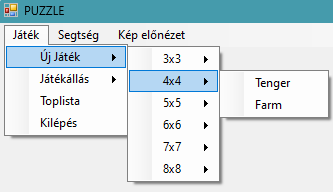
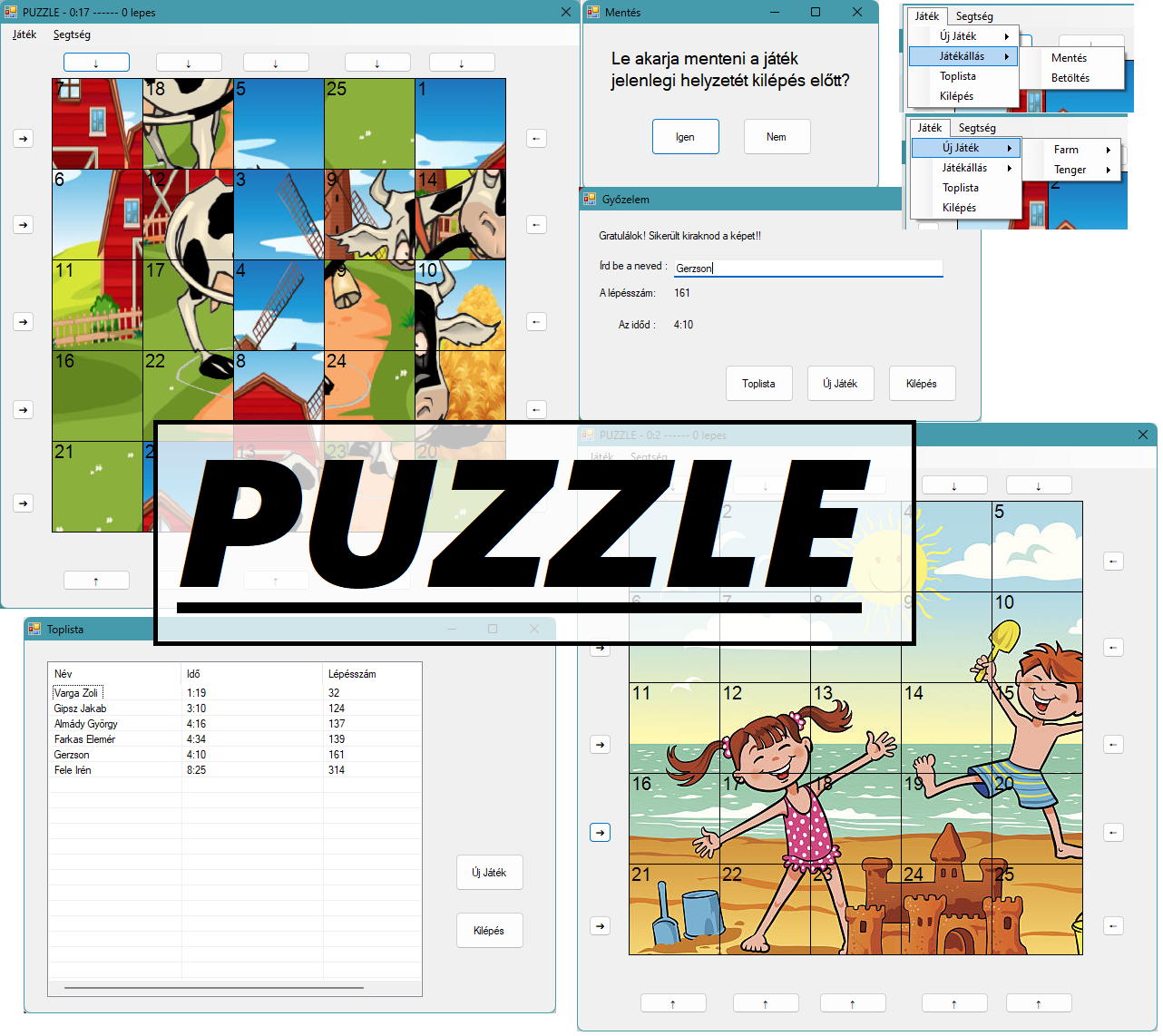
**Lucrare de atestat la Informatică**

**Liceul Teoretic “Ady Endre”, Oradea**

# Tartalomjegyzék



**2023**

|  |  |  |
| --- | --- | --- |
| **Prof. Coordonator:** |  | **Autor:** |
| **Kotró László Lehel** |  | **Kulcsár Gerzson** |

[Tartalomjegyzék 1](#_Toc134570851)

[Bevezetés 3](#_Toc134570852)

[Szoftver és hardver követelmények 4](#_Toc134570853)

[A dolgozat leírása 5](#_Toc134570854)

[A program felépítése 5](#_Toc134570855)

[frmJatek 5](#_Toc134570856)

[frmNyert (fny) 6](#_Toc134570857)

[frmToplista (ft) 6](#_Toc134570858)

[frmMent (fm) 7](#_Toc134570859)

[frmElonezet (fe) 7](#_Toc134570860)

[A pálya 8](#_Toc134570861)

[Függvények 9](#_Toc134570862)

[le 9](#_Toc134570863)

[fel 9](#_Toc134570864)

[jobb 10](#_Toc134570865)

[bal 10](#_Toc134570866)

[keveres 11](#_Toc134570867)

[mentes 12](#_Toc134570868)

[betolt 12](#_Toc134570869)

[kiir 12](#_Toc134570870)

[rendezes 13](#_Toc134570871)

[nyert 13](#_Toc134570872)

[gyozelem 14](#_Toc134570873)

[gomblathato 15](#_Toc134570874)

[Felhasználási dokumentáció 16](#_Toc134570875)

[A játék elindítása 16](#_Toc134570876)

