PROJEKTLABOR BEADANDÓ

TARTALOM

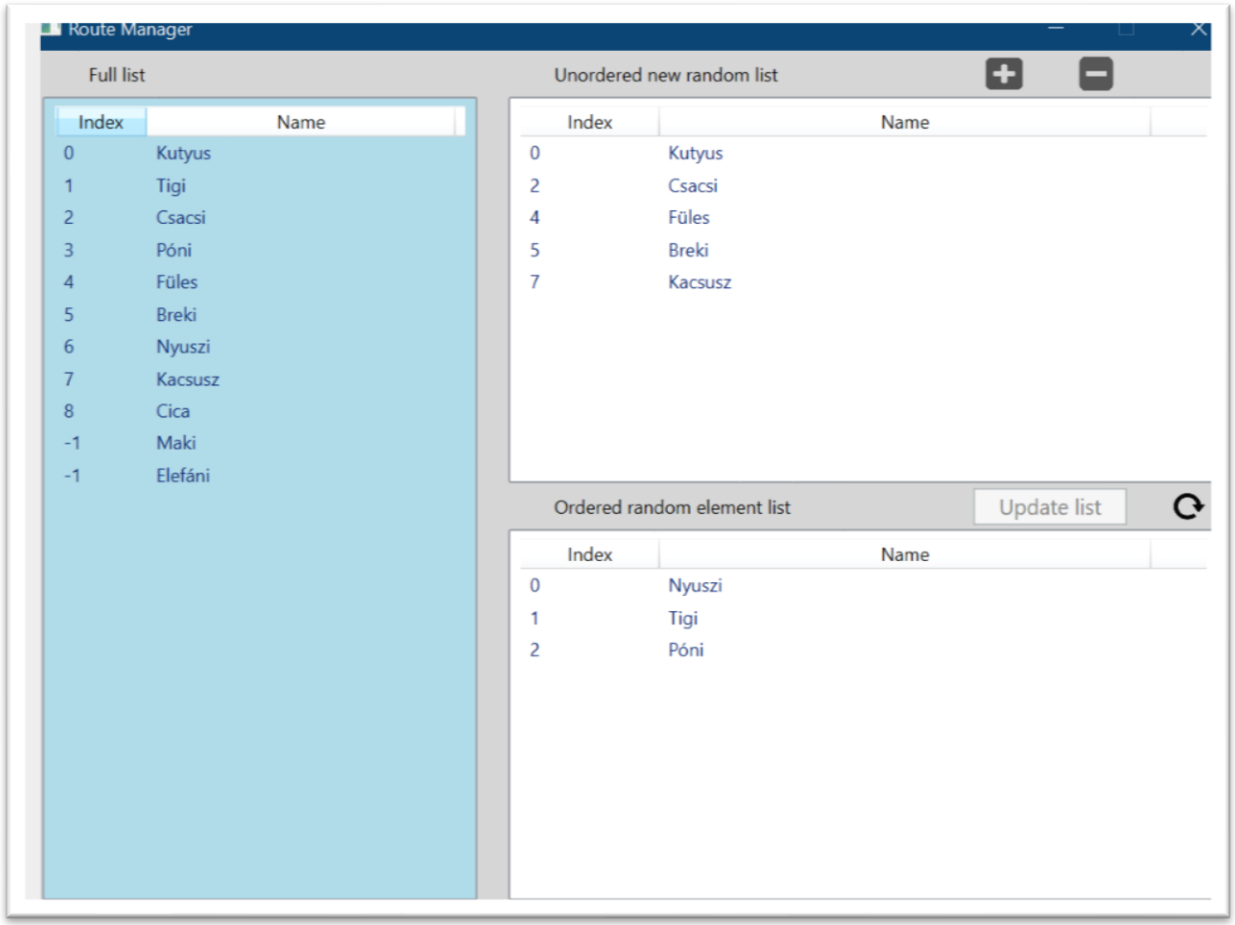
PROGRAM BEMUTATÁSA ÉS MŰKÖDÉSE

A feladat elkészítéséhez a Visual Studio IDE-t használtam fel, hogy a WPF technológia egyszerű kezdő szintű funkcióit kihasználhassam az információ jól látható megjelenítéséhez. A program a feladathoz mérten képes egy nagy rendezésre váró lista megjelenítésére, véletlenszerű mennyiségű és sorszámú elemeket tartalmazó randomizált lista feltöltésére.

