**Baranya Vármegyei Szakképzési Centrum Simonyi Károly Technikum és Szakképző Iskola**

**Vizsgaremek**

Készítették: Pete Krisztián

Szabó Patrik

**Pécs**

**2025**

**Baranya Vármegyei Szakképzési Centrum Simonyi Károly Technikum és Szakképző Iskola**

*Szakma megnevezése: Szoftverfejlesztő és –tesztelő*

*A szakma azonosító száma: 5 0613 12 03*

**Vizsgaremek**

Nonogram Puzzle

Készítették: Pete Krisztián

Szabó Patrik

**Pécs**

**2024**

**EREDETISÉGI NYILATKOZAT**

Alulírottak: Pete Krisztián Károly, Szabó Patrik a Baranya Vármegyei SzC Simonyi Károly Technikum és Szakképző Iskola, Szoftverfejlesztő és tesztelő végzős tanulói, büntetőjogi és fegyelmi felelősségünk tudatában nyilatkozunk és aláírásunkkal igazoljuk, hogy a

Nonogram Puzzle

című vizsgaremek saját, önálló munkánk, az abban hivatkozott szakirodalom felhasználása a forráskezelés szabályai szerint történt.

Tudomásul vesszük, hogy vizsgaremek esetén plágiumnak számít:

- szószerinti idézet közlése idézőjel és hivatkozás megjelölése nélkül;

- tartalmi idézet hivatkozás megjelölése nélkül;

- más publikált gondolatainak saját gondolatként való feltüntetése.

Alulírott kijelentjük, hogy a plágium fogalmát megismertük, és tudomásul vesszük, hogy plágium esetén a vizsgaremekünk visszautasításra kerül.

Pécs, 2025. február hó 15. nap

Pete Krisztián Károly

Szabó Patrik

tanulók

Tartalom

[1. **Bevezető** 1](#_Toc187013624)

[**1.** **Csapatmunka bemutatása** 1](#_Toc187013625)

[**2.** **Felhasználói dokumentáció** 1](#_Toc187013626)

[**2. A program általános specifikációja** 1](#_Toc187013627)

[2.1 Mi az a nonogram? 2](#_Toc187013628)

[2.2 Alapszabályok 2](#_Toc187013629)

[**3. Rendszerkövetelmény** 3](#_Toc187013630)

[3.1 Hardverkövetelmények 3](#_Toc187013631)

[3.2 Szoftverkövetelmények 4](#_Toc187013632)

[**4. A játék kicsomagolása és elindítása** 4](#_Toc187013633)

[**5. A program használatának a részletes leírása** 5](#_Toc187013634)

[**3.** **Fejlesztői dokumentáció** 15](#_Toc187013635)

[**6. Témaválasztás indoklása** 15](#_Toc187013636)

[**7. Alkalmazott fejlesztői eszközök** 15](#_Toc187013637)

[7.1 Felhasznált nyelvek 15](#_Toc187013638)

[7.2 Fejlesztői környezet 17](#_Toc187013639)

[**8. Tervezési módszer** 20](#_Toc187013640)

[8.1 Objektumorientált Programozás (OOP) 20](#_Toc187013641)

[8.2 MVVM Tervezési Minta 21](#_Toc187013642)

[8.3 Agilis Fejlesztési Módszertan 22](#_Toc187013643)

[8.4 Verziókezelés 22](#_Toc187013644)

[**9. OOP megvalósítása** 22](#_Toc187013645)

[9.1 Absztrakció 23](#_Toc187013646)

[9.2 Inheritance (Öröklődés) 23](#_Toc187013647)

[9.3 Polymorphism (Polimorfizmus) 24](#_Toc187013648)

[9.5 Példa az MVVM Mintára 25](#_Toc187013649)

[**10. Projektmenedzsment szoftver** 27](#_Toc187013650)

[**11. Adatmodell** 30](#_Toc187013651)

[11.1 Tervezés 32](#_Toc187013652)