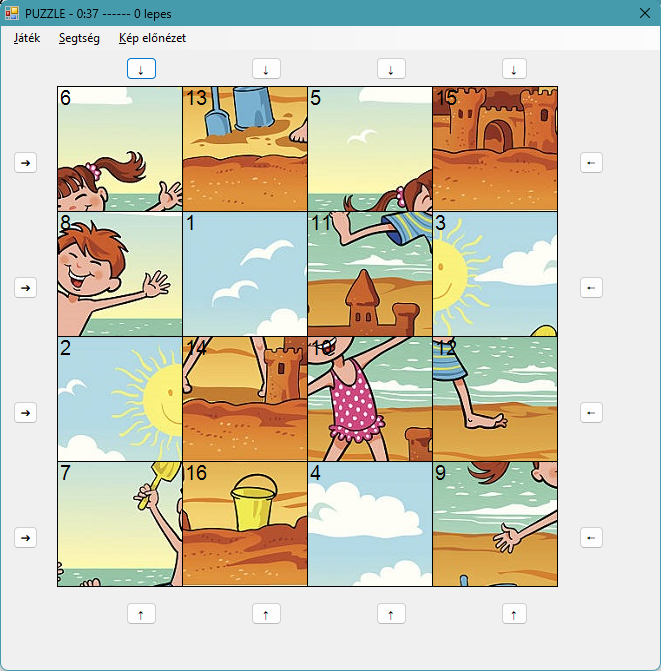
[A kép kirakásának mechanikai leírása 16](#_Toc134570877)

[A kép kirakása útmutató 17](#_Toc134570878)

[Bibliográfia 19](#_Toc134570879)

# Bevezetés

A program, amit készítettem egy kevésbé szokványos Puzzle játék, ahol interaktívan, logikánkat és eszünket próbára téve célunk kirakni a megadott képet.

Először is még gyerekként, megismerkedve a Puzzle játékok világával, mindig szerettem ezzel időt tölteni. Koromnak megfelelően egyre kisebb- és több-darabszámú Puzzle játékon törtem a fejem, viszont egy idő után unalmassá váltak, mivel nem volt szükséges a racionális gondolkozást, logikát használni kirakásukhoz. Megismerkedtem többek között a Rubik kockával, a bűvös hengerrel, valamint a tologatós Puzzle játékokkal. Ezeket a tologatós Puzzle játékokat a gombos telefonokon ismertem meg, amelyekkel rengeteg időt töltöttem, míg igazi mesterévé váltam. Ezekben a játékokban mindig egy kocka üresen marad, és ennek helyére új kockákat lehetett tologatni, ezzel alakítva a képkockákat a kép kirakása érdekében.

Különböző, de erősen az előbbire emlékeztető mechanikával és kirakási technikával rendelkezik a görgetős Puzzle játék, az, amelyet én is készítettem. Ebben a játékban egyszerre egy egész sort, illetve egy oszlopot tudunk eltolni, azaz „görgetni”, ami következtében mindegyik adott képkocka a megjelölt irányba eltolódik egy egységgel, az utolsó pedig az első helyére kerül. A program felhasználói kezelőfelülete a következőképpen néz ki: minden oszlop felett, illetve alatt és minden sor mellett és a megfelelő irányba mutató, görgető gombbal van ellátva. A kép kirakásában segítséget nyújt a képkockák bal felső sarkában lévő fekete sorszámozás, ami a bal felső saroktól indulva, egyesével növekszik jobbra irányban. A helyes sorrend kirakása után a kép is készen lesz, ezzel sikeresen befejezve a játékot.

Egyéni megvalósításként megnevezném a kreatív kirakható képeket, melyeket én személyesen szerkesztettem össze különböző interneten talált fotókból, ezáltal teljesen egyedi, máshol nem felkereshető képeket rakhat ki a játékos. Továbbá minden függvényt és programrészlet saját munka.

# Szoftver és hardver követelmények

Az általam használt szoftverfejlesztői környezet a programom megírására a Visual C# 2010 Express volt.

A Visual C# 2010 Express egy Microsoft által fejlesztett szoftverfejlesztő környezet (IDE) C# programozási nyelvhez. A szoftver minimális és ajánlott rendszerkövetelményei a következők:

Minimális rendszerkövetelmények:

Operációs rendszer: Windows XP Service Pack 3, Windows Vista Service Pack 2, Windows 7, Windows Server 2003 Service Pack 2, Windows Server 2008 Service Pack 2, vagy Windows Server 2008 R2

Processzor: 1.6 GHz-es vagy gyorsabb processzor

Memória: 1 GB RAM (32 bites rendszerekhez) vagy 2 GB RAM (64 bites rendszerekhez)

Tárhely: 3 GB szabad merevlemez-terület

Grafikus kártya: DirectX 9-kompatibilis videokártya 1024 x 768-as felbontással

Ajánlott rendszerkövetelmények:

Operációs rendszer: Windows 7 vagy Windows Server 2008 R2

Processzor: 2.4 GHz-es vagy gyorsabb processzor

Memória: 2 GB RAM (32 bites rendszerekhez) vagy 4 GB RAM (64 bites rendszerekhez)

Tárhely: 4 GB szabad merevlemez-terület

Grafikus kártya: DirectX 10-kompatibilis videokártya 1024 x 768-as felbontással

Saját rendszertulajdonságok:

Operációs rendszer: Windows 11 Pro Insider Preview

Processzor: AMD Ryzen 7 4700U with Radeon Graphics 2.00 GHz

Memória: 16.0 GB RAM (64 bites rendszer)

Tárhely: 64 GB szabad merevlemez-terület

Grafikus kártya: AMD Radeon Graphics (1920 x 1080-as felbontással)