A randomizált listából 2 gomb (+ és – jel) használatával tudjuk a jobb alsó szegmensben elhelyezett rendezett, a randomizált lista elemeiből álló listát feltölteni. A minusz gomb esetén az adott elemet törölhetjük az összehasonlításból. A rendezett lista feltöltése közben, valamint feltöltése után is lehetőségünk van visszavonni minden rendezést és módosítást a két kisebb listában a „reset" jellel. A gomb megnyomásakor a kis rendezett lista tartalma törlődik, a randomizált lista pedig újra megkapja eredeti elemeit.

Az „update list" gombra kattintva megtörténik az elemek összehasonlítása, és a köztük kialakult relációk alapján a teljes listába való beillesztés.

A beillesztés után a program újraindítja a rendezést, és egy véletlen elemszámú részintervallumot hoz létre a teljes lista elemeiből a jobb felső ListBox tartalmaként újbóli rendezéshez.



ALGORITMUS

Az algoritmust a RouteManager osztály tartalmazza. Ez az osztály felelős a Listák kezeléséért, valamint a modeljét biztostja a view megjelenítési rétegének. Legfontosabb része tehát az algoritmust tartalmazó updateFullList() metódus. Akkor fut le, amikor a felhasználó rákattint az „update list" gombra.

```
bool madeChange = false;
for (int i = 0; i < orderedRandomListBox.Count - 1; i++)
{
    SingleElement firstOrdered = orderedRandomListBox[i];
    SingleElement lastOrdered = orderedRandomListBox[i + 1];
    SingleElement firstFull = null, lastFull = null;
    int index = 0;
    while(firstFull == null || lastFull == null)
    {
        if(firstFull == null)
        {
            firstFull = fullListBox[index].getElementByName(firstOrdered.Name);
        }
        if(lastFull == null)
        {
            lastFull = fullListBox[index].getElementByName(lastOrdered.Name);
        }
        index++;

        if (lastFull == null && index > orderedRandomListBox.Count)
        {
            lastFull = lastOrdered;
            lastFull.Index = orderedRandomListBox.Count + 1;
        }
    }
```

Párosával végighalad a létrejövő rendezett részhalmaz elemein, ahol első lépésként megfeleltetéseket hajt végre, hogy a teljes lista tartalmával tudjunk dolgozni. Ha esetleg nem találná az utolsó elemet, akkor lastFull értékének egy olyan magas indexet ad, mellyel biztosan utolsóként kerül majd rá a sor.

```
if(firstFull.Index == -1)
{
    indexManager.insertAfterFirstDefined(firstFull);
    madeChange = true;
    i--;
}
else if (lastFull.Index == -1)
{
    indexManager.insertAfterElement(firstFull, lastFull);
    madeChange = true;
}
else if (lastFull.isEqual(firstFull))
{
    //MessageBox.Show(firstFull + "'s index is equal to " + lastI
    indexManager.insertAfterFirstDefined(firstFull);
    madeChange = true;
}
else if (lastFull.Index != -1 && firstFull.isLater(lastFull))
{
    indexManager.swapElement(firstFull, lastFull);
    madeChange = true;
}
```

Ahogy sikerült azonosítani a sorrendben szereplő elemeket, ellenőrizzük azok relációit. Ha az első elem sincs rendezve, akkor biztosak lehetünk benne, hogy bekerül az első még be nem töltött helyre: (insertAfterFirstDefined). Ha a második elem indexe -1, akkor biztosan az előtte levő elem után fog következni (insertAfterElement), és nem kell más műveletet végezni, mivel még nem szerepelt a lista rendezett részén. Ha a két elem megegyezik, akkor szintén a rendezetlen részen járhatunk, ahol -1 minden elem indexe, ezért csak beillesztjük az első lehetséges helyre (insertAfterFirstDefined). Ha mindkét elem rendezve van, mégis fordított sorrendben vannak, abban az esetben csere történik, ahol a lastFull objektumot a firstFull objektum teljes listában elfoglalt helyére rakjuk, és az alattuk levő listaelemeket mind eggyel lejjebb csúsztatjuk (swapElement).

```
else if (firstFull.isLater(lastFull))
{
    indexManager.insertAfterElement(firstFull, lastFull);
    madeChange = true;
}
listToBeSorted();
```