[11.2 Normalizálás 36](#_Toc187013653)

[11.3 Adatvédelem (GDPR és SQL-injektálás) 43](#_Toc187013654)

[**12. Részletes feladatspecifikáció, algoritmusok** 44](#_Toc187013655)

[12.1 Feladatspecifikáció 44](#_Toc187013656)

[**13. Forráskód** 46](#_Toc187013657)

[13.1 LoginViewModel.cs 46](#_Toc187013658)

[13.2 DBManager.cs 48](#_Toc187013659)

[13.3 HashHelper.cs 50](#_Toc187013660)

[13.4 MainViewModel Osztály 50](#_Toc187013661)

[Adatbázis Kapcsolat (LoadUserData 53. ábra) 53](#_Toc187013662)

[13.5 Játékállapot Kezelés 53](#_Toc187013663)

[**14. Tesztelési dokumentáció** 66](#_Toc187013664)

[**15. Továbbfejlesztési lehetőségek** 67](#_Toc187013665)

[**IV. Összegzés** 69](#_Toc187013666)

[**Hivatkozások:** 69](#_Toc187013667)

# **Bevezető**

A projekt a következő publikus GitHub linken tekinthető meg: https://github.com/Example-Repo/Nonogram

Vizsgaremekünk témája a Nonogram című logikai játék, amely a rejtett kép kirakására ösztönzi a játékosokat rácsokban elhelyezett számok segítségével. A játékhoz weboldalt is készítettünk, amelynek segítségével a felhasználó megtekintheti a játékban elért eredményeit.

## **Csapatmunka bemutatása**

* **Pete Krisztián** - Design, játék oldal, adatbázis:
  + Krisztián felelt a játék designjának megtervezéséért és megvalósításáért.
  + Ő készítette a játék funkcionalitását, játéktér oldalát, amely tartalmazza a főbb funkciókat és navigációs elemeket.
  + Az adatbázis tervezése és implementálása is az ő feladata volt.
* **Szabó Patrik** - Egyéb funkciók megvalósítása:
  + Patrik felelt a bejelentkező felület, a regisztráció, és az elfelejtett jelszó funkciók implementálásáért.
  + Ő valósította meg a segítségek vásárlását és a profil nézetet.
  + A játék tesztelése is az ő feladata volt.

# **Felhasználói dokumentáció**

## **2. A program általános specifikációja**

Ebben a fejezetben bemutatjuk a Nonogram játék főbb jellemzőit és funkcióit, hogy segítsük a leendő játékosokat az eligazodásban és a döntéshozatalban.

A játék indítását követően amint regisztráltunk már kezdhetjük is a játékot. e-mailben kapunk egy kis eligazítást a játék menetéről és a szabályokról.

A Nonogram egy izgalmas logikai játék, amely számos kihívást és szórakozást kínál a játékosok számára. A játék lényege, hogy a játékosok kitöltsenek egy rácsot, ahol a számok jelzik, hogy az egyes sorokban és oszlopokban hány mezőt kell beszínezni. Ahogy a játékosok haladnak előre a különböző pályákon, egyre összetettebbé és nehezebbé válik a kirakós, így folyamatosan új stratégiákat és technikákat kell alkalmazniuk.

Azonban ez a játék nem hagy magunkra ugyanis segítségeket vehetünk igénybe Ha elakadtunk. természetesen nem ingyen ugyanis úgynevezett tokenekért cserébe vásárolhatunk. de a regisztráció akkor ajándékba Kapunk 50 kiinduló tokent ebből felhasználhattunk a segítségekre. Tokent és pontokat sikeresen megfejtett képekért kapunk, így tudunk további segítségeket vásárolni.

A Nonogram játék során a játékosok élvezhetik a logikai gondolkodás kihívásait és az egyre bonyolultabb rejtvények megoldásának örömét. A sikerélmény, amelyet egy-egy kirakós megoldása nyújt, motivációt és elégedettséget ad a játékosoknak. Emellett a játék remek lehetőséget nyújt a szabadidő hasznos eltöltésére és a mentális képességek fejlesztésére.

