Eötvös Loránd Tudományegyetem
Informatikai Kar

Eseményvezérelt alkalmazások

---

10. előadás

Multi-platform Avalonia UI alkalmazások

Dr. Cserép Máté
mcserep@inf.elte.hu
https://mcserep.web.elte.hu

Avalonia UI alapismeretek

Alkalmazások mobil környezetben

- A mobil/táblagépes környezetben alkalmazásunknak számos speciális követelményt kell teljesíteni
  - *könnyű áttekinthetőség*, infografikus megjelenítés
  - *folyamatos, gyors működés* aszinkron tevékenységekkel
  - *alkalmazkodás* (*resposiveness*): az alkalmazás
    - különböző hardvereken,
    - különböző méretű/felbontású képernyőkön,
    - különböző tájolással (portré/tájkép),
    - különböző üzemmódokban (teljes képernyő, oldalra zárt, stb.) futhat
  - *kézmozdulatok* kezelése, és kihasználása
  - *speciális hardverek* igénybevétele (GPS, gyorsulásmérő, stb.)

Avalonia UI alapismeretek

Kézmozdulatok kezelése

- Célszerű az alkalmazás vezérlésében a felhasználó kézmozdulataira támaszkodni
  - bármely vezérlőre állíthatunk kézmozdulat érzékelést (**`GestureRecognizer`**), így tetszőleges tevékenységet (**`Command`**) rendelhetünk bármely vezérlőhöz
  - támogatott a görgetés (**`ScrollGestureRecognizer`**), a csíptetés (**`PinchGestureRecognizer`**) és a húzás (**`PullGestureRecognizer`**), illetve tetszőleges egyedi mozdulat megvalósítása (**`GestureRecognizer`**), pl.:

```
<Image …>
    <Image.GestureRecognizers> <!-- érzékelés -->
        <PullGestureRecognizer Command=… />
        <!-- érintés hatására fut a parancs -->
```

Avalonia UI alapismeretek

Méret kezelés

- A mobil eszközök képernyőmérete jelentősen eltérhet, és az alkalmazásunknak alkalmazkodnia kell a teljes képernyős megjelenítéshez
  - használjunk relatív elrendezőket és pozícionálást
  - kezelhető a vezérlő átméretezése (**`OnSizeChanged`** felüldefiniálásával a nézet háttérkódjában)
  - pl.:

```
protected override void OnSizeChanged(…) {
    base.OnSizeChanged(e);

    Double height = e.NewSize.Height;
    Double width = e.NewSize.Width;
    // ...
}
```

Avalonia UI alapismeretek

Tájolás kezelés

- A mobil eszközök többféle tájolásban helyezkedhetnek el
  - általában az portré/tájkép (álló/fekvő) tájolásokat különböztetjük meg, de ezeknek lehetnek speciális esetei
  - a támogatott tájolásokat mobil platformokon korlátozhatjuk is
    - Android esetén a főtevékenység (**`MainActivity`**) egyik jellemzője a támogatott tájolás (**`ScreenOrientation`**)
  - az eszköz tájolását és az alkalmazás képernyőjének méretét egyszerre célszerű szabályozni a vezérlő méretváltoztatásának kezelésével (**`OnSizeChanged`**)

Avalonia UI alapismeretek

Tájolás kezelés

- pl.:

```
protected override void OnSizeChanged(
    SizeChangedEventArgs e)
    // megkapjuk az aktuális
    // szélességet/magasságot az argumentumban
{
    base.OnSizeChanged(e);

    // orientáció meghatározása
    if (e.NewSize.Width > e.NewSize.Height)
        … // tájkép
    else
        … // portré
}
```

Avalonia UI alapismeretek

Eszközkezelés

- Az alkalmazások felhasználhatóak táblagépes, illetve mobil környezetben is, és mindkét környezetben megfelelő megjelenítést kell biztosítanunk
  - kódból lekérhetjük az eszköz típusát (**`OperatingSystem`**), és arra megfelelően reagálhatunk, pl.:

```
if (OperatingSystem.IsAndroid() ||
    OperatingSystem.IsIOS()) {
   image.Source =
      ImageSource.FromFile("small.jpg");
} else {
   image.Source =
      ImageSource.FromFile("large.jpg");
}
// táblagépen nagyobb képet használunk
```