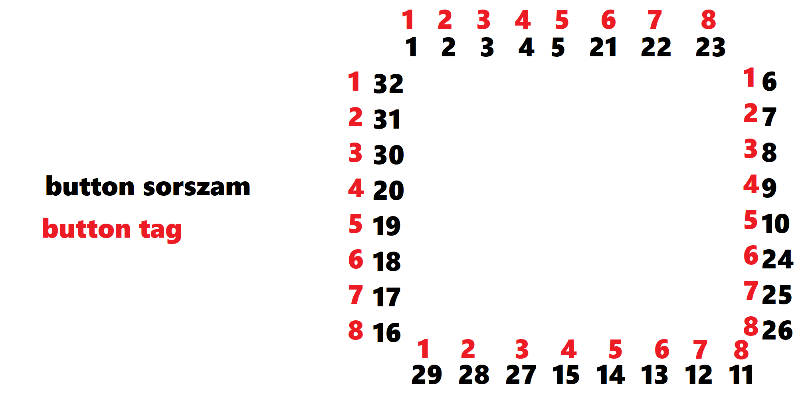
# A dolgozat leírása

## A program felépítése

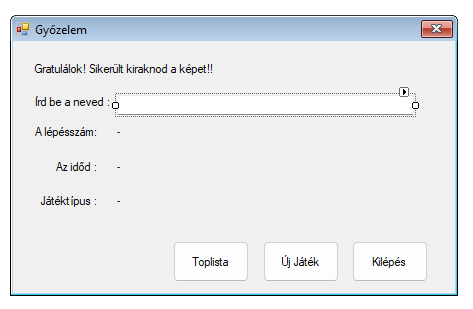
A program öt különálló Form-ból épül fel: frmJatek, frmNyert, frmToplista, frmMent, frmElonezet. A program fő futási ablaka a frmJatek, ez a játék, a többi ablak csak a program kiegészítésére szolgál.

### frmJatek

A Puzzle játék futtatásakor megnyíló ablak, amelyben tudunk játszani. Fő alkotó eleme egy *PictureBox* és négy sor, nyolc darab iránymutató *button,* amelyek alkotják a játék kezelőfelületét. Ez a *form* továbbá tartalmaz még egy *menuStrip1* nevű menüsort, *Játék*, *Segítség* és *Kép előnézet* gombokkal ellátva. A *Játék* menüben tudjuk elindítani a játékot, szabadon választható *n x n* -es felosztásban és tetszés szerinti képpel. Itt tudunk lementeni egy játékállást, amennyiben később, akár a program bezárása után is, bármikor szeretnénk azt folytatni, erre ugyanitt található betöltő gomb is. A menüből megnyitható a toplista, illetve ki is tudunk lépni a programból. A *Segítség* menüpont tartalmaz egy játékleírást, a *Kép előnézet* pedig megmutatja a két választható kép opciót. Az ablak tartalmaz egy *timer*-t, amely a játék futási idejét méri, ez a *form* címsorába van írva a lépésszámhoz hasonlóan.

A programnak ez a *form*-ja számtalan függvényt tartalmaz, melynek nagyrésze gombnyomásra vonatkoznak, vagy egyéb design-elemek interakciójaként, de van számolásra és fájlkezelésre szánt függvény is. Ezeket a függvényeket később bemutatom részletesen.

### frmNyert (fny)

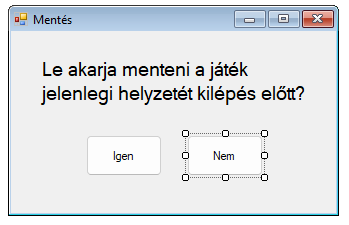
Ez az ablak akkor jelenik meg ha a játékos sikeresen teljesítette a Puzzle-t, tehát nyert. Ez a *form* áll nyolc darab *label* cellából, melyek közül az *5*, *6*, és *8* számúak adatfeltöltés céljából jöttek létre, míg a többi informatív jellegű, konstans szöveget tárolnak. Ezen kívül van egy *textBox* rajta, ahova a játékos saját nevét írhatja be, ami a toplistázás miatt fontos. Található még itt három gomb, az egyik egyenesen a toplistához vezet, a másik visszavezet a játékfelületre, a harmadik pedig kilép a programból.

### frmToplista (ft)

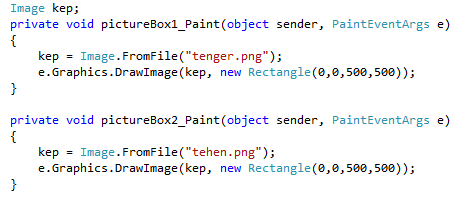
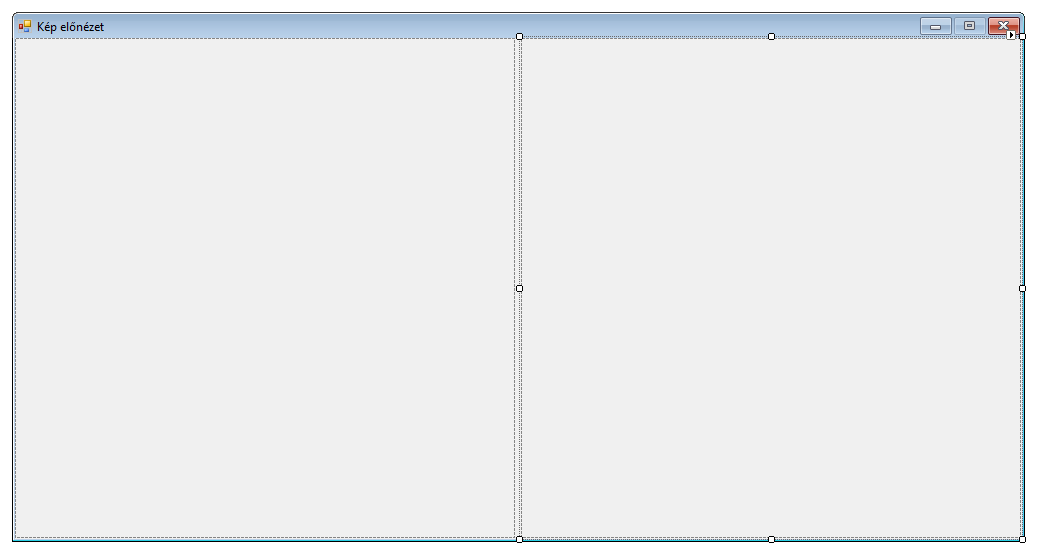
Ez a *form* gomb által megnyitható, akár a *frmJáték* menüsorából, akár egy sikeres játék utáni *frmNyert* ablakából. Fő alkotóeleme egy *listView* táblázat, melyben a *Név, Idő, Lépésszám* és *Játéktípus* tulajdonságok láthatóak, a program futása során kitöltve a toplistás játékosok adataival.

