

Szoftvertchnikák önálló gyakorlat

3. önálló feladat

Felhasználói felület kialakítása

Windows Forms, dokkolás, horgonyzás,
menük, TreeView, ListView, grafikus megjelenítés

A gyakorlatot kidolgozta: Kővári Bence, (Benedek Zoltán)

Utolsó módosítás ideje: 2020.03.24.

Tartalom

TARTALOM	2
BEVEZETÉS	2
FELADAT 1 – MENÜ	5
FELADAT 2 – DIALÓGUSABLAK	5
FELADAT 3 – FÁJLKEZELŐ	7
FELADAT 4 – RAJZOLÁS	8
EMLÉKEZTETŐ	12
OPCIONÁLIS PLUSZ FELADAT – 3 IMSC PONTÉRT	12

Bevezetés




Az önálló feladat a 3-5. előadásokon elhangzottakra épít. A feladatok gyakorlati hátteréről a „3. gyakorlat – Felhasználói felületek kialakítása” laborgyakorlat szolgál.



A fentiekre építve, jelen önálló gyakorlat feladatai a feladatleírást követő rövidebb iránymutatás segítségével elvégezhetők.

Az önálló gyakorlat célja:

- Windows Forms tervező használatának gyakorlása
- Alapvető vezérlők (gomb, szövegdoz, menük, listák) használatának gyakorlása
- Eseményvezérelt programozás gyakorlása
- Grafikus megjelenítés gyakorlása Windows Forms technológiával

A feladat publikálásának és beadásának alapelvei megegyeznek az előző feladatával, pár kiemelt követelmény:

- A feladathoz tartozó GitHub Classroom hivatkozás:
<https://classroom.github.com/a/K5Y9ImBL>
 A munkamenet megegyezik az előző házi feladatával: a fenti hivatkozással mindenkinek születik egy privát repója, abban kell dolgozni és a határidőig a feladatot beadni.
-  Az elindulással **ne várd meg a határidő közeledtét**, legalább a saját repó létrehozásáig juss el mielőbb. Így ha bármi elakadás lenne, még időben tudunk segíteni.
-  A repository gyökérmappájában található neptun.txt fájlba írd bele a Neptun kódod, csupa nagybetűvel.
-  A beadott megoldások mellé külön indoklást, illetve leírást nem várunk el, ugyanakkor az elfogadás feltétele, hogy a beadott kódban a feladat megoldása szempontjából relevánsabb részek **kommentekkel legyenek ellátva**.

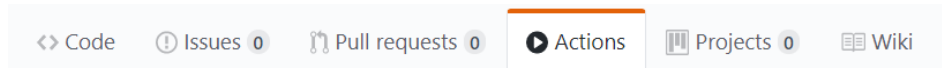
-  A munka során a kiindulási repóban levő solutionben/projektben kell dolgozni, új projektet/solutiont ne hozz létre.
-  A kiinduló projektben van egy .github/workflows mappa, ennek tartalmát tilos megváltoztatni, törölni, stb.
- A megoldást a tanszéki portálra **nem** kell feltölteni, de az eredményt itt fogjuk meg hirdetni a kapcsolódó számonkérés alatt.
- Amikor a házi feladatod beadottnak tekinted, célszerű ellenőrizni a GitHub webes felületén a repository-ban a fájlokra való rápillantással, hogy valóban minden változtatást push-oltál-e.
- Az egyes feladatok leírásánál **kék színnel** megjelöltük azokat az azonosítókat, szövegeket, melyeknél fontos, hogy a beadott feladatban a megadott érték szerepeljen.