A Nonogram különféle pályákat és kihívásokat kínál, így minden játékos megtalálhatja a számára megfelelő nehézségi szintet. Akár kezdő, akár haladó játékosok is élvezhetik a Nonogram világát, hiszen mindig van lehetőség a fejlődésre és az újabb kihívások leküzdésére.

Összefoglalva, a Nonogram játék kiváló szórakozást és kihívásokat nyújt minden logikai játék kedvelő számára. A játék élménye és a megoldott rejtvények sikere garantáltan magával ragadja a játékosokat, és biztosítja a folyamatos szórakozást.

### 2.1 Mi az a nonogram?

A nonogram, más néven picross, griddler vagy Hanjie, egy logikai kirakós játék, amelyben egy négyzetrácsot kell kitölteni fekete és üres (vagy más színű) mezőkkel bizonyos számértékek alapján. A játék célja, hogy egy rejtett képet fedjünk fel a rács kitöltésével. A kitöltéshez a rács mellett és felett található számok vannak a segítségünkre, melyek az adott sorban vagy oszlopban található teli (fekete vagy színes) négyzetek egymáshoz képesti elhelyezkedésére utalnak.

### 2.2 Alapszabályok

1. A számok jelentése
   * Ha egy sor vagy oszlop mellett például a {3, 2} számokat látod, az azt jelenti, hogy először három összefüggő mezőt kell beszínezni, majd legalább egy üres hely kihagyása után két másik összefüggő mező következik.
   * A különböző blokkok között legalább egy üres mező kell legyen. (Ez alól csak a színes nonogramok jelentenek kivételt, de ilyen a játék egyelőre nem tartalmaz.)
2. A kitöltés módja
   * Egy négyzetet be lehet színezni, ha biztosak vagyunk benne, hogy az egy adott számhoz tartozik. Bizonyos esetekben ez egyértelmű de más esetekben korántsem biztos.
     + Például ha a sorunk 20 négyzetből áll és a sor melletti szám a 16, akkor biztosak lehetünk benne, hogy az 5-16. négyzetek telik lesznek, még ha azt nem is tudjuk, hogy mi a helyzet az 1-4. és a 17-20. négyzetekkel. Ha ugyanis az elejéről színezzük a 16 négyzetet, akkor az utolsó a 16. lesz, ha pedig a végéről, akkor az első az 5., 1gy bizonyos, hogy bárhogy is helyezzünk el 16 egymást érintő négyzetet a 20 négyzetből álló sorban az 5-16. négyzetek nem lehetnek üresek.
   * Egy négyzetet ki lehet hagyni (x-szel jelölni vagy üresen hagyni), ha biztosak vagyunk benne, hogy az nem része egyetlen számsorozatnak sem.
3. A rács logikus kitöltése
   * A játék során a sorok és oszlopok számainak figyelembevételével kell következtetni arra, hogy mely négyzeteket kell beszínezni vagy kihagyni.
   * Általában azokat a sorokat vagy oszlopokat érdemes először megvizsgálni, ahol nagy számok szerepelnek, vagy sok szám szerepel, mert ezekből könnyebben lehet következtetéseket levonni.

## **3. Rendszerkövetelmény**

Rendszerkövetelményként feltüntettük a minimális és ajánlott hardver konfigurációt, amely a játék és a weboldal futtatásához szükséges. Ezen kívül még feltüntettük, hogy mely operációs rendszeren fut és milyen egyéb szoftver komponensek szükségesek a működéséhez.