Az utolsó eset az, amikor az utolsó elemünk még nincs rendezve, és csak ezért tűnik úgy, mintha a listában előrébb lenne mint a firstFull objektumunk (-1 kisebb mint minden a teljes listában helyet foglaló elem indexe). Ebben az esetben a már rendezett elemünk mögé helyezzük az eddig nem rendezett második elemet (insertAfterElement).

A listToBeSorted() metódus pedig az indexek updatelésére szolgál, hogy minden egyes rendezés után a helyes indexek szerepeljenek a teljes listában.

A madeChange boolean érék az utolsó fontos alkotóeleme a rendezési algoritmusnak. Annyi a feladata, hogy jelezze minden ellenőrzés alkalmával azt, hogy szükségünk volt-e rendezésre ahhoz, hogy a részhalmaz elemeivel azonos sorrendet állítsunk elő a teljes listában. Ha igen, akkor a madeChange igazra vált, ami az updateFullList() metódus újraindítását jelenti, miután egy rendezési ciklus lefutott. Így biztosak lehetünk abban, hogy csak akkor lép ki a rendezésből a program, ha már nincs további rendezés, melyet eszközölnie kéne.

ROUTEMANAGER

Az itt található ObservableCollection<SingleElement> listák szolgálnak az egyes ListBox-ok ItemSource-aként, az egész osztály pedig a DataContext. A listákon kívül egy kinézetben létrehozott „update list" gombbal is összekötöttem a RouteManager-ben létrehozott Button példányt. A binding segítségével a kódban történő változás látszik a UI gombján is, amit itt csak arra használtam, hogy aktívvá és inaktívvá változtassam az „update list" gombot.

A RouteManager ezenfelül a UI felületen történő gombnyomások eventjeit kezeli, az elemek hozzáadásához, a lista „reset"-eléséhez stb. Egy IsortListet is implementál az osztály, mely egy listenerként viselkedik, hogy az IndexManager változtatásairól értesíteni tudja a RouteManagert az indexelés frissítéséhez.

ISortList listToBeSorted() metódus:

```
6 references
public ObservableCollection<SingleElement> listToBeSorted()
{
    int index = 0;
    foreach (SingleElement element in fullListBox)
    {
        if(element.Index != -1)
        {
            element.Index = index++;
        }
    }
    ArrayList elements = new ArrayList(fullListBox.ToArray());
    elements.Sort();
    fullListBox.Clear();
    foreach (SingleElement element in elements)
    {
        fullListBox.Add(element);
    }
    return fullListBox;
}
```

Példa a UI eventek kezelésére:

```
1 reference
internal void insertElement(SingleElement selectedItem)
{
    if(selectedItem != null)
    {
        unorderedRandomListBox.Remove(selectedItem);
        SingleElement newItem = new SingleElement(selectedItem.Name, orderedRandomListBox.Count);
        orderedRandomListBox.Add(newItem);
        if (unorderedRandomListBox.Count == 0)
        {
            canUpdate.IsEnabled = true;
        }
    }
}

1 reference
internal void removeElement(SingleElement selectedItem)
{
    unorderedRandomListBox.Remove(selectedItem);
    if (unorderedRandomListBox.Count == 0)
    {
        canUpdate.IsEnabled = true;
    }
}
```

mainwindow.xaml.cs ahol az eventek küldése történik:

```
private void BtnStartOrdering_Click(object sender, RoutedEventArgs e)
{
    manager.updateFullList();
    manager.orderedRandomListBox.Clear();
    manager.continueDestinationProcess();
}

1 reference
private void BtnAddElement_Click(object sender, RoutedEventArgs e)
{
    SingleElement selectedItem = (SingleElement)listUnorderedRandomElements.SelectedItem;
    if(selectedItem != null)
    {
        manager.insertElement(selectedItem);
    }
}

1 reference
private void BtnRemoveElement_Click(object sender, RoutedEventArgs e)
{
    SingleElement selectedItem = (SingleElement)listUnorderedRandomElements.SelectedItem;
    if (selectedItem != null)
    {
        manager.removeElement(selectedItem);
    }
}
```