A házi feladat előellenőrzése és hivatalos értékelése

- Minden egyes alkalommal, miután a GitHub-ra push-oltál kódot, a GitHub-on automatikusan lefut a feltöltött kód (elő)ellenőrzése, és meg lehet nézni a kimenetét! Az ellenőrzőt maga a GitHub futtatja. A pusht követően a feladat egy várakozási sorba kerül, majd adott idő után lefutnak az ellenőrző tesztek. Azt nem lehet tudni, mennyi ez az idő, a GitHub-on múlik. Amikor csak egy-két feladat van a sorban a szervezetre (ez nálunk a tárgy), akkor a tapasztalatok alapján az ellenőrzés 1-2 percen belül elindul. De ha a tárgy alatt egyszerre sokan kezdik majd feltölteni a megoldást, akkor ez jó eséllyel belassul, erről még nincs tapasztalatunk. Nem érdemes ezért sem utolsó pillanatra hagyni a beadást: lehet, hogy ekkor a késleltetések miatt már nem kapsz esetleg időben visszajelzést.
- **Hivatalosan a feladat azon állapota kerül értékelésre, amely a határidő lejártakor GitHub-on fent van, ebben nincs változás** a korábbi házi feladatokhoz képest. A hivatalos ellenőrzést szokásos módon, saját, oktatói környezetben végezzük és az eredményt a tárgy honlapján publikáljuk a számonkérésnél. Vagyis a hivatalos eredmény tekintetében teljesen mindegy, hogy a GitHub-on a határidő lejárt lefutott-e már bármiféle (elő)ellenőrzés, vagy hogy az ellenőrzés estleg csak később tudott elindulni. A GitHub általi ellenőrzés csak azt a célt szolgálja, hogy még a határidő lejárt előtt visszajelzést kaphasson mindenki. A hivatalos ellenőrzés tartalmazhat még plusz lépéseket a GitHub alapú előellenőrzéshez képest (pl. vannak-e kódjegyzések)!
- **Azt kérnénk, hogy ne apránként push-oljatok, lehetőleg csak a kész, átnézett megoldást tegyétek fel.** Ez nem a legyszerencsebb, de a GitHub korlátozott időt biztosít az ellenőrzők futtatására: ha elfogy a havi keret, akkor már nem fogtok visszajelzést kapni, csak a határidő utáni hivatalos ellenőrzés kimenetét kapja meg mindenki.
- Ez most még egy kísérleti projekt (korábbi években nem volt semmilyen előellenőrzés), egyelőre tapasztalatot gyűjtünk. Ha valaki az útmutatóban inkonzisztenciát talál, vagy az ellenőrző adott helyzetben nem elég robosztus és indokolatlanul panaszkodik, Benedek Zoltán felé legyen szíves jelezni.

A GitHub által futtatott ellenőrzések megtekintése

1. GitHub-on a navigálás a repository-hoz
2. Actions tabfültre váltás



3. Itt megjelenik egy táblázat, minden push által futtatott ellenőrzéshez egy külön sor, a tetején van legfrissebb. A sor elején levő ikon jelzi a státuszt: vár, fut, sikeres, sikertelen lehet. A sor szövege a Git commit neve.
4. Egy sorban a commit nevén kattintva jelenik meg egy átfogó oldal az ellenőrző futásáról, ez sok információt nem tartalmaz. Ezen az oldalon baloldalt kell a build-and-check linken kattintani, ez átnavigál az ellenőrzés részletes nézetére. Ez egy „élő” nézet, ha fut a teszt, folyamatosan frissül. Ha végzett, a csomópontokat lenyitva lehet megnézni az adott lépés kimenetét. Ha minden sikerült, egy ehhez hasonló nézet látható:



Itt a lényegi részek ezek, a többi csak technikai körítés:

- Run UI tests - Feladat1 (Menü)
- Run UI tests - Feladat2 (Dialógusablak)
- Run UI tests - Feladat3 (Fájlkezelő)
- Run UI tests - Feladat4 (Rajzolás)
- Run UI tests - IMSc opcionális

Ha valamelyik lépés sikertelen, a csomópont kibontva a teszt kimenete utal a hiba okára. A „Error Message”-re, ill. az „Assert”-re érdemes szövegesen keresni a kimentben, ennek a környékén szokott lenni hivatkozás a hiba okára. Pl.:

Error Message:

12 Assert.Fail failed. Nem található a következő felületelem: File menü

Ez azt jelzi, hogy a File menü hiányzik a megoldásban.

Természetesen az opcionális feladathoz tartozó IMSc tesztnek nem kell sikeresnek lenni ahhoz, hogy a házi feladat elfogadásra kerüljön.

Feladat 1 – Menü

Bevezető feladat

A főablak fejléce a **MiniExplorer** szöveg legyen. Ehhez az űrlapunk Text tulajdonságát állítsuk be erre a szövegre (a kiinduló projektben ez már jó esetben be van állítva, de ha mégsem lenne, akkor javítsuk).