### 3.1 Hardverkövetelmények

**Minimum:**

* Processzor: Intel Core i5 vagy ekvivalens AMD processzor
* Memória: 4 GB RAM
* Grafikus kártya: Intel HD 530
* Tárhely: Legalább 10 GB szabad tárhely
* Operációs rendszer: Windows 10 vagy újabb, macOS 10.12 vagy újabb, Linux (64-bit)

**Ajánlott:**

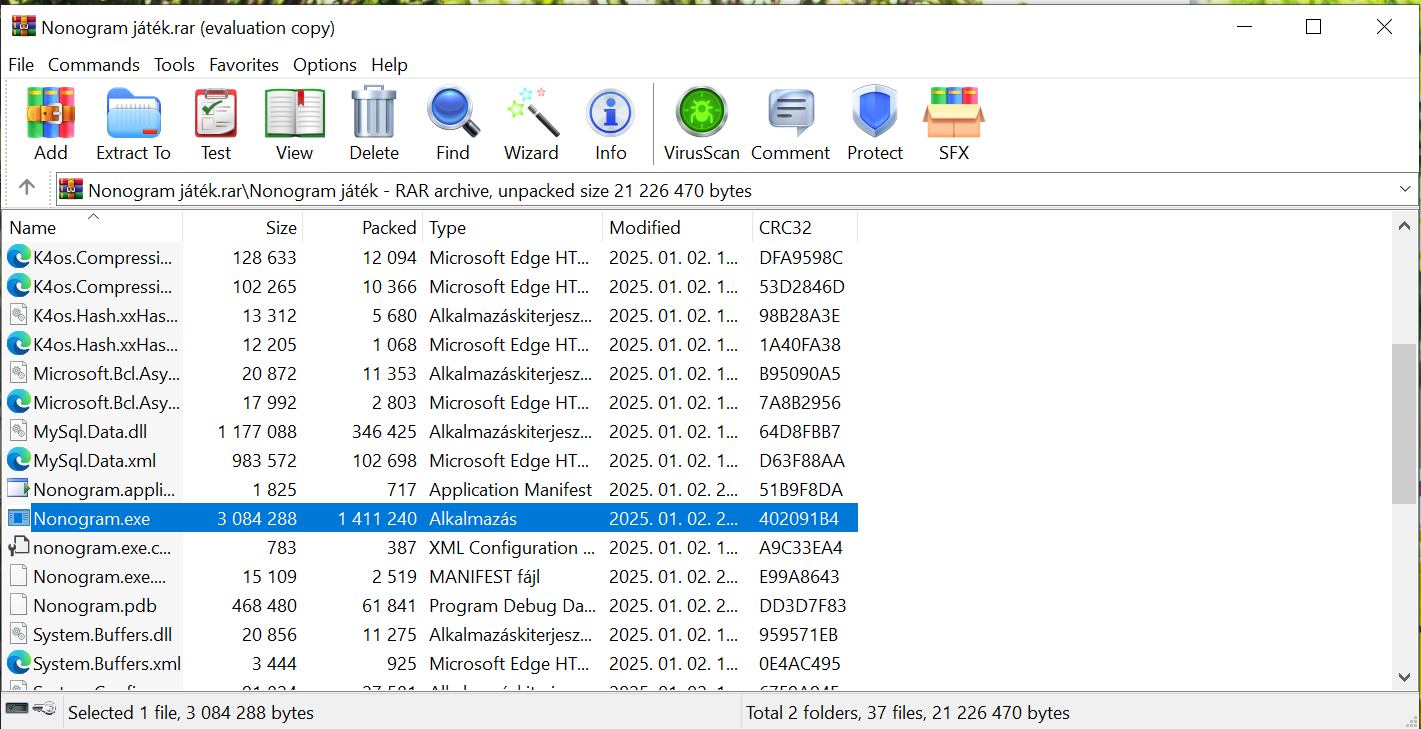
* Processzor: Intel Core i7 vagy ekvivalens AMD processzor
* Memória: 8 GB RAM
* Grafikus kártya: NVIDIA GeForce GTX 1060 vagy AMD Radeon RX 580
* Tárhely: Legalább 20 GB szabad tárhely
* Operációs rendszer: Windows 10 vagy újabb, macOS 10.14 vagy újabb, Linux (64-bit)

### 3.2 Szoftverkövetelmények

Operációs rendszer: Windows 10, macOS 10.15, vagy Ubuntu 20.04 LTS

## **4. A játék kicsomagolása és elindítása**

A játékot nem szükséges telepíteni, elég csak pl. WinRAR-ral kicsomagolni a játékot tartalmazó zip fájlt a felhasználó által megjelölt meghajtóra és elindítani a programot az Nonogram.exe fájl segítségével, ahogy az 1. ábrán is látható.



