12. Gyakorlat

A mai gyakorlaton egy Ping-Pong játékot fogunk elkészíteni animációkkal. Az programot WPF alkalmazásként készítjük el, amely csak nézet réteget fog tartalmazni.

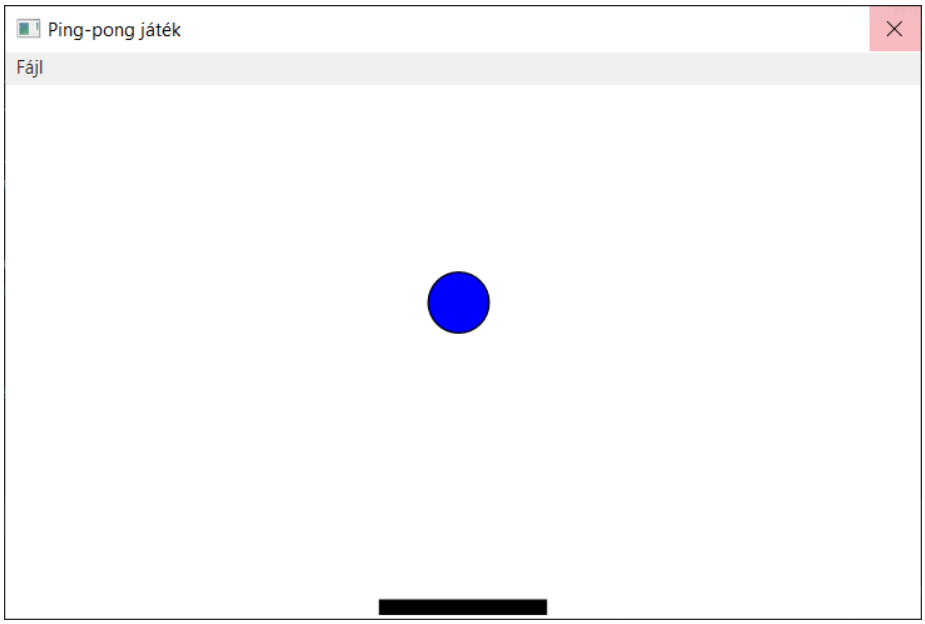

Figure 1: Ping-pong játék

Animációk WPF-ben

A WPF támogatja az animációk végrehajtását, amely lényegében függőségi tulajdonságok adott időn keresztül történő folyamatos módosítását jelenti. A grafikus vezérlők lényegében bármelyik tulajdonságát animálhatjuk, definiálva a kezdőállapotot (`From`), a végállapotot (`To`), valamint az időt (`Duration`). A különböző típusokhoz megfelelő animációs típus létezik: `DoubleAnimation`, `ThicknessAnimation`, `ColorAnimation`, stb.

Az animációkat deklaratívan XAML kóddal vagy C# háttérkódban is megadhatjuk. Több animációt forgatókönyvbe (`Storyboard`) foglalhatunk - XAML kód esetén kötelező. Például:

```
<Button Name="myButton" ...>

<Storyboard Storyboard.TargetName="myButton" Duration="0:00:04">
    <DoubleAnimation From="1" To="0" Storyboard.TargetPropety="Opacity" />
</Storyboard>
```

Ugyanez C# háttérkóddal:

```
DoubleAnimation myAnimation = new DoubleAnimation
{
    From = 1,
    To = 0,
    Duration = new Duration(TimeSpan.FromSeconds(4))
};
myButton.BeginAnimation(Button.OpacityProperty, myAnimation);
```

XAML (MainWindow.xaml)