Feladat

Vezessünk be egy menüsort a főablakunk (Form1) tetején. A menüben egyetlen elem legyen „File” néven, 3 almenüvel:

- **Open**: később adunk neki funkciót
- **Exit**: kilép az alkalmazásból

Lényeges, hogy a menük szövegei a fent megadottak legyenek!

Megoldás

1. Húzzunk be a felületre egy **MenuStrip** vezérlőt
2. A **MenuStrip** vezérlő bal szélén megjelenő szövegdobozba írjuk be, hogy „File”, ezzel létrehoztuk a főmenüt
3. Az újonnan létrehozott főmenüt kijelölve hozzuk létre a 3 almenüt.
4. Egyesével kijelölgetve a menüelemeket, töltsük ki a nevüket (**miOpen**, **miExit**).
5. Valósítsuk meg a kilépés funkciót a kapcsolódó gyakorlathoz hasonlóan.

Feladat 2 – Dialógusablak

A Windows Forms világban gyakran fordul elő, hogy egyedi vezérlőket, vagy ablak típusokat akarunk definiálni, továbbá ezek és a programunk többi része között információkat akarunk átadni. A következő feladat erre mutat példát.

Feladat

Készíts egy új ablak (Form) típust **InputDialog** néven (a fejléce is legyen **InputDialog**), mely egy szövegdobozt (TextBox), továbbá egy „Ok” és egy „Cancel”

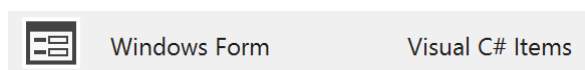
feliratú gombot tartalmaz. A két gomb bezáráskor beállítja a `Form DialogResult` tulajdonságát `DialogResult.OK`, illetve `DialogResult.Cancel` értékre. Az ablak ezen felül tartalmazzon egy publikus, `string` típusú, `Path` nevű tulajdonságot!

Az ablak tartalma arányosan változzon az átméretezés során (`TextBox` szélessége növekedjen, a gombok pedig a hozzájuk közelebbi sarokhoz képest rögzített pozícióban maradjanak).

A feladatot próbáld meg önállóan megoldani, majd a lenti leírás alapján ellenőrizd a megoldásod!

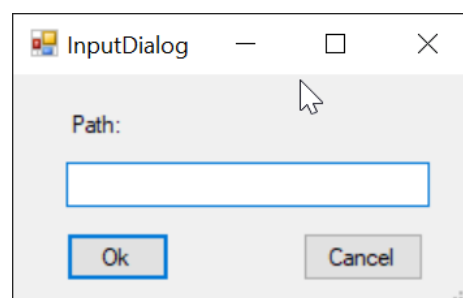
Megoldás

1. Hozzunk létre egy új Windows Forms alkalmazást
2. Adjunk hozzá a projekthez egy új ablak típust (projekten jobb klikk, majd Add / Windows Form), a neve legyen `InputDialog`.



3. Adjunk az ablakhoz egy `TextBox`, egy `Label` és két `Button` vezérlőt. Rendezzük el őket a felületen és állítsuk be a tulajdonságaikat:

- a. `TextBox`
 - i. **Name:** `tPath`
- b. `Button`
 - i. **Name:** `bOk`
 - ii. **Text:** "Ok"
 - iii. **DialogResult:** `OK`
- c. `Button`
 - i. **Name:** `bCancel`
 - ii. **Text:** "Cancel"
 - iii. **DialogResult:** `Cancel`
- d. `Label`
 - i. **Text:** "Path"
- e. `InputDialog` (maga a Form)
 - i. **AcceptButton:** `bOk`
 - ii. **CancelButton:** `bCancel`



A dialógusablak elkészítésekor kihasználjuk azt, hogy egy modális dialógusablakot nem csak a `Close` utasítással lehet bezárni, hanem úgy is, ha értéket adunk a `DialogResult` tulajdonságának. Ezt kódból is megtehetjük volna, de mi most a gombok erre szolgáló mechanizmusát használtuk a `Form Accept` és `Cancel` button tulajdonságaival.

4. Az egyes vezérlők `Anchor` tulajdonságainak beállításával érjük el, hogy az ablak tartalma arányosan változzon az átméretezés során (`TextBox` szélessége

növekedjen, a gombok pedig a hozzájuk közelebbi sarokhoz képest rögzített pozícióban maradjanak).