Avalonia UI alapismeretek

Eszközkezelés

- nézetből az eszköznek megfelelő értékeket adhatjuk át (**`OnPlatform`**), az alapértelmezett és a specifikus értékek megadásával, pl.:

```
<Label FontSize="{OnPlatform 12, Windows=24}" />
```

- a vezérlők bármely tulajdonsága, akár a tartalmuk is testreszabható ilyen módon pl.:

```
<StackPanel>
    <OnPlatform>
        <OnPlatform.Default>
            ...
        </OnPlatform.Default>
        <OnPlatform.Windows>
            ...
        </OnPlatform.Windows>
    </OnPlatform>
</StackPanel>
```

Avalonia UI alapismeretek

Témák és stílusok

- Az Avalonia UI alkalmazásunk alapvető megjelenítését a beállított téma határozza meg.
  - Jellemzően az alkalmazás szintjén kerül beállításra, pl.:

```
<Application.Styles>
    <FluentTheme />
</Application.Styles>
```

- Két beépített téma (**`SimpleTheme`**, **`FleuntTheme`**) és több NuGet csomagból elérhető is rendelkezésünkre áll.
  - **`SimpleTheme`**: egyszerű, minimalista téma
  - **`FluentTheme`**: a Microsoft Fluent Design alkalmazása
  - **`Material.Avalonia`**: a Google Material Design alkalmazás
  - **`Classic.Avalonia`**: Windows 9x-re hasonlító retró téma

Avalonia UI alapismeretek

Témák és stílusok

- Avalonia UI alkalmazás XAML kódjában a stílusok vezérlőkhöz csatolásakor lehtőségünk van CSS-szerű *szelektorok* használatára.
  - Adott típusú vezérlőkre illesztés:
    `<Style Selector="Button">`
  - Név alapján konkrét vezérlőre illesztés:
    `<Style Selector="#MyButton">`
  - Adott osztállyal (**`large`**) rendelkező vezérlőkre illesztés:
    `<Style Selector="Button.large">`
  - Adott hierarchiának megfelelő vezérlőkre illesztés:
    `<Style Selector="StackPanel Button">`
  - Adott tulajdonságokkal rendelkező vezérlőkre illesztés:
    `<Style Selector="Button[(Grid.Row)=0]">`

Avalonia UI alapismeretek

Témák és stílusok

- Az osztályokra adatkötés (Binding) is lehetséges, logikai értékek formájában, pl.:

```
<Button ...
        Classes="Large" <!-- nem kondícionális -->
        Classes.IsActive="{Binding IsButtonActive}">
        <!- kondícionális osztály,
            adatkötéssel a nézetmodellre -->
  <Button.Styles>
    <Style Selector="Button.Large">...</Style>
    <Style Selector="Button.IsActive">...</Style>
  </Button.Styles>
</Button>
```

- A stílusok **`UserControl`** szintjén is megadhatóak tranzitívan vagy **`ResourceDictionary`**-ben (hasonlóan WPF-hez).

Avalonia UI alapismeretek

Példa

Feladat: Készítsünk egy egyszerű számológépet, amellyel a négy alapműveletet végezhetjük el, illetve láthatjuk korábbi műveleteinket is.