A kitöltésre szolgáló összes függvény és parancs a *frmJáték* nevű fő *form*-on belül található, amelyeket később bővebben is bemutatok majd. Ezen kívül, található rajta még a már ismert *Új Játék* és *Kilépés* gombok is.

### frmMent (fm)

Ez a *form* egy kifejezetten egyszerű ablak, ami egyetlen kérdésre vár választ a játékostól. Található rajta egy *label* és két *button*, a cellában a kérdést olvashatjuk, a két gomb segítségével pedig válaszolhatunk beleegyezően vagy nemlegesen. Akkor jelenik meg, ha a játékos egy Puzzle kirakása közben, amíg nem fejezte be azt, ki szeretne lépni a játékból bármilyen formában. Akár a frmJáték *menuStrip*-jében található menüponttal, akár a jobb felső sarokban található piros *”X”* bezáró gombbal, akár más billentyűkombinációval (*Alt + F4*) próbálkozik a játékos, össze fog találkozni ezzel az ablakkal, ha képernyőjén lévő Puzzle-t összekeverten hagyta. A lementés során egy *”ment.txt”* nevű fájlban eltárolja a program a Puzzle állását, a későbbi folytatás érdekében.

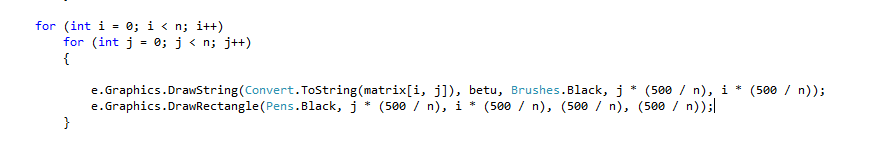
### frmElonezet (fe)

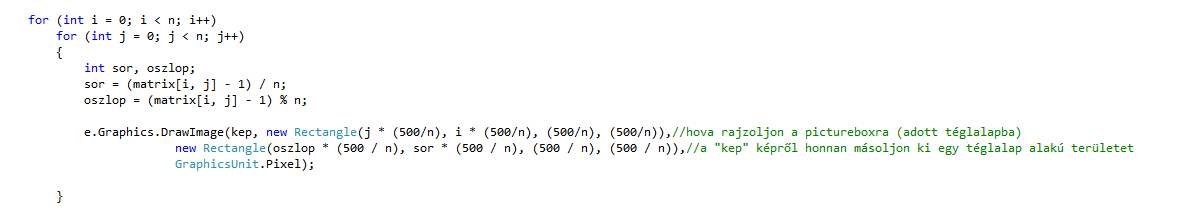


Ez az ablak a két kirakható kép előnézetét mutatja meg a játékosoknak, amelyet az egész *form*-ot kitöltő két 500 x 500 -as *PictureBox*-ban jelenít meg.  
A mellék *form*-ok közül ez  
az egyetlen amelyik önálló programrészletet tartalmaz, ebben a két PictureBox-ra rajzolja ki a két választható képet a mellékleten látható módon relatív hivatkozással:  
Ezt az ablakot meg tudjunk nyitni a menüsorból akár játék közben segíthet a kirakásban vagy a játék előtti képkiválasztásban.

## A pálya

A játék pályájának virtuális kirajzolása egy *Picturebox* nevű grafikus képmezőben történik meg a következőképpen:

A *Picturebox* négyzethálókba van felosztva, a választott *n x n* -es elrendezésben, mindegyik négyzet bal felső sarkában megszámozva a kép kirakását segítve. Ez a programrészlet itt látható: 

Ebben a grafikus mezőben a négyzetekhez hozzá van rendelve a kép adott része, úgymond felszeletelve, lehetővé téve ennek különálló mozgását. Fontos ugyanakkor, hogy egy általánosított,  
*n x n* -es formula legyen használva, a játék megfelelő működése érdekében, bármilyen dimenzióval rendelkező Puzzle játékot választunk is ki. Ennek kivitelezése a következőképpen történt: 

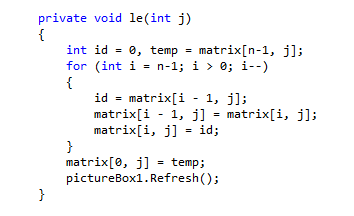
Mivel a *Picturebox* egy *500 x 500* -as dimenziókkal rendelkezik, egy *n x n* -es négyzethálóhoz tartozó négyzet mindegyik oldalának hossza *500/n*. Ez az *e.Grapics.DrawImage* függvény egyszerre mindig csak egy négyzetet rajzol ki, a dupla *for* ciklus segítségével pedig eljut az egész kép végleges kirajzolásához.

## Függvények

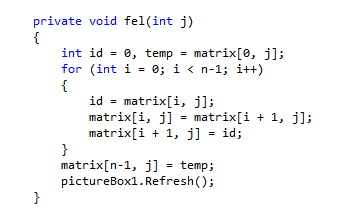
A függvények leírása során minden fontos függvényt megemlítek, melyeknél részletesen tárgyalom működésük leírását, néha bemutatva egy-egy bonyolultabb programrészletet is.

A játék működésének alapfüggvényei a sorok és oszlopok mozgatását foglalják magukba.

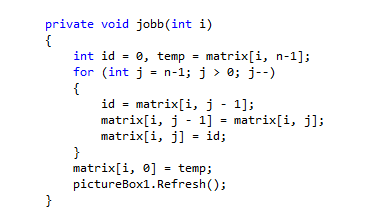
### le

A *le* függvény paraméterként *(int j)* megkapja a „hanyadik oszlop” értéket, és egy cserét hajt végre az *id* és *temp* segédváltozók segítségével. Az utolsó  
tagot *(matrix[ n-1, j])* tárolja  
a *temp* *változóban*, majd minden  
tagot kicserél az felette lévővel  
*matrix [i , j] << matrix [i-1 , j])*. A legelső tag *(matrix [0 , j])* megkapja a legutolsó értékét a *temp* változó értékeként.