5. Vegyünk fel egy `Path` nevű tulajdonságot az `InputDialog.cs` fájlba, mely a `TextBox` tartalmát teszi elérhetővé az osztályon kívülről is. (A tervezői nézet és a forrásnézet között az F7 billentyűvel válthatunk)

```
public string Path
{
    get { return tPath.Text; }
    set { tPath.Text = value; }
}
```

6. Kössük be a dialógusablakot a főablakba! Ehhez kattintsunk duplán a „Open” menüelemre és írjuk meg a dialógusablak megnyitásának kódját.

```
private void miOpen_Click(object sender, EventArgs e)
{
    InputDialog dlg = new InputDialog();
    if (dlg.ShowDialog() == DialogResult.OK)
    {
        string result = dlg.Path;
        MessageBox.Show(result);
        // TODO: további lépések...
    }
}
```

Tipp: a WinForms világban rendkívül gyakori, hogy egy adott információ különböző szintű elérésért egy vezérlő és egy tulajdonság is felel (mint esetünkben a `tPath` szövegdoz és a `Path` tulajdonság). A vezérlők neveinek prefixálásával (amit itt is alkalmaztunk) elkerülhetjük a nem kívánt névütközéseket.

Fontos: A `MessageBox.Show(result);` sort kommentezzük is ki, a későbbiekben zavaró lenne.

Feladat 3 – Fájlközelítő

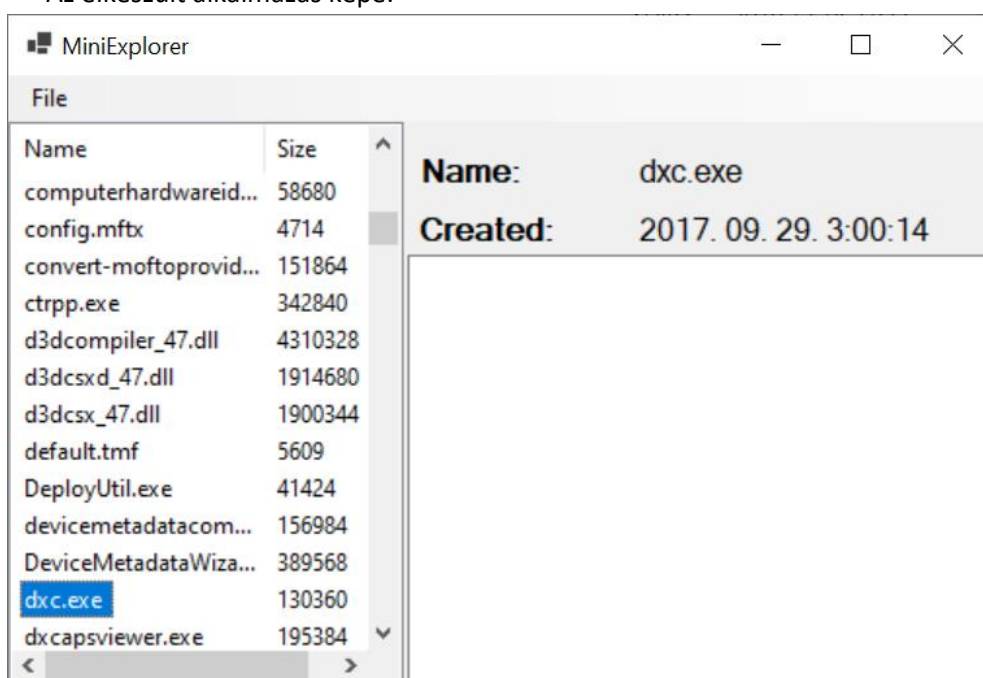
Feladat

A meglévő kódunkból kiindulva valósítsunk meg egy fájl nézegető alkalmazást. Az alkalmazás felületét osszuk két részre. Miután a felhasználó az „Open” menüponttal bekért egy mappanevet (a korábban elkészített `InputDialog` felhasználásával), a bal oldalon listázzuk ki az adott mappában található fájlok neveit és méreteit két külön oszlopban („Name” és „Size” fejlécű oszlopok). A form jobb oldalát egy fix magasságú panel (a neve legyen: `detailsPanel`) és egy többsoros szövegdoz töltse ki. A panelen mindig az aktuálisan kiválasztott fájl nevét és létrehozásának dátumát mutassuk (egy `lName` illetve `lCreated` nevű `Label` típusú vezérlő segítségével). Amennyiben a felhasználó a fájllistából egy fájlra duplán kattint, a többsoros szövegdozban jelenítsük meg a fájl tartalmát (szöveges formátumban). A többsoros szövegdoz neve `tContent` legyen.