- alakítsuk át a nézetet, hogy könnyen használható legyen mobil környezetben, és alkalmazkodjon a platformhoz
- szövegbevitel helyett gombokat használunk a számokhoz, ezért ki kell egészítenünk a nézetmodellt a számok kezelésével (**`Calculate`**)
- a nézetmodellek ősosztályát (**`ViewModelBase`**) egészítsük ki a tájolás kezelésével (**`IsPortrait`**, **`IsLandscape`**), így szelektor osztályok segítségével tudunk reagálni az elforgatásokra
  - ehhez a nézet **`OnSizeChange()`** műveletét felül kell definiálnunk

Összetett Avalonia UI alkalmazások

Időzítés

- Felületi időzítésre használhatjuk a platformfüggő időzítőt (**`DispatcherTimer`**), amely az Avalonia saját típusa, nem egyezik meg a WPF azonos nevű időzítőjével
  - megadhatjuk az időintervallumot (**`Interval`**), és az időzítésre (**`Tick`**) elvégzendő tevékenységet.
- Felületfüggetlen időzítésre továbbra is használható a **`System.Timers.Timer`** típus
  - az időzítő egy háttérszálon váltja ki az eseményt, az Avalonia UI keretrendszer esetében is igaz, hogy a felületi vezérlőkhöz csak az őket létrehozó szálról férhetünk hozzá
  - használjuk a **`Dispatcher.UIThread.InvokeAsync()`** hívást a szálak közötti szinkronizáláshoz

Összetett Avalonia UI alkalmazások

Példa

Feladat: Készítsünk Avalonia UI felülettel egy vizsgatétel generáló alkalmazást, amely ügyel arra, hogy a vizsgázók közül ketten ne kapják ugyanazt a tételt.

- a modell (**`ExamGeneratorModel`**) valósítja meg a generálást, tétel elfogadást/eldobást, valamint a történet tárolását, a modellre egy interfészen keresztül (**`IExamGenerator`**) hivatkozunk
- két nézetet hozunk létre, egyik a generáló nézet (**`MainView`**), a másik a beállítási nézet (**`SettingsView`**), ezeket ablakokként is tegyük elérhetővé (**`MainWindow`**, **`SettingsWindow`**)
- a két nézetet ugyanaz a nézetmodell (**`ExamGeneratorViewModel`**) szolgálja ki, amelybe befecskendezzük a modellt

Összetett Avalonia UI alkalmazások

Példa

- a nézetmodell tárolja a start/stop funkcióért, beállítások megnyitásáért és bezárásáért felelős utasításokat
- a nézetmodell kezeli a modell eseményét (**`NumberGenerated`**), és frissíti a megjelenített számot
- a nézetmodell egy listában tárolja a kihúzott tételeket (**`History`**), ehhez létrehozunk egy segédtípus (**`HistoryItem`**), amely tárolja az elem sorszámát, illetve az állapotát (kiadható, vagy sem), ezeket a tulajdonságokat kötjük a nézetre
- az alkalmazás (**`App`**) felel az egyes rétegek példányosításáért, valamint a nézetmodell események kezeléséért

- Megfigyelhetjük, hogy a modell és nézetmodell rétegek a feladat korábbi változataival megegyeznek.