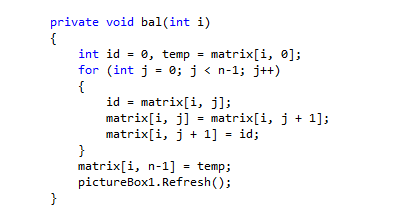
### fel

A *fel* függvény paraméterként *(int j)* megkapja a „hanyadik oszlop” értéket, és egy cserét hajt végre az *id* és *temp*segédváltozók segítségével. Az első   
tagot *(matrix[ 0 , j])* tárolja a  
*temp* *változóban*, majd minden tagot  
kicserél az alatta lévővel  
*matrix [i+1 , j] << matrix [i , j])*. A legutolsó tag *(matrix [n-1 , j])* megkapja a legelső értékét a *temp* változó értékeként.

### jobb

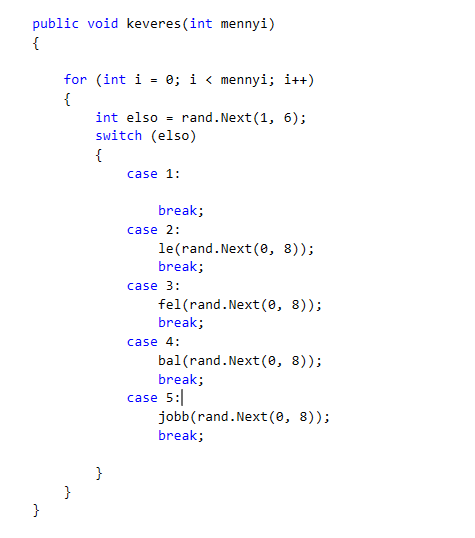
A *jobb* függvény paraméterként *(int i)* megkapja a „hanyadik sor” értéket, és egy cserét hajt végre az *id* és *temp*  
segédváltozók segítségével.  
Az utolsó tagot  
*(matrix[ i , n-1])* tárolja a *temp* változóban, majd minden tagot kicserél a  
mellette jobbra lévővel  
*matrix [i , j] << matrix [i , j-1]).* A legelső tag *(matrix [i , 0])* megkapja a legutolsó értékét a *temp* változó értékeként.

### bal

A *bal* függvény paraméterként *(int i)* megkapja a „hanyadik sor” értéket, és egy cserét hajt végre az *id* és *temp*  
segédváltozók segítségével. Az első tagot *(matrix[ i , 0])* tárolja a *temp* változóban, majd minden tagot kicserél a  
mellette balra lévővel  
*(matrix [i , j+1] << matrix [i , j]).* A legutolsó tag *(matrix [i , n-1])* megkapja a legutolsó értékét a *temp* változó értékeként.

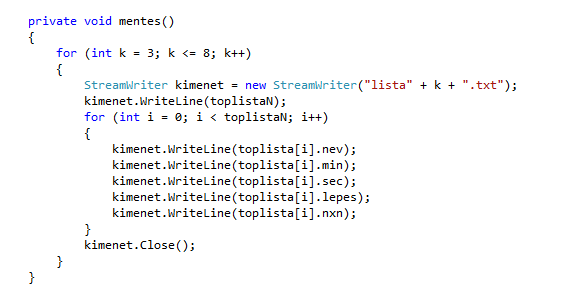
A játékban lévő kép összekeveréséhez használt függvény a *keveres.*

### keveres

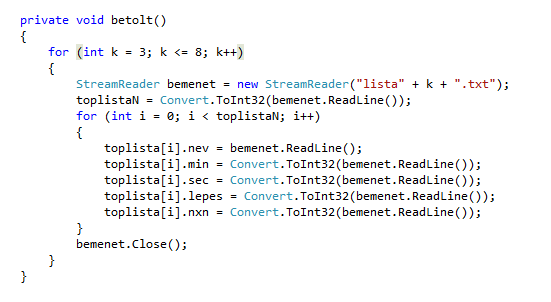
 A *keveres* függvény a játékot megkeveri, azaz a játék mátrixát, ami alapján van felosztva a kép. Fontos, hogy a keverés nem egyedi random értékeket ad a mátrixnak, hanem az irányfüggvények meghívásával keveri meg a játékot. Kap egy *mennyi* nevű változót, ami a keverés lépésszámát fogja jelenteni, aztán generál egy  
random számot *1-től 5-ig (rand.Next(1,6))*, majd a *random szám* meghatározza, hogy melyik irányban fog történni az elmozdulás, és a *függvény argumentumában* mégegy random számot generál egy másik *random számot*, amelyik a *hanyadik sor/oszlop*-ot határozza meg.

A játék toplistájára vonatkozó függvények a *mentes*, a *betolt*, a *kiir*, és a *rendezes* függvények.

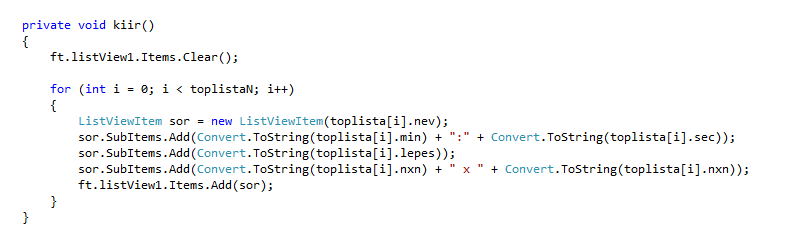
### mentes

 A *mentes* függvény a toplista jelenlegi értékeit menti ki egy szöveges dokumentumba, aminek neve áll a „*lista*” szóból, követve az *n x n* -es puzzle „*n*” számával, majd a kiterjesztés „*.txt*”. Külön, minden *n x n* -es játékmódnak saját, névreszóló toplistája van, ezzel megkönnyítve a rendezést.