Megoldás

A feladat megoldásához a kapcsolódó gyakorlatban már alkalmazott, illetve az itt korábban megismert elemeket kell alkalmazni és kombinálni. A megoldás lépéseit csak nagy vonalakban adjuk meg, néhány kiegészítő segítséggel:

- Az ablak területének kettéosztására használhatjuk ismét a `SplitPanel` vezérlőt.
- Az aktuálisan kiválasztott elem adatainak megjelenítését a `ListView` `SelectedIndexChanged` eseményével célszerű megoldani.
- Ahhoz, hogy a `TextBox` vezérlő kitölthesse a rendelkezésére álló teret, nem elég a `Dock` tulajdonságát `Fill`-re állítani, szükséges a `Multiline` tulajdonság `true`-ra állítása is.
- Egy fájl tartalmát egyszerűen betölthetjük egy stringbe a `File` statikus osztály `ReadAllText(filename)` függvényével.
- A `FileInfo` osztály `FullName` tulajdonsága megadja egy fájl teljes nevét, a `CreationTime` pedig létrehozásának idejét (melyet a `ToString()` művelettel alakítsunk stringé).
- Ne felejtjük el, hogy a felhasználó többször egymás után is választhat mappát az „Open” menüponttal. Az új mappa tartalmának betöltése előtt az aktuális fájl listát mindig üríteni kell.
- Az elkészült alkalmazás képe:



Feladat 4 – Rajzolás

Feladat

Amennyiben a felhasználó megnyitott egy fájlt, akkor a megnyitott fájl tartalmát adott időközönként frissítjük, a frissítési időköz **4** másodperc. A frissítés jelzésére a kijelölt fájl adatait (név és létrehozás dátuma) tartalmazó panel felső felére (**0,0** koordinátából kezdve) rajzoljunk ki **zöld** színnel egy **5** pixel magas, kezdetben **120** pixel széles kitöltött

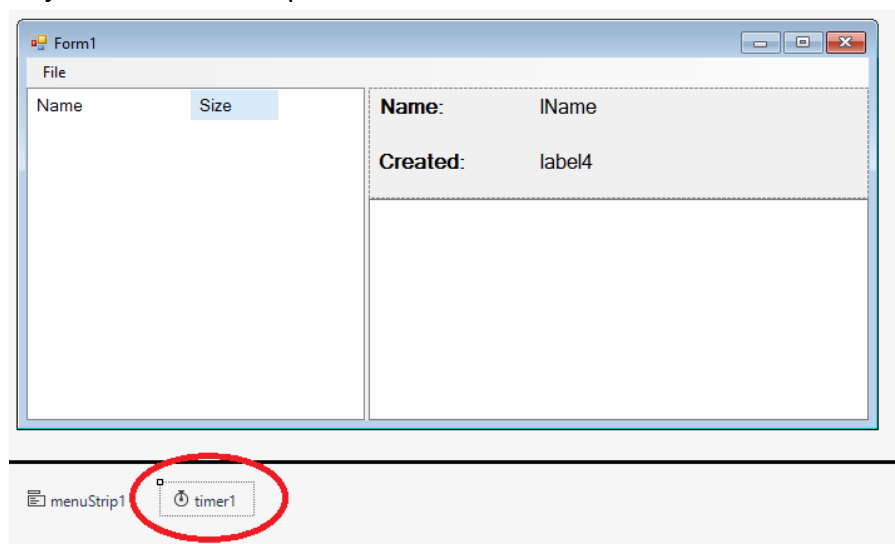
téglalapot. A téglalap hossza a következő frissítésig hátralevő idővel legyen arányos: ennek megfelelően minden tizedmásodpercben arányosan csökkentsük a hosszát. Így minden frissítési időköz végén a téglalap hossza nulla lesz. A frissítési időköz végén (amikor a téglalap hossza elérte a 0-t) a korábban kiválasztott fájl tartalmát töltjük be újból és kezdjük elejéről a folyamatot. Az időzítésre `Timer` komponenst használjunk!