Összetett Avalonia UI alkalmazások

Alkalmazások környezete

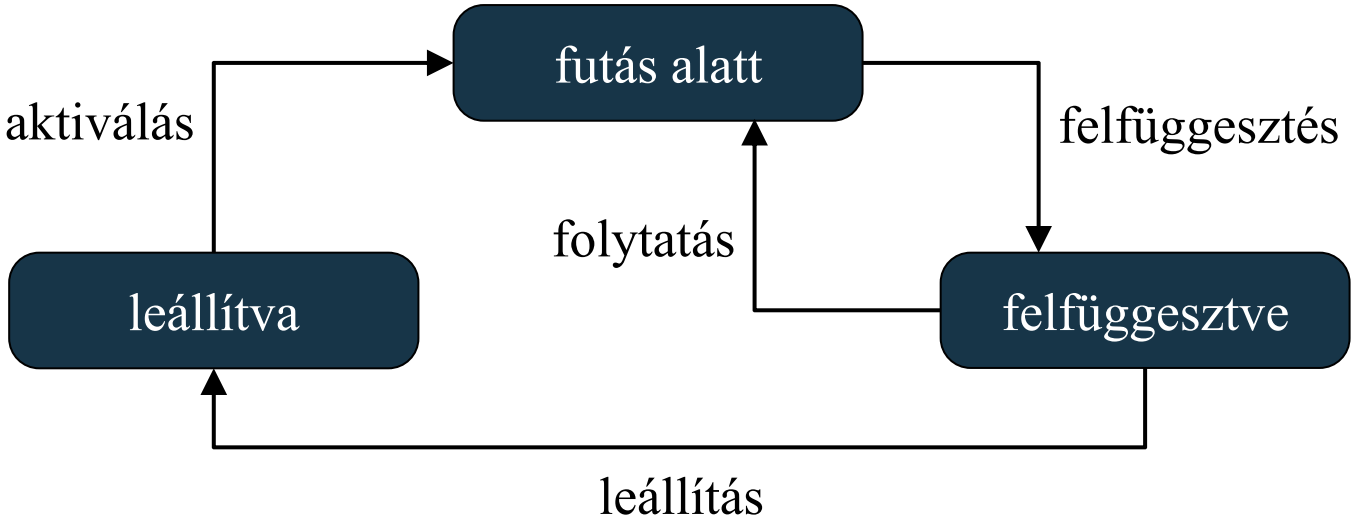
- Az alkalmazások egy biztonságos környezetben futnak
  - nem férhetnek hozzá más alkalmazások adataihoz
  - csak korlátozott módon férhetnek hozzá a rendszer adataihoz (pl. fájlrendszer), és azt is csak engedéllyel
  - szintén engedéllyel használhatják csak az eszközöket (*application manifest*)

- Mindegyik alkalmazás számára rendelkezésre áll egy lokális könyvtár, amely csak az alkalmazás fájljait tárolja
  - Ennek ellenéréséhez nem szükséges külön jogosultság semelyik támogatott platformon
  - Alkalmazás törlésekor a hozzá tartozó adatok is törlődnek

Összetett Avalonia UI alkalmazások

Alkalmazások életciklusa

- A mobil alkalmazások más életciklusban futnak, mint az asztali alkalmazások
  - a *futás alatt (running)* és a *terminált (not running)* állapotok mellett megjelenik a *felfüggesztett* (*suspended*) állapot is, amely akkor lép életbe, ha az alkalmazás a háttérbe (vagy a gép alvó állapotba) kerül, célja a takarékosság