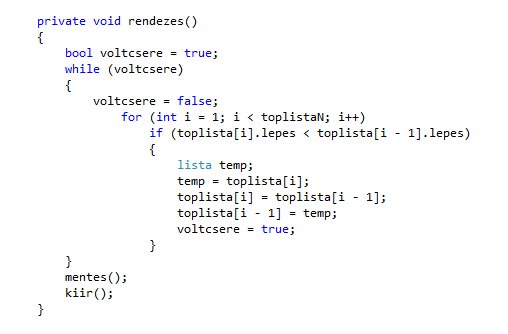
### betolt

 A *betolt* függvény a *toplista* *txt*-ből olvassa ki a benne található értékeket, előbb a *3x3*-as toplista fájlból majd sorban, egyesével, egészen *8x8*-ig. A szöveges dokumentum első sorában megtalálható a számláló *(toplistaN)*, aztán *név, perc, másodperc, lépés, n x n* sorrendben tölti fel a toplista nevű struktúrát az adatokkal.

### kiir

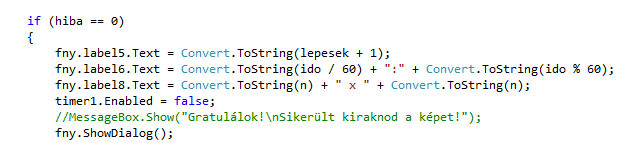
 A *kiir*  függvény a *toplista* struktúrában található elemeket írja ki a *frmToplista* *form* *ListViewItem* táblázatába.

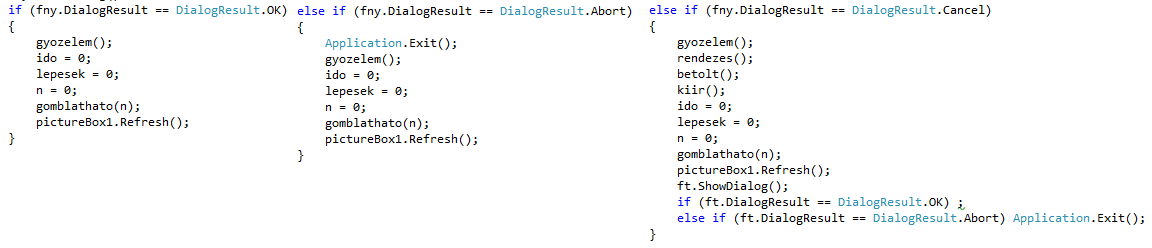
### rendezes

 A *rendezes* függvény a *toplista* *struct*-ban lévő elemeket rendezi *lépésszám* szerint. Fontos megjegyezni, hogy nem kell külön *nevet, percet*, stb cserélni, hanem a *toplista[i]* a *toplista* összes *i*-edik értékére vonatkozik. Itt tesz fontos különbséget a beolvasás során *n x n* -es sorrend, mert így már *n x n* szerint sorrendben vannak, és elég csak a lépések számát összehasonlítani.

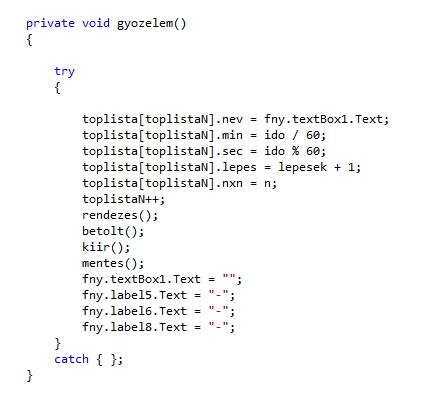
A játék megnyerésére vonatkozó függvények a *nyert* és a *győzelem*. A nyert leteszteli hogy nyert-e a játékos, ezt minden lépésre (gombnyomásra) végrehajtja a program. A *győzelem* a *nyert* függvény teljesülése esetében van meghívva, azaz csak akkor fut le, ha tényleg győzött a játékos.

### nyert

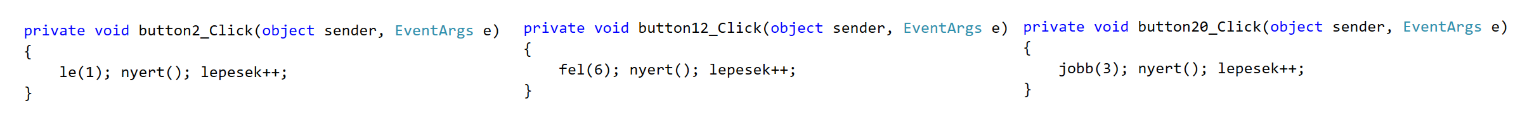
A *nyert* függvény tesztelése a következőképpen történik: megnézi  
hogy a jelenlegi elem pontosan  
eggyel kisebb-e, mint a következő  
*(matrix [i , j] != matrix [i , j+1*]) és hogy az alatta lévő elem pontosan n-el nagyobb, mint a jelenlegi *(matrix [i , j] != matrix [i , j+1]).* Amennyiben nem talált egyetlenegy hibát sem, leállítja az időzítőt, a *frmNyert* formban feltölti a megfelelő értékekkel a *label*-eket és megjeleníti azt. Az itt kapott gomb-válaszok alapján különböző akciókat hajt végre: az *OK* válasz az *Új Játék*, az *Abort* a *Kilépés* és a *Cancel* pedig a *Toplista* menüpontokat jelentik az adott esetben.



### gyozelem

 A *gyozelem* függvény feltölti a toplista struktúrába a *toplistaN*-edik játékos adatait, amit a *frmNyert* *form*-ból megtudott, ahol minden új *struct* elem után növeli a számlálót *(toplistaN)*, majd rendezi, újra tolti és kiírja az értékeket, majd végül lementi azt. Ekkor a *frmNyert* *form*-ban található mezők tartalmát kiüríti. Az egész folyamat egy *try {…} catch {};* -be van rakva az esetleges hibák elkerülése végett (például valaki nem töltötte ki a nevét, ezáltal üres mezővel feltöltve a toplistát).