A feladatot próbáld meg önállóan megoldani, majd a lenti leírás alapján ellenőrizd a megoldásod!

Megoldás

A megoldás alapját egy `Timer` komponens fogja adni. Ez egy olyan vezérlő, mely nem rendelkezik vizuális felülettel, csupán néhány testre szabható tulajdonsággal és egy `Tick` eseménnyel, mely az `Interval` tulajdonságban (milliszekundumban) megadott időközönként automatikusan meghívódik. Első lépésként ezt az ütemezést állítjuk be.

1. Húzzunk egy `Timer` komponenst (Toolbox / Components) `Form1`-re! Figyeljük meg, hogy a komponens csupán a Form alatti szürke területen jelenik meg. Itt tudjuk kijelölni a későbbi lépésekhez.



2. Ellenőrizzük, hogy az `Interval` tulajdonsága 100-ra van állítva. Ez 100 milliszekundumonként, vagyis minden tizedmásodpercben kiváltja a `Tick` eseményt.
3. Állítsuk a `Name` tulajdonságot `reloadTimer`-re!
4. Vezessünk be néhány új tagváltozót a `Form1` osztályban:
 - `loadedFile` az utoljára betöltött fájl adatait tartalmazni
 - `counter` az újratöltésig szükséges tizedmásodpercek számát tartalmazza, a későbbiekben minden tizedmásodpercben eggyel csökkentjük egy időzítő segítségével, míg el nem éri a nullát
 - `counterInitialValue` a `counter` számláló kezdőértéke (ahonnan visszaszámol).

A tagváltozókat az osztály elejére szoktuk beszúrni:

```
public partial class Form1 : Form
```

```

{
    FileInfo loadedFile = null;
    int counter;
    readonly int counterInitialValue;

    ...
}

```

5. A konstruktorban állítsuk be a `counterInitialValue` értékét (később ez nem is változik):

```

public Form1()
{
    InitializeComponent();
    counterInitialValue = <a frissítési
                                időköznek megfelelő érték>
}

```

(A `counterInitialValue` értékét a fenti kódban neked kell meghatározni: számítsd ki a frissítési időköz és az `timer Interval` alapján!)

6. Egészítsük ki a duplakattintást kezelő eseménykezelőnket, hogy ne csak betöltse a fájlt, hanem:
- Indítsa el a `Timer`-t a `reloadTimer.Start()` hívással
 - Állítsa be `counter` értékét `counterInitialValue`-ra
 - Állítsa be `loadedFile` értékét a mindenkor kiválasztott fájl leírójára

Megjegyzés: a megoldás minden egyes új fájl megnyitásakor meghívja a `Timer` osztály `Start` függvényét. Ez nem jelent gondot, mivel ilyenkor a már elindított `Timer` egyszerűen fut tovább és figyelmen kívül hagyja a további `Start` hívásokat.

7. Iratkozzunk fel `Timer` komponens `Tick` eseményére. Ehhez `reloadTimer` kijelölése után a Property Editor-ban az events fülön kattintsunk duplán a `Tick` eseményre, ezzel létrejön a kapcsolódó eseménykezelő (`reloadTimer_Tick`). Töltsük ki a kódját:

```

private void reloadTimer_Tick(object sender, EventArgs e)
{
    counter--;

    // Fontos! Ez váltja ki a Paint eseményt és ezzel a
    // a téglalap újrarajzolását
    detailsPanel.Invalidate();

    if (counter <= 0)
    {
        counter = counterInitialValue;
        tContent.Text = File.ReadAllText(loadedFile.FullName);
    }
}

```

A fenti megoldás minden egyes `Tick` eseményre csökkenti `counter` értékét, egészen addig, amíg el nem éri a 0 értéket, ilyenkor ugyanis visszaállítjuk a kezdőértékre és újra betöltjük a fájlt.