Összetett Avalonia UI alkalmazások

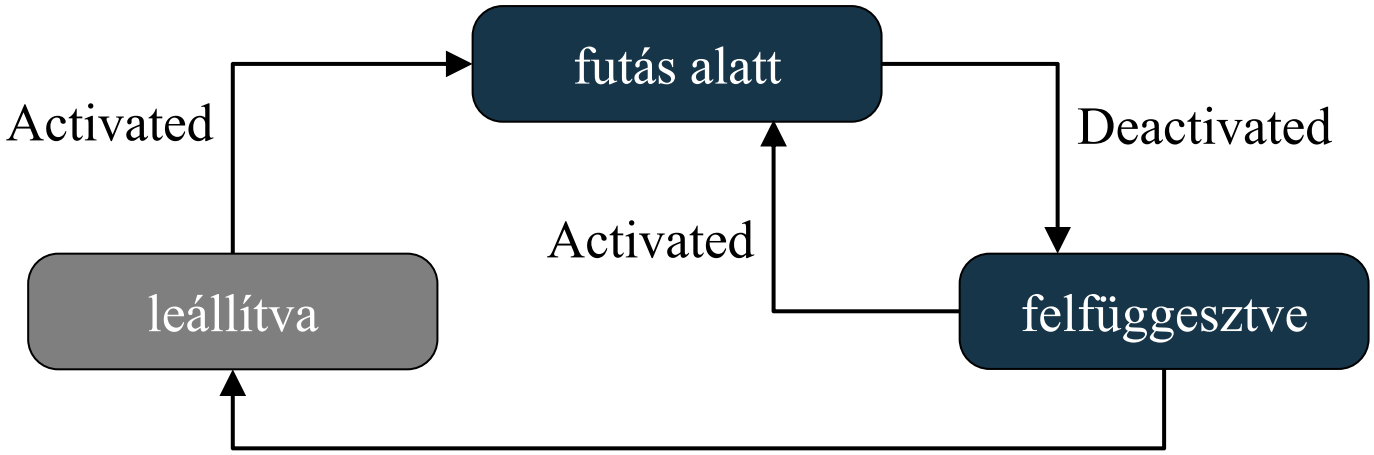
Alkalmazások életciklusa

- A felfüggesztés célja az erőforrásokkal való takarékoskodás
  - a fejlesztőnek törekednie kell rá, hogy felfüggesztett állapotban az alkalmazás minél kevesebb erőforrást igényeljen
  - a futó tevékenységeket célszerű leállítani, az adatokat perzisztálni
- A rendszer úgy is dönthet (pl. ha kevés a memória), hogy a felfüggesztett alkalmazást leállítja, majd újraindítja, ha a felhasználó visszaváltott rá
  - célszerű, hogy a felhasználó ennek ellenére olyan állapotban kapja vissza az alkalmazást, amelyben hagyta, így ezt az állapotot vissza kell állítanunk
  - az állapot eltárolását felfüggesztéskor kell elvégeznünk

Összetett Avalonia UI alkalmazások

Alkalmazások életciklusa

- Az Avalonia UI keretrendszerű mobil alkalmazások egységes életciklus kezeléssel rendelkeznek:
  - az **`App`** osztályunk **`OnFrameworkInitializationCompleted`** metódusát felüldefiniálva, eseménykezelőkkel adható meg az életciklus váltáskor végrehajtandó tevékenység (**`Activated`**, **`Deactivated`**), minden platformra (Android, iOS, macOS).



Összetett Avalonia UI alkalmazások

Alkalmazások életciklusa

- Pl.:

```
if (Application.Current
      .TryGetFeature<IActivatableLifetime>()
      is { } activatableLifetime)
{
    activatableLifetime.Activated += (sender, args) =>
    {
       // alkalmazás aktív
    };
    activatableLifetime.Deactivated += (sender, args) =>
    {
       // alkalmazás inaktív
    };
}
```

Összetett Avalonia UI alkalmazások

Alkalmazások életciklusa

- Asztali alkalmazásoknál Windows és Linux operációs rendszerek esetében hasonló módon az alkalmazás elindítása és leállítása kezelhető, mint életciklus események
  - az **`App`** osztályunk **`OnFrameworkInitializationCompleted`** metódusát felüldefiniálva, eseménykezelőkkel adható meg az alkalmazás elindításakor és leállításakor (**`Startup, Exit`**) végzendő tevékenység.
  - Pl.:

```
desktop.Startup += (sender, args) =>
{
    // ...
};
```

Összetett Avalonia UI alkalmazások

Platformfüggetlen perzisztencia

- A **`System.IO`** névtérben korábban már megismert osztályok és eljárások a fájlkezelés összes lehetőségét biztosítják
  - könyvtárak (**`Directory`**) elérését, listázását (**`GetFiles`**, **`GetDirectories`**), benne könyvtárak létrehozását (**`CreateDirectory`**), fájlok és könyvtárak törlését (**`Delete`**)
  - fájlok (**`File`**) létrehozását, megnyitását (**`Open`**), olvasást (**`ReadAllBytes`**, **`ReadAllLines`**, **`ReadAllText`**), írást (**`WriteAllBytes`**, …), másolást (**`Copy`**), …
  - az írás és olvasás történhet adatfolyamok segítségével is (**`StreamReader`**, **`StreamWriter`**), amelyeket a megszokott módon használhatunk

Összetett Avalonia UI alkalmazások

Platformfüggetlen perzisztencia

- Speciális, jól ismert útvonalakat az **`Environment.SpecialFolder`** *enum* segítségével kérhetünk le
  - ügyelni kell arra, hogy az egyes útvonalak platformfüggően mást jelenthetnek, pl.:
    - Dokumentumok könyvtár asztali operációs rendszereken:
      ```
      Environment.GetFolderPath(
        Environment.SpecialFolder.MyDocuments)
      ```
    - alkalmazás saját adatkönyvtára mobil platformokon:
      ```
      Environment.GetFolderPath(
        Environment.SpecialFolder.
        LocalApplicationData)
      ```
      - ez az útvonal asztali operációs rendszeren azonban nem alkalmazás specifikus

Összetett Avalonia UI alkalmazások

Példa

Feladat: Készítsünk egy vizsgatétel generáló alkalmazást, amely ügyel arra, hogy a vizsgázók közül ketten ne kapják ugyanazt a tételt.