1. ábra: A játék kicsomagolása és elindítása

A kicsomagolás után akár egy parancsikont is létrehozhat az asztalon és onnan rögtön tudja indítani a játékot. Én is így tettem.

## **5. A program használatának a részletes leírása**

A játék használatának első lépése a regisztráció vagy bejelentkezés. Amikor először megnyitod a játékot először is regisztrálnod kell egy fiókot, ha még nem rendelkezel egyel. Ehhez meg kell adnod néhány alapvető információt, például felhasználónevet, jelszót és e-mail címet. Miután sikeresen regisztráltál, bejelentkezhetsz az általad létrehozott fiókkal. Amikor a felhasználó elindítja a játékot, a 2. ábrán található regisztrációs és/vagy bejelentkező felület fogja fogadni.

A képen szöveg, képernyőkép, fa, kültéri látható

Automatikusan generált leírás

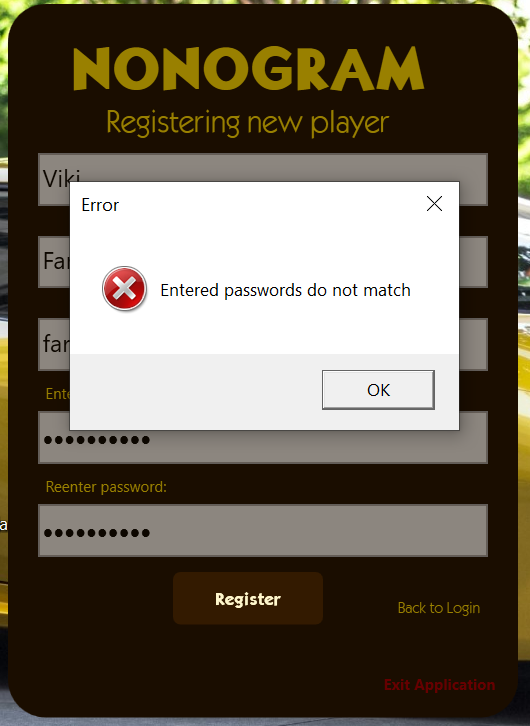
. ábra: Bejelentkező és Regisztrációs felület

Amennyiben a felhasználó rossz vagy nem létező felhasználónevet és jelszót ad meg, vagy regisztrációkor nem ugyanaz a két jelszó, akkor a 3. és 4. ábrán található hibaüzenettel találkozik:

A képen szöveg, képernyőkép, Téglalap, szoftver látható

Automatikusan generált leírás

3. ábra: Rossz jelszó vagy felhasználónév



4. ábra Regisztrációnál nem egyezik a két beírt jelszó

Amennyiben a jelszót jól adtuk meg és rányomunk a regisztráció gombra, hibaüzenetet fogunk kapni ugyanis előbb el kell fogadnunk az adatkezelési tájékoztatót is.

A képen szöveg, képernyőkép, tervezés látható

Automatikusan generált leírás

5. ábra: Az adatkezelési tájékoztató elfogadása nélkül nem regisztrálhatunk

Az adatkezelési tájékoztatónak van egy linkje ha arra a ráklikkelünk akkor a következő oldal fog megjelenni:

A képen szöveg, képernyőkép, szoftver, Weblap látható

Automatikusan generált leírás

6. ábra: Az adatvédelmi tájékoztató

Amennyiben megegyező jelszót ad meg a felhasználó, a következő üzenetet kapja, amelyet az 7. ábra szemléltet.

