)
  - állapotkezelés és váltás a **Save(...)** és **Restore(...)** műveletekkel, így visszakaphatjuk a korábbi koordinátarendszer beállításokat
  - szövegkiterjedés mérése a **MeasureString(...)** művelettel
  - rajzfelület vágása a **SetClip(...)**, ... műveletekkel

# Windows Forms elemi grafika

## Egérkezelés

---

- Az alapvető egérekattintás (**Click**) mellett számos, az egérrel kapcsolatos eseményt tudunk kezelni az alkalmazásban, pl.:
  - egérgomb lenyomása (**MouseDown**), felengedése (**MouseUp**), amely során lekérdezhetjük a gombot (**Button**), valamint az aktuális egérpozíciót (**X, Y**)
  - görgőmozgás (**MouseWheel**), amely során lekérdezhetjük a mozgás mértékét (**Delta**)
  - egér mozgása (**MouseMove**), amely során lekérhetjük az esetlegesen lenyomott gombot, és az egérpozíciót
  - adott vezérlőn történő megjelenése (**MouseEnter**), mozgása (**MouseHover**) és eltávolodása (**MouseLeave**)

# Windows Forms elemi grafika

## Példa

*Feladat:* Készítsünk egy egyszerű rajzolóprogramot, amellyel alapvető alakzatokat tudunk egy felületre rajzolni.

- az alakzatok rögzítettek: zöld téglalap, piros ellipszis, egyenlő szárú sárga háromszög, az alakzat típusát rádiógombokkal tudjuk kiválasztani
- lehetőségünk lesz a rajz törlésére (gomb, vagy *Delete* billentyű segítségével), betöltésére és mentésére
- a rajzolás a bal egérgomb lenyomására történik, ekkor kék kerettel jelöljük az alakzatot, majd felengedéssel el is helyezzük azt a vásznon
- az alkalmazást modell/nézet architektúrában készítjük el