- kiegészítjük életciklus kezeléssel az alkalmazást, eltároljuk a modell állapotát, valamint a generálás állapotát az alkalmazás lokális könyvátárba JSON formátumban szerializálva
- mivel nincs külön perzisztencia, közvetlenül a modelltől kérjük el az információkat, és tároljuk el egyenként
- az alkalmazás életciklusát az **`Application`** vezérli, ezért itt definiálunk egy **`SaveState`** és egy **`LoadState`** eljárást
- indításkor, illetve folytatáskor a korábbi állapotot betöltjük, és ennek megfelelően inicializáljuk a modellt

Összetett Avalonia UI alkalmazások

Példa

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

- a megjelenítést grafikus alakzatokkal (**`Line`**, **`Ellipse`**, **`Rectangle`**) valósítsuk meg
- ugyanakkor továbbra is gombokat jelenítünk meg (amely kattintható), de felüldefiniáljuk a sablont (**`Template`**) egy egyedi felépítéssel (**`ControlTemplate`**), így a gomb megjelenése teljesen más lesz
  - definiáljunk stílusokat (**`Empty`**, **`X`**, **`O`**), amelyeket a nézetmodell **`TicTacToeField`** osztályának 1-1 logikai tulajdonságához köthetünk (**`IsEmpty`**, **`IsX`**, **`IsO`**)
- a mentés egy beégett útvonalon legyen lehetséges; az alkalmazás leállításakor / háttérbe kerülésekor automatikusan mentsünk

Összetett Avalonia UI alkalmazások

Dialógus ablakok

- Avalonia UI keretrendszerben platformfüggetlen módon kérhetünk fájl betöltéséhez és mentéséhez dialógusablakot
  - ehhez a **`StorageProvider`** **`OpenFilePickerAsync()`** és **`SaveFilePickerAsync()`** metódusait használhatjuk
    - betöltésnél megadhatjuk, hogy több fájl is kiválasztható-e (**`AllowMultiple`**)
    - mentésnél megadhatjuk a javasolt fájlnevet (**`SuggestedFileName`**) és alapértelmezett kiterjesztést (**`DefaultExtension`**)
    - választható fájltípusokat, ehhez elérhetőek előre definiált beállítások (pl. **`FilePickerFileTypes.ImageJpg`**), de saját is megadható
  - hasonlóan kérhetjük könyvtár tallózását is a **`OpenFolderPickerAsync()`** eljárás használatával

Összetett Avalonia UI alkalmazások

Dialógus ablakok

- A **`StorageProvider`** osztály ún. szolgáltatásként (*service*) érhető el az Avalonia UI keretrendszerben
  - a szolgáltatások a **`TopLevel`** vezérlőn keresztül érhetőek el
  - az ablakok, nézetek, és felületi vezérlők egymásba ágyazásával a vezérlők egy fa struktúrát alkotnak (*visual tree*)
  - ezen fa struktúra gyökér eleme a **`TopLevel`** vezérlő
  - asztali ablakos alkalmazásoknál a **`Window`** a **`TopLevel`**
  - mobil alkalmazásoknál platform specifikus lehet, pl. Android esetén a **`MainActivity`** lesz az
  - platformfüggetlen módon bármely vezérlőhöz lekérdezhető: **`TopLevel.GetTopLevel(control);`**