A játék kezelőfelülete a PictureBox mellett, áll még a sorokat/oszlopokat mozgató gombokból. Ezeknek a száma eltérő kell legyen, minden *n x n* -es játék esetében. A form tartalmazza mind a nyolc darab gombot mind a négy irányban, viszont a *gomblathato* függvény meghatározza, hogy melyik gombok legyenek láthatóak *(button1.Visible=true/false).*

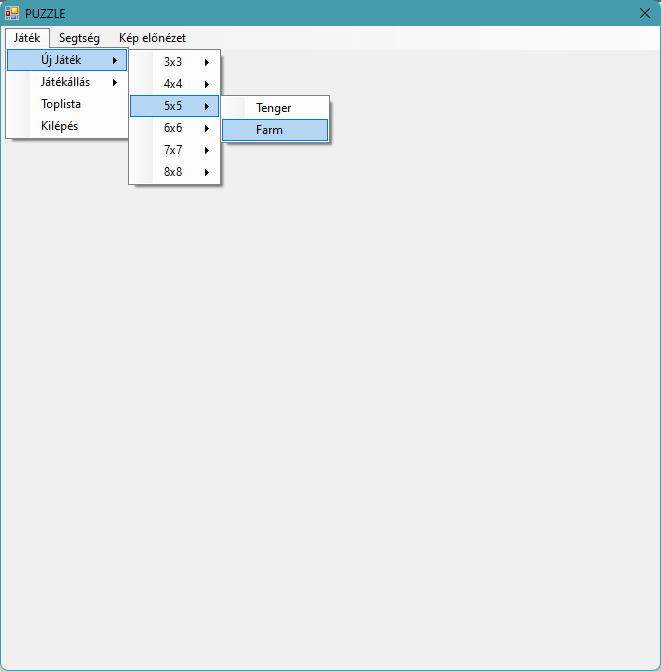
A gombok megnyomásához kötött függvények tartalmaznak egy *irányfüggvény* meghívást a megadott sorra, a *nyert* függvény meghívását a teszteléshez és a *lépésszámláló növelését*. Példa: 

### gomblathato

A függvény a gomb *visible* tulajdonságát használja, viszont mivel van 32 gomb, relatív hivatkozással beszélek egyszerre az összes gombról, *foreach* ciklus használatával. A *Design view* segítségével meghatároztam a gombok *Tag*-jét, amit az alapján töltöttem ki, hogy az adott gomb *„hány-szor-hányas”* elrendezésben kell szerepeljen. A ciklus ezáltal az összes *x Button*-ra megnézi, hogy kisebb vagy egyenlő-e az *n* értékével, majd láthatóvá teszi ezeket. Az összes többi esetet láthatatlanná teszi, viszont, hogy a játék elkezdése előtt se legyenek gombok, a program indításakor meghívom a *gomblathato(0)* függvényt. Továbbá fontos a gombok pozíciója, ezért külön kell tárgyalni *a fel, le jobb, bal* irányú gombokat, hogy minél általánosabban tudjam pozícionálni. Ezt a *button1.Location = new Point (x,y)* tulajdonság átírásával teszem, *foreach* ciklus *x* változójával, azaz *((Button)x).Location*.

# Felhasználási dokumentáció

## A játék elindítása

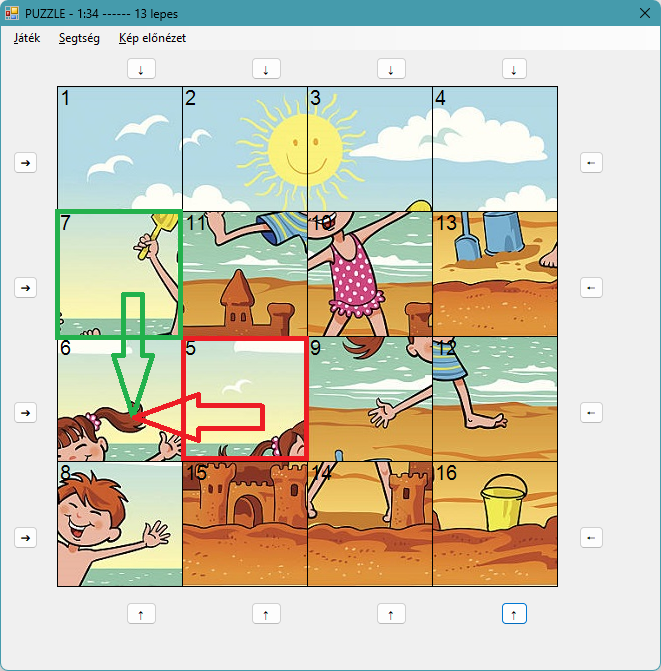
A program elindítása után egy üres képernyővel  
és egy menüsorral találkozik a felhasználó. A játék elindítása a *„Játék”* című menüpontra kattintva  
*„Új Játék”* gombbal lehetséges. Itt, ki kell választani a Puzzle felosztásának méretét, azaz, hogy *hány-szor-hány* képkockából álljon a puzzle, majd, hogy melyik képet szeretné kirakni a felhasználó.