A képen szöveg, képernyőkép látható

Automatikusan generált leírás

7. ábra Sikeres regisztráció, és a játék kezdéshez 50 tokent kap ajándékba a felhasználó

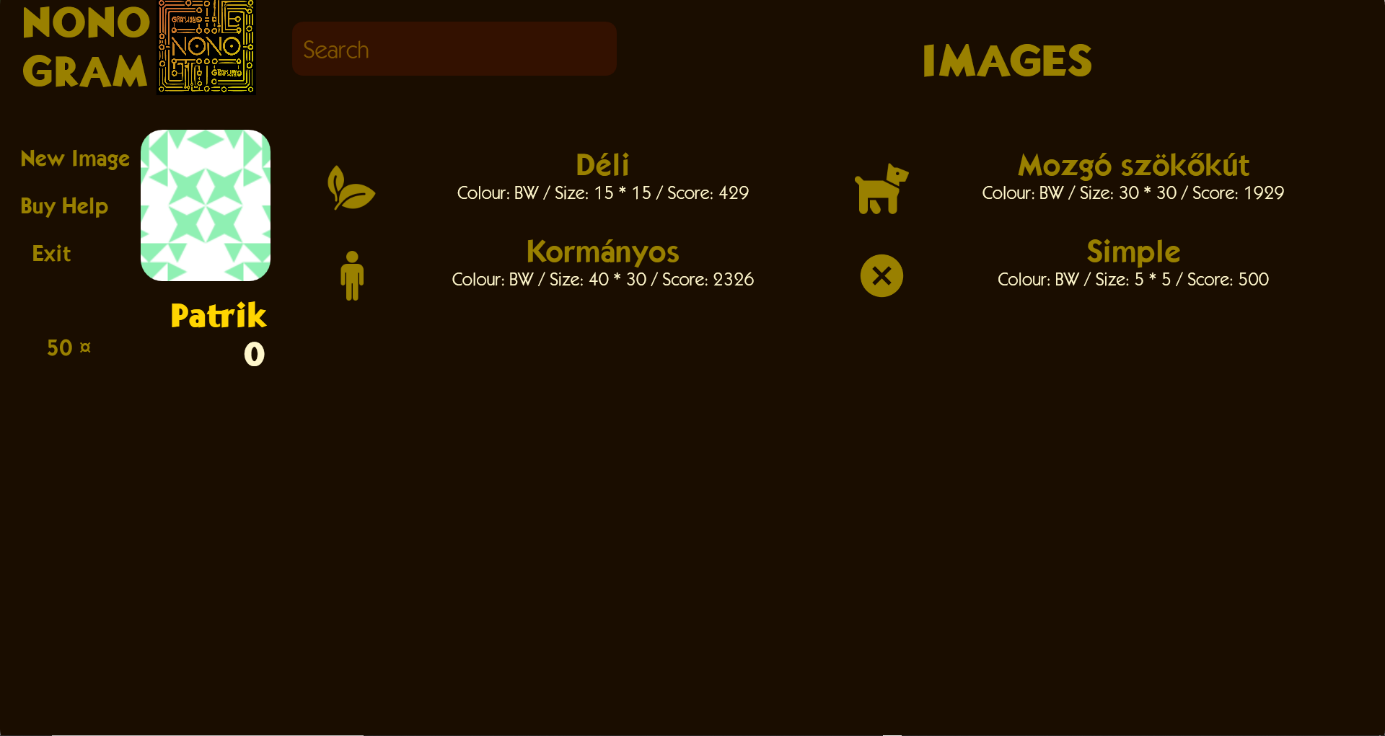
Még mielőtt belépnénk a játékba, fontos tudni, hogy egy e-mailt kap a felhasználó a megadott e-mail címre, ahol üdvözli a játék csapata, közli vele a felhasználónevét, és egy játékszabályzatot ami segíti a friss felhasználót, hogy még is hogyan kell kirakni a képet. Ezt a 8. ábra mutatja.