Összetett Avalonia UI alkalmazások

Dialógus ablakok

- Pl.:

```
var files = await
TopLevel.StorageProvider.OpenFilePickerAsync(
    new FilePickerOpenOptions
    {
        Title = "Select file to load",
        AllowMultiple = false,
        FileTypeFilter = new[]
        {
            new FilePickerFileType("Data file")
            {
                Patterns = new[] { "*.data" }
            }
        }
    });
```

Összetett Avalonia UI alkalmazások

Példa

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

- a mentéshez az útvonalat a felhasználó választhassa meg
  - csak a `.data` kiterjesztésű fájlok mentését és betöltését kínáljuk fel
  - Android operációs rendszer támogatásához az `AndroidManifest.xml` állományban a szükséges `READ_EXTERNAL_STORAGE` és `WRITE_EXTERNAL_STORAGE` jogosultságokat kérjük meg
- az alkalmazás leállításakor / háttérbe kerülésekor továbbra is automatikusan készítsünk egy mentést

Összetett Avalonia UI alkalmazások

Animációk

- Avalonia UI keretrendszerben az animációk a WPF keretrendszerben megismertekhez hasonló módon támogatott
  - stílusként (**`Styles`**) adható meg a nézet vagy vezérlő szintjén
  - Pl. áttetszőség és forgatás egy időben történő animálása:

```
<Animation Duration="0:0:3" IterationCount="4">
  <KeyFrame Cue="0%">
     <Setter Property="Opacity" Value="0.0"/>
     <Setter Property="RotateTransform.Angle"
             Value="0.0"/>
  </KeyFrame>

  <KeyFrame Cue="100%">
    <Setter Property="Opacity" Value="1.0"/>
    <Setter Property="RotateTransform.Angle"
            Value="90.0"/>
  </KeyFrame>
</Animation>
```

Összetett Avalonia UI alkalmazások

Átmenetek

- Avalonia UI keretrendszerben átmeneteket is alkalmazhatunk,
  - ezek működését leginkább a webfejlesztésből ismert CSS animációkhoz hasonlíthatjuk
  - a tulajdonság változása nem pillanatszerű, hanem animált lesz

```
<Window.Styles>
  <Style Selector="Rectangle.red">
    <Setter Property="Fill" Value="Red"/>
    <Setter Property="Opacity" Value="0.5"/>
  </Style>
  <Style Selector="Rectangle.red:pointerover">
    <Setter Property="Opacity" Value="1"/>
  </Style>
</Window.Styles>

<Rectangle Classes="red">
  <Rectangle.Transitions>
    <Transitions>
      <DoubleTransition Property="Opacity" Duration="0:0:0.2"/>
    </Transitions>
   </Rectangle.Transitions>
</Rectangle>
```

Összetett Avalonia UI alkalmazások

Animációk vs. átmenetek

- Az animációk és az átmenetek egyaránt vizuális változásokat hoznak létre, de különböző célokra szolgálnak és eltérően működnek.

	Animations	*Transitions*
Indítás	Kód, triggerek, események	Property értékváltozás
Cél	Összetettebb mozgatások, animációk	Property-k animált értékváltozása
***Keyframe*-k támogatása**	Igen	Nem
Több *property* animálása	Igen	Jellemzően 1 *transition / property*
Javasolt felhasználás	Látványos effektek, animáció sorozatok	UI reszponzívabb megjelenítése
Teljesítmény	Kicsit erőforrásigényesebb	Nagyon könnyűsúlyú

Összetett Avalonia UI alkalmazások

Példa

Feladat: Készítsünk egy dinamikus méretezhető táblát, amely három szín között (piros, fehér, zöld) állítja a kattintott gombot, valamint a vele egy sorban és oszlopban lévőket.

- a színt a nézet adja meg, így a nézetmodell nem adhat vissza konkrét színt, csak egy sorszámot (0 és 2 között), amely alapján a szín állítható (`ColorNumber`)
- a színt stílus osztályok segítségével állítsuk a nézetben, a gomb emellett animálódjon, amennyiben a kurzort rávisszük (csak asztali környezetben), illetve ha lenyomva tartjuk
- a stílusokat a nézet erőforrásaként hozzuk létre