## A kép kirakásának mechanikai leírása

A játék elindítása után a program számítja az eltelt időt és a megtett lépések számát, ezt nyomon követhetjük a címsorban. A kép megjelenik a képernyőn, szétvágva kockákba a kiválasztott felosztás szerint, megszámozva. Segítségül veheti a menüsorban található *„Kép előnézet”* opciót, ami megmutatja az eredeti képet. Mindegyik sor jobb és bal oldalán található egy-egy gomb, egy iránymutató nyíllal ellátva. A jobbra mutató nyíl megnyomása esetén, a mellette található sor minden elemét eltolja egy képkockával jobbra, anélkül, hogy változtatna a többi sor elemeinek sorrendjén, és ugyanígy a balra mutató nyíl balra. Mindegyik oszlop felett és alatt hasonlóan található egy-egy gomb nyíljelzéssel. Akárcsak az előbbiek, a hozzájuk rendelt oszlopokat mozgatják anélkül, hogy a többi oszlop változna vagy kicserélődne. A gombok hozzárendelt oszlopai amelyik pontosan alatta, illetve felette helyezkedik el. A játék mechanikája ennyiben kimerül, a gombok segítségével, a sorok és oszlopok eltolása által ki kell rakni a puzzle játékot.

## A kép kirakása útmutató

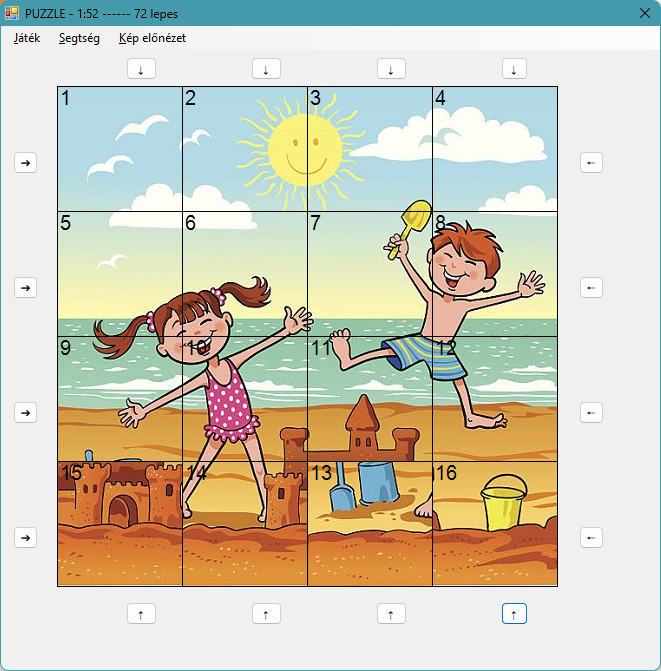
A játékot kirakásának most bemutatott módszere bármelyik sor/ oszlop sorrendben működik, az eljárás ugyanaz, csak az egyszerűség kedvéért én növekvő sorrendben fogom segíteni a kirakást.

Első lépésként, az első sort ajánlott kirakni. Ez nagyon egyszerű, semmire nem kell figyelni csak bárhogyan tologatni a sorokat az első sor készenlétéig.

Következő fázis az összes alatta lévő sor kirakása, az utolsó sort leszámítva. Erre van egy nagyon hatékony technika, úgynevezett „csel”, ami segítségével a fentebbi sorok elrontása nélkül rakjuk be a megfelelő helyre a képkockát. Ezt a műveletsort is növekvő sorrendben hajtjuk végre.

A mozdulatsor a következő:

1. bemérjük a helyére rakandó kockát, jelen esetben az 5-ös számút
2. bemérjük a kocka megfelelő helyét (hol kellene legyen), jelen esetben az 1-es alatt, azaz a 7-es számú helyén
3. a kockát (5) a belehelyezendő oszlop mellé toljuk, azaz a második oszlopba, jelen esetben ez pont ilyen helyeztben áll fent ezért ezt a lépést kihagyjuk
4. a kocka helyét (1-es alatt: 7) lehozzuk a kocka (5) mellé
5. a kocka sorát (a sor, amiben szerepel az 5: harmadik sor) jobbra vagy balra toljuk, hogy a kocka elérje a megfelelő helyét, jelen esetben, hogy az 5-ös az 1-es alatt legyen
6. a mozgatott oszlopot (első oszlop) visszatoljuk a megfelelő állapotba, azaz a kialkotott képpárt visszahelyezzük az eddig elkészült kép többi eleméhez

Az utolsó fázis, a legnehezebb, az utolsó sor kirakását jelenti. Eddig, mindig az alatta lévő sorok feláldozásával, vagyis ezek összekeverésével, könnyedén lehet kirakni a fentebbi sorokat. Most viszont elérkeztünk ahhoz a ponthoz amikor nincs mit feláldozni. Éppen ezért, a jelenleg egyetlen dolog, amit tehetünk, az, hogy az utolsó két sort felváltva rakjuk ki, egyszer egyiket, követezőre a másikat felkeverve, ezáltal cserélődnek egymáshoz képest a kockák, majd végül a kép kirakódik. Nem találtam konkrét sablont, mi szerint a puzzle mindig kirakható a megadott helyzetekből, viszont nekem ez a próbálgatás mindig működött. Ha te esetleg tudsz erre egy konkrét, biztos megoldási mozdulatsort, amely bármilyen hasonló helyzetben felállított puzzle játékot megold (vagyis csak az utolsó sor van összekeverve), akkor megoldásod leírását sok szeretettel várom a [*gerzson709@gmail.com*](mailto:gerzson709@gmail.com) email-címre.

# Bibliográfia

1. [*https://www.c-sharpcorner.com/UploadFile/834980/how-to-pass-data-from-one-form-to-other-form-in-windows-form/*](https://www.c-sharpcorner.com/UploadFile/834980/how-to-pass-data-from-one-form-to-other-form-in-windows-form/)
2. *https://www.daveoncsharp.com/2009/09/using-the-button-dialogresult-property-in-your-custom-dialog/*
3. [*https://stackoverflow.com/questions/21362070/how-can-i-foreach-all-buttons-in-winforms-c*](https://stackoverflow.com/questions/21362070/how-can-i-foreach-all-buttons-in-winforms-c)
4. [*https://support.microsoft.com/en-us/topic/description-of-visual-studio-2010-service-pack-1-1f12811e-3826-6728-9f40-b11ee9ae2a0e*](https://support.microsoft.com/en-us/topic/description-of-visual-studio-2010-service-pack-1-1f12811e-3826-6728-9f40-b11ee9ae2a0e)