SINGLEELEMENT

A teszteléshez készítettem egy alapvető nagy listát, rendezetlen elemekkel, különböző (nem általam!) kitalált állatnevekkel. A SingleElement osztály reprezentál egy-egy elemet, ahol a toString() metódust írtam át annak érdekében, hogy a ListBoxokban jól megjeleníthető adatot kapjak. Az osztály ezenkívül összehasonlítást tud végrehajtani elemein, valamint ellenőrzéseket. Ezen példányokat használom fel az ObservableCollection<> és List<> objektumoknál elemekként.

IComparable megvalósításához szükséges metódus:

```
0 references
public int CompareTo(object obj)
{
    if (obj == null) return 1;
    SingleElement otherElement = obj as SingleElement;
    if(this.index == -1)
    {
        return 1;
    }
    if (otherElement != null)
        return this.index.CompareTo(otherElement.index);
    else
    {
        throw new ArgumentException("Object is not a SingleElement!");
    }
}
```

```
2 references
public SingleElement getElementByName(string name)
{
    if(this.name == name)
    {
        return this;
    }
    return null;
}

2 references
protected void OnPropertyChanged(string name)
{
    PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));
}

1 reference
internal bool isEqual(SingleElement firstFull)
{
    if(index == firstFull.index)
    {
        return true;
    }
    return false;
}
```

INDEXMANAGER

A különböző listák elemein hajtja végre a kívánt választásokat. Itt történik az elemek beillesztése, cseréje, valamint a randomizált listák és a teljes lista első feltöltése. Implementálja az ISortList interface-t, amit minden rendezés után meghív.

Példa műveletek:

```
1 reference
internal void swapElement(SingleElement firstFull, SingleElement lastFull)
{
    //MessageBox.Show("swapElement" + firstFull + " || " + lastFull);
    for (int i = 0; i < list.Count; i++)
    {
        if (list[i].Name.Equals(lastFull.Name))
        {
            list.RemoveAt(i);
        }
    }
    for (int i = 0; i < list.Count; i++)
    {
        if (list[i].Name.Equals(firstFull.Name))
        {
            list.RemoveAt(i);
        }
    }
    list.Insert(lastFull.Index, firstFull);
    list.Insert(lastFull.Index + 1, lastFull);
    list = sortList.listToBeSorted();
}
```

```
1 reference
internal static ObservableCollection<SingleElement> shuffleNewElements(ObservableCollection<SingleElement:
{
    ObservableCollection<SingleElement> randomList = new ObservableCollection<SingleElement>();
    Random random = new Random();
    int capacity = random.Next(2, fullListBox.Count);
    int sortRate = random.Next(1, fullListBox.Count/2);
    for (int i = 0; i < fullListBox.Count; i++)
    {
        if( (i%sortRate) == 0 && randomList.Count < capacity)
        {
            randomList.Add(fullListBox[i]);
        }
    }
    return randomList;
}
```

```
2 references
internal void insertAfterElement(SingleElement firstFull, SingleElement lastFull)
{
    //MessageBox.Show("insertAfterElement" + firstFull + " || " + lastFull);
    int index = -1;
    for (int i = 0; i < list.Count; i++)
    {
        if (list[i].Name.Equals(lastFull.Name))
        {
            index = i;
        }
    }
    list.RemoveAt(index);
    lastFull.Index = firstFull.Index + 1;
    list.Insert(lastFull.Index, lastFull);
    list = sortList.listToBeSorted();
}
```

KÖSZÖNÖM A FIGYELMET!