- Az ablak szélessége (`Width`) legyen 600, magassága (`Height`) 400 pixel.
- A nézet elemeit egy négyzetrácsban (`Grid`) fogjuk tárolni.
- A rácsba kerüljön egy felülre igazított (`VerticalAlignment=Top`) menü (`Menu`), amely egy menüponttal fog rendelkezni (`MenuItem`). A menüpontnak legyen két saját menüpontja, amelyek lehetőséget biztosítanak az új játék kezdésére, és a kilépésre. Ezek `Click` attribútmához rendeljük hozzá a nézet megfelelő eseménykezelőit (`NewGame`, `Exit`)!
- Vegyünk fel a rácson belül egy ellipszist (`Ellipse`) a labda ábrázolásához, ennek neve (`Name`) legyen *ellipseBall*! Az ellipszis szélessége (`Width`) és magassága (`Height`) is legyen 40 pixel! A `Margin` attribútummal helyezzük el az ellipszist a bal oldaltól 270, a felső sávtól 140 pixel távolságra, majd igazítsuk az ellipszist vízszintesen (`HorizontalAlignment`) balra, függőlegesen felfelé. Az ellipszist töltsük ki (`Fill`) kék színnel.
- Vegyünk fel a rácson belül egy négyszöget (`Rectangle`) az ütő ábrázolásához, ennek neve (`Name`) legyen *rectanglePad*! Magassága 10 pixel, szélessége 120 pixel legyen, `Margin` attribútuma (235,350,0,0). Igazítsuk függőlegesen felfelé, vízszintesen balra, töltsük ki fekete színnel.
- A C# kódból az ütőre és a labdára a nevükkel fogunk hivatkozni.

Nézet mögötti kód (MainWindow.xaml.cs)

Felhasználandó típusok

- `Thickness`: számnégyes, amellyel rendre a balról, felülről, jobbról, illetve lentről való pozícionális távolság adható meg.
  - Propertyk: `Left`, `Top`, `Right`, `Bottom`
- `ThicknessAnimation`: olyan animációtípus, amellyel `Thickness` típusú tulajdonságok animálhatóak.
  - Propertyk: `From`, `To`, `Duration`, `SpeedRatio`. Utóbbival módosíthatjuk az idő relatív múlását az adott animáció tekintetében.

Mezők

Vegyük fel a következő változókat!

- `Thickness` típusú változók a labda kezdő-, aktuális és következő pozíciójának, valamint az ütő kezdő- és aktuális pozíciójának tárolására (*ballStartPosition*, *ballCurrentPosition*, *ballNextPosition*, *padStartPosition*, *padCurrentPosition*)
- `ThicknessAnimation` típusú változók az ütő és a labda animációjának reprezentációjához (*ballAnimation*, *padAnimation*)
- egy `DateTime` típusú változó, amely a játék kezdésének időpontját tárolja (*startTime*)
- egy logikai típusú változó annak tárolására, hogy a játék elkezdődött-e (*isStarted*)
- egy lebegőpontos szám típusú változó a labda sebességének tárolására (*speedFactor*)

Eseménykezelők

- `NewGame(object, RoutedEventArgs)`: meghívja a játékot elindító `StartGame` metódust.
- `Navigate(object, KeyEventArgs)`: az ütő mozgatásáért felel. Kérjük le az XAML kódtól az ütő jelenlegi pozícióját a `Margin` attribútummal! A `KeyEventArgs` típusú paraméterből lekérhetjük a felhasználó által leütött billentyűt (`Key`). Ha a felhasználó a bal nyilat (`Key.Left`) ütötte le, és az ütő az ablak bal szélétől távolabb van, toljuk el az ütőt -100 pixellel, ha jobb nyilat ütötte le, és az nincs a jobb szélen, akkor 100 pixellel toljuk el! Az eltolást az `AnimatePad` metódus meghívásával végezhetjük el.
- `BallLayoutUpdated(object, EventArgs)`: a labda mozgását leíró eseménykezelő. Csak akkor csinál bármit, ha már elindult a játék. Szeretnénk kezelni minden esetet, amikor a labda falnak vagy az ütönek ütközik, ezt a labda és az ütő `Margin` attribútumának elemei (`Top`, `Left`) és az ablak méretei alapján állapíthatjuk meg. A vesztes eset kivételével minden eset végén hívjuk meg az `AnimateBall` metódust. A következő eseteket kezeljük:
  - A labda az ütőnek ütközik balról: növeljük meg a labda sebességét 5%-kal! Állítsuk be a *ballNextPosition* `Top` propertyjét 0-ra (felfelé akarunk mozogni), a `Left` propertyjét pedig az eddigi pozícióhoz képest toljuk el pl. -200-zal (ablak szélessége / 3), mert bal felé is szeretnénk haladni.
  - A labda az ütőnek ütközik jobbról: növeljük meg a labda sebességét 5%-kal! Állítsuk be a *ballNextPosition* `Top` propertyjét 0-ra (felfelé akarunk mozogni), a `Left` propertyjét pedig az eddigi pozícióhoz képest toljuk el pl. 200-zal, mert jobb felé is szeretnénk haladni.
  - A labda a tetőnek ütközik: a *ballNextPosition* `Top` propertyje kapja értékül az ablak magasságát (az ablak alja felé kezdjen mozogni).
  - A labda az ablak bal oldalának ütközik: a *ballNextPosition* `Left` propertyje kapja értékül az ablak szélességét (jobbra kezdjen mozogni).
  - A labda az ablak jobb oldalának ütközik: a *ballNextPosition* `Left` legyen 0 (balra kezdjen mozogni).