# Windows Forms elemi grafika

## Példa

---

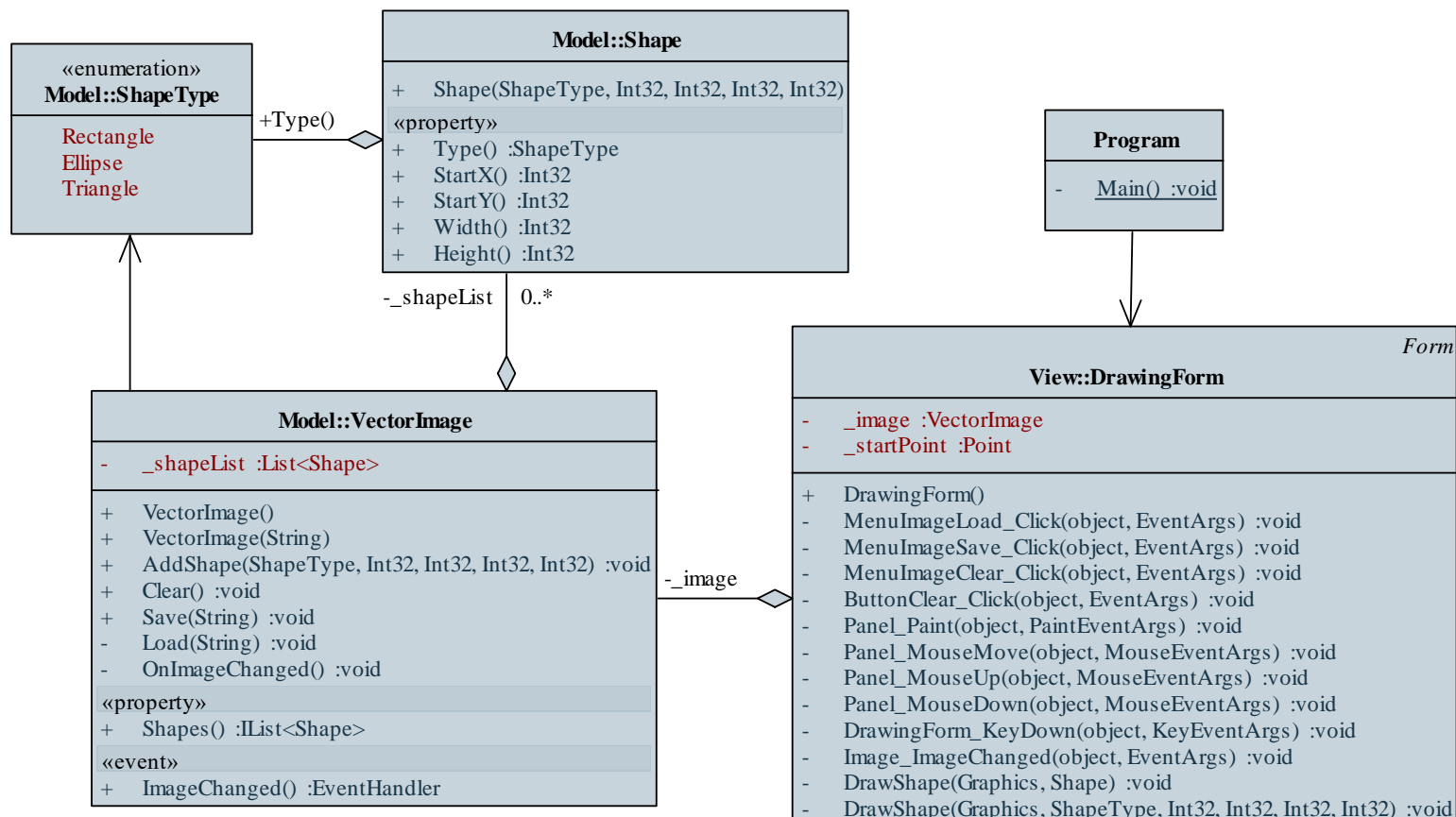
*Tervezés:*

- minden alakzat leírható befoglaló téglalap segítségével, ezért egy típusban (**Shape**) modellezzük őket megadva az alakzattípust (**ShapeType**)
- az alakzatokat egy képbe helyezzük (**VectorImage**), amely lehetőséget ad a szöveges fájlból történő betöltésre, mentésre, hozzáadásra és törlésre, illetve eseménnyel (**ImageChanged**) jelzi, ha változott a kép
- a nézetben (**DrawingForm**) feldolgozzuk a panel egér eseményeit, valamint az ablak billentyűzet eseményét, minden változáskor frissítjük a panelt, és újrarajzoljuk az elemeket (**Panel\_Paint**)

# Windows Forms elemi grafika

## Példa

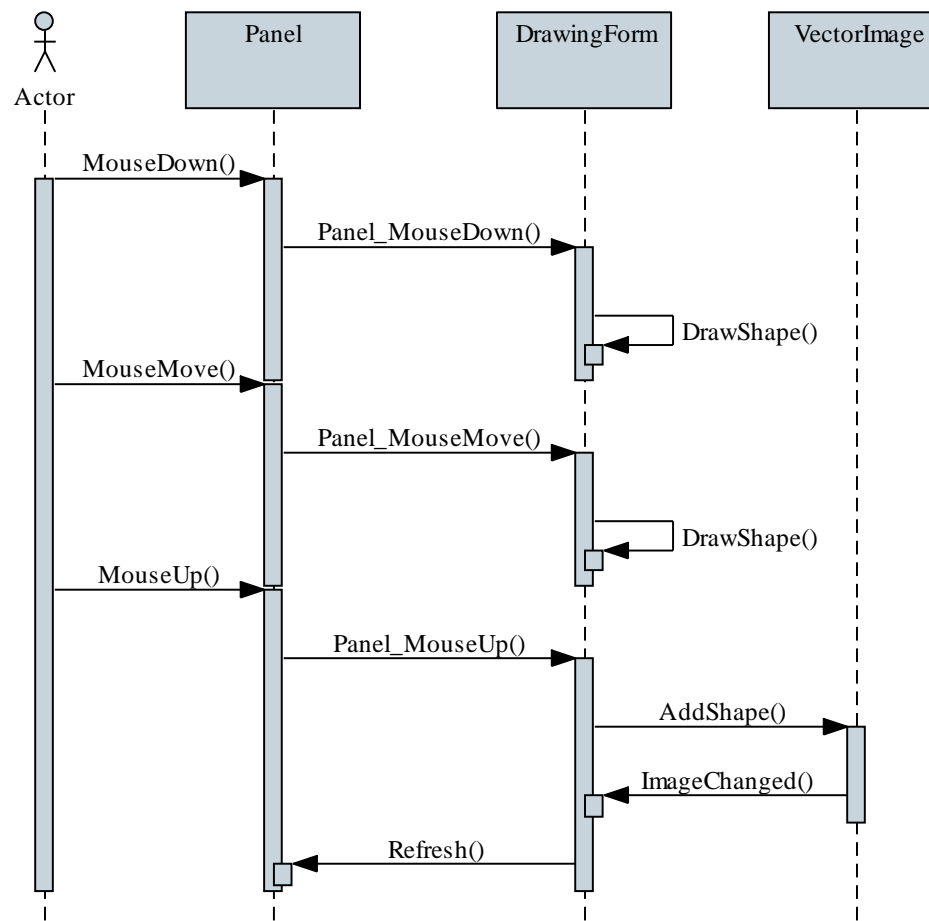
Tervezés:



# Windows Forms elemi grafika

## Példa

*Tervezés:*



# Windows Forms elemi grafika

## Példa

*Megvalósítás (DrawingForm.cs):*

```
private void Panel_Paint(object sender,
                          PaintEventArgs e) {
    Graphics graphics = e.Graphics;
    // rajzeszköz az eseményargumentumból

    foreach (Shape shape in _image.Shapes)
        DrawShape(graphics, shape);
    // alakzatok kirajzolása
}
...
private void Image_ImageChanged(...) {
    _panel.Refresh();
}
```

# Windows Forms elemi grafika

## Példa

*Megvalósítás (DrawingForm.cs):*

```
private void DrawShape(Graphics graphics,
                        Shape shape) {
    switch (shape.Type) {
        case ShapeType.Rectangle:
            graphics.FillRectangle(
                Brushes.LightGreen,
                shape.StartX, shape.StartY,
                shape.Width, shape.Height);
            // kitöltés
            graphics.DrawRectangle(Pens.Green, ...);
            // keret
            break;
        ...
    }
}
```

# Windows Forms elemi grafika

## Példa

*Feladat:* Módosítsuk az előző programot úgy, hogy ne villogjon a képernyő sok alakzat esetén sem.

- a megoldás, hogy nem közvetlenül a képernyőre rajzolunk, hanem egy, a memóriában lévő képre (**Bitmap**)
- a képet kezdetben olyan színűre színezzük, mint a vezérlő (**SystemColors.Control**)
- minden alakzatot a képre rajzolunk, majd a képet egy lépésben kirajzoljuk a képernyőre (**DrawImage**), így az csak egyszer frissül
- mozgatás közben nem használjuk a frissítést, csupán egy lépésben kirajzoljuk a képet

# Windows Forms elemi grafika

## Példa

*Megvalósítás (DrawingForm.cs):*

```
private void Panel_Paint(...) {  
    Bitmap bitmap = new Bitmap(_panel.Width,  
        _panel.Height); // kép létrehozása  
    Graphics graphics = Graphics.FromImage(bitmap);  
    // rajzeszköz a képre  
    graphics.Clear(SystemColors.Control);  
    // a vezérlő színére festjük a képet  
    foreach (Shape shape in _image.Shapes)  
        DrawShape(graphics, shape);  
    // alakzatok kirajzolása  
    e.Graphics.DrawImage(bitmap, 0, 0);  
    // kép kirajzolása a panelre  
}
```

# Windows Forms elemi grafika

## Példa

*Feladat:* Készítsünk egy Tic-Tac-Toe programot, amelyben két játékos küzdhet egymás ellen.