A megoldás jól szemlélteti a Windows Forms alkalmazásokban a grafikus megjelenítés tipikus mechanizmusát:

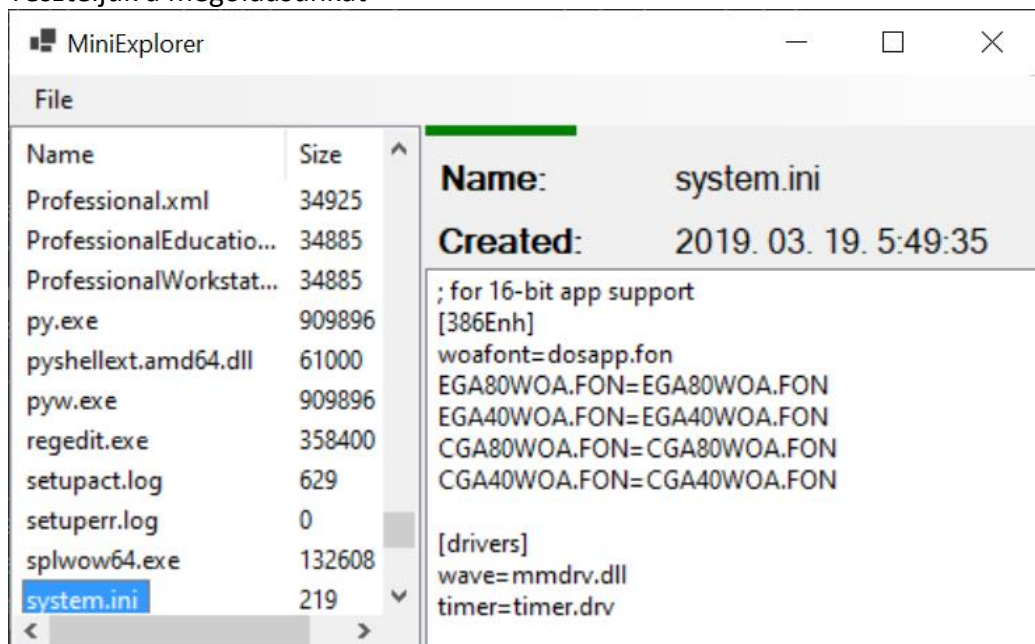
- Tényleges rajzolást az állapotot megváltoztató műveletben nem végzünk, hanem a form/vezérlő (esetünkben panel) **Invalidate** műveletének váltjuk ki a **Paint** eseményt
- A konkrét téglalap (aktuális állapotnak megfelelő) megjelenítéséért/kirajzolásáért az űrlap/vezérlő (esetünkben a panel) **Paint** eseménye felelős.

8. Iratkozzunk fel a `detailsPanel` komponens `Paint` eseményére. Ehhez a panel kijelölése után a Property Editor-ban az events fülön kattintsunk duplán a `Paint` eseményre, ezzel létrejön a kapcsolódó eseménykezelő (`detailsPanel_Paint`). Töltsük ki a kódját:

```
private void detailsPanel_Paint(object sender, PaintEventArgs e)
{
    if (loadedFile!=null)
        e.Graphics.FillRectangle(<paraméterek>);
    // A téglalap szélessége a téglalap kezdőhosszúságából (adott
    // a feladatkiírásban) számítható,
    // szorozva a számláló aktuális és max értékének arányával
}
```

A `FillRectangle` pontos paraméterezést a példakódban szereplő segítség alapján tudod meghatározni.

9. Teszteljük a megoldásunkat



Emlékeztető

Lényeges, hogy a beadott megoldások mellé külön indoklást, illetve leírást nem várunk el, ugyanakkor kérjük, hogy a beadott kódban a feladat megoldása szempontjából relevánsabb részek **kommentekkel legyenek ellátva**. Ezen felül ne felejtse el beadáskor a **neptun.txt**-t is kitölteni.

Opcionális plusz feladat – 3 iMsc pontért

Feladat

Egészítsük ki az alkalmazásunkat úgy, hogy fájlok közt „Total Commander”-szerűen tudjunk mozogni, vagyis:

- A listában jelenjenek meg a mappák nevei is. Ezekre duplán kattintva a teljes fájl lista cserélődjön le az aktuális mappa tartalmára. A mappanevek eredeti formájukban jelenjenek meg (pl. ne legyenek körbevételre szögletes vagy egyéb zárójelekkel).
- A lista elejére kerüljön be egy speciális „..” nevű elem, mely mindig az aktuális mappa szülőmappájának tartalmát listázza ki.
- Amikor gyökérelemben vagyunk (pl.: „C:\”), ne jelenjen meg a „..” elem