  - A labda túlmegy az ütőn: dobjunk fel egy `MessageBox`-ot (`Show`) a "Játék vége" üzenettel, és írjuk ki, hogy hány másodpercig tartott a játék, majd állítsuk meg a játékot.
- `Exit(object, RoutedEventArgs)`: bezárja az alkalmazást.

Privát metódusok

- `StartGame`: beállítja a változók kezdeti értékeit. A *startTime* változót állítsuk az aktuális időre (`DateTime.Now`), a labda aktuális pozícióját a kezdőpozíciójára, a kezdőiránynak pedig adjunk meg véletlenszerű értékeket! A *speedFactor* legyen 1, az *isStarted* értéke igaz. Hívjuk meg az `AnimateBall` metódust, ezzel elkezdődik a játék.
- `StopGame`: hamisra állítja az *isStarted* változó értékét.
- `AnimateBall`: definiáljuk felül a *ballAnimation*-t! A `From` property kapja a *ballCurrentPosition* értékét. A `To` property kapja a *ballNextPosition* értékét. A `Duration` property értékét állítsuk 5 milliszekundumra (`TimeSpan.FromMilliseconds(5)`)! A `SpeedRatio` legyen a *speedFactor* és a `BallTravelDistance` metódus eredményének hányadosa. Végül hívjuk meg a labda `BeginAnimation` metódusát, amely a következő paramétereket várja
  - melyik propertyt kell beállítani (`Ellipse.MarginProperty`),
  - mire (*ballAnimation*),
  - mi történjen, ha még a befejezés előtt újra meghívásra kerül ez a metódus (`HandoffBehavior.SnapshotAndReplace`; azt szeretnénk, ha megszakítaná az animáció kirajzolását, és újat kezdene).
- `AnimatePad(Int32)`: definiáljuk felül a *padAnimation*-t! A `From` property kapja a *padCurrentPosition* értékét. A `To` property legyen az ütő `Margin` attribútuma a jobb és bal értékek megfelelő eltolásával! A `Duration` property értékét állítsuk 100 milliszekundumra! Végül hívjuk meg az ütő `BeginAnimation` metódusát az előzőnek megfelelő módon.
- `BallTravelDistance`: kiszámítja a *ballNextPosition* és a *ballCurrentPosition*, mint koordinátarendszerbeli pontok távolságát.

Konstruktor

- Hívjuk meg az alkalmazás `InitializeComponent` metódusát, amely inicializálja az ablakot!
- A *ballStartPosition* kapja a labda `Margin` attribútumának értékét.
- A *padStartPosition* és a *padCurrentPosition* kapja az ütő `Margin` attribútumának értékét.
- A *ballNextPosition* változót egyelőre csak definiáljuk.
- Az *isStarted* változót állítsuk hamisra!
- A `KeyDown` eseményre iratkozzunk fel a `Navigate` eseménykezelővel!
- A labda `LayoutUpdated` eseményére iratkozzunk fel a `BallLayoutUpdated` eseménykezelővel!