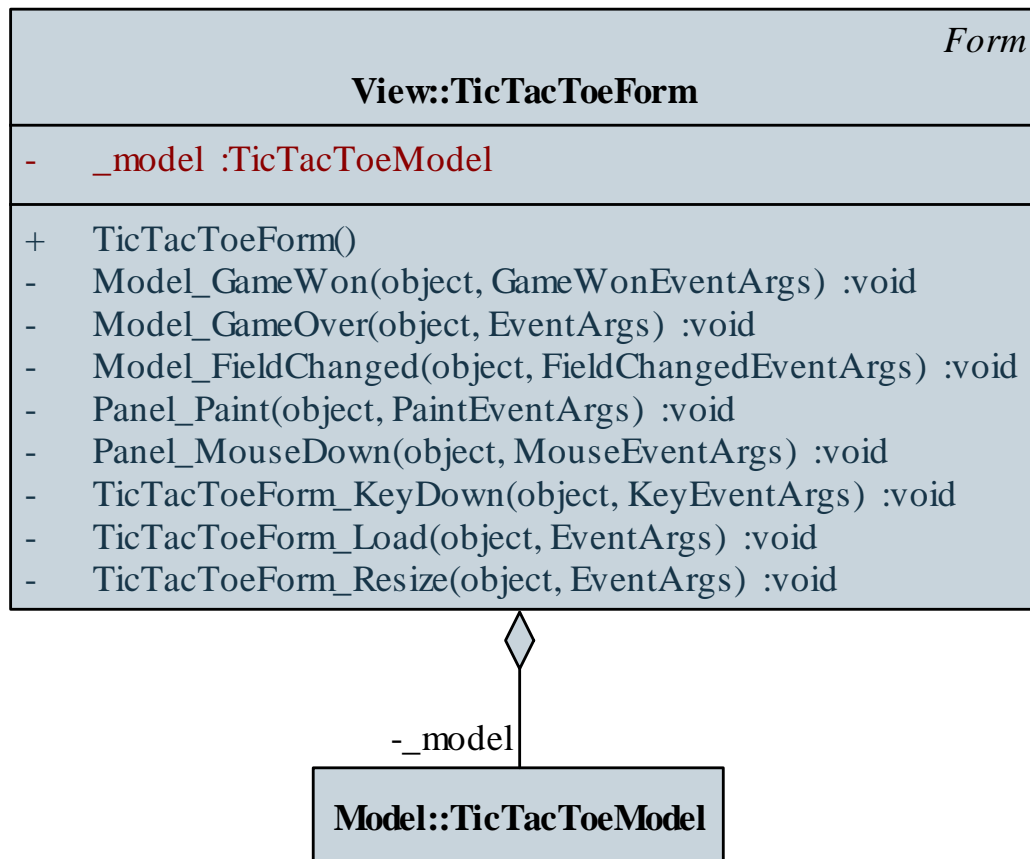
- a programban lehetőséget adunk új játék kezdésére, valamint lépésre (felváltva)
- a programban ,X' és ,0' jelekkel ábrázoljuk a két játékost
- a program automatikusan jelez, ha vége a játéknak (előugró üzenetben), majd automatikusan új játékot kezd
- lehetőséget adunk, hogy a felhasználó bármikor új játékot indítson (**Ctrl+N** billentyűzetkombinációra)
- az alkalmazás felületét elemi grafika segítségével valósítjuk meg



# Windows Forms elemi grafika

## Példa

*Tervezés:*



# Windows Forms elemi grafika

## Példa

*Megvalósítás (TicTacToeForm.cs):*

```
private void Panel_MouseDown(object sender,
                               MouseEventArgs e)
{
    // megállapítjuk, melyik mezőn van az egér
    Int32 x = 3 * e.X / _panel.Width;
    Int32 y = 3 * e.Y / _panel.Height;

    try {
        _model.StepGame(x, y); // lépünk a játékban
    }
    catch { }
}
```

# Windows Forms elemi grafika

## Példa

*Megvalósítás (TicTacToeForm.cs):*

```
private void TicTacToeForm_KeyDown(object sender,
                                   KeyEventArgs e)
{
    if (e.KeyCode == Keys.N &&
        e.Modifiers == Keys.Control)
    {
        // Ctrl+N esetén új játék indítása
        _model.NewGame();
        _panel.Refresh();
    }
}
```