A képen képernyőkép, szöveg, szoftver, Számítógépes ikon látható

Automatikusan generált leírás

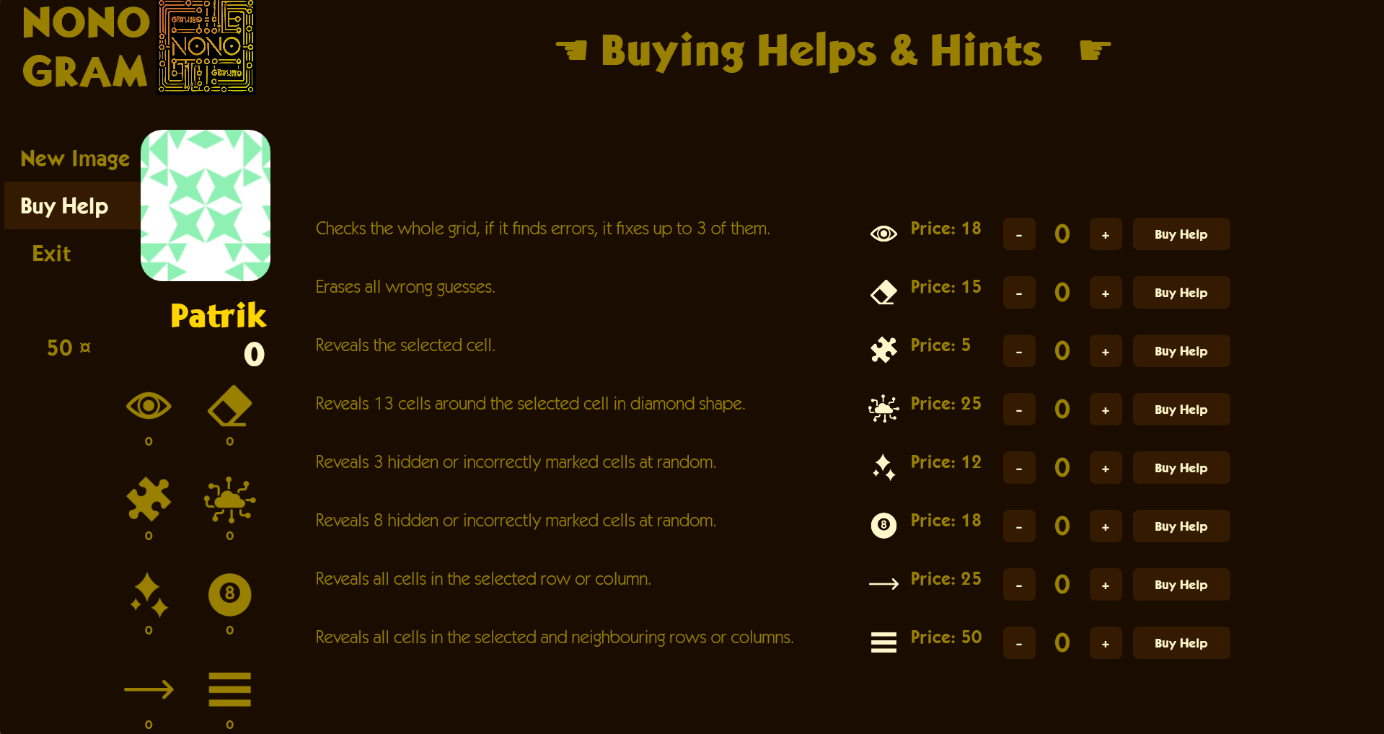
8. ábra Üdvözlő e-mail

Ezek után már be is lehet lépni, és ha minden adatot helyesen adtunk meg úgy a program tovább engedi őt a játék főmenüjébe, amelyet az 7. ábra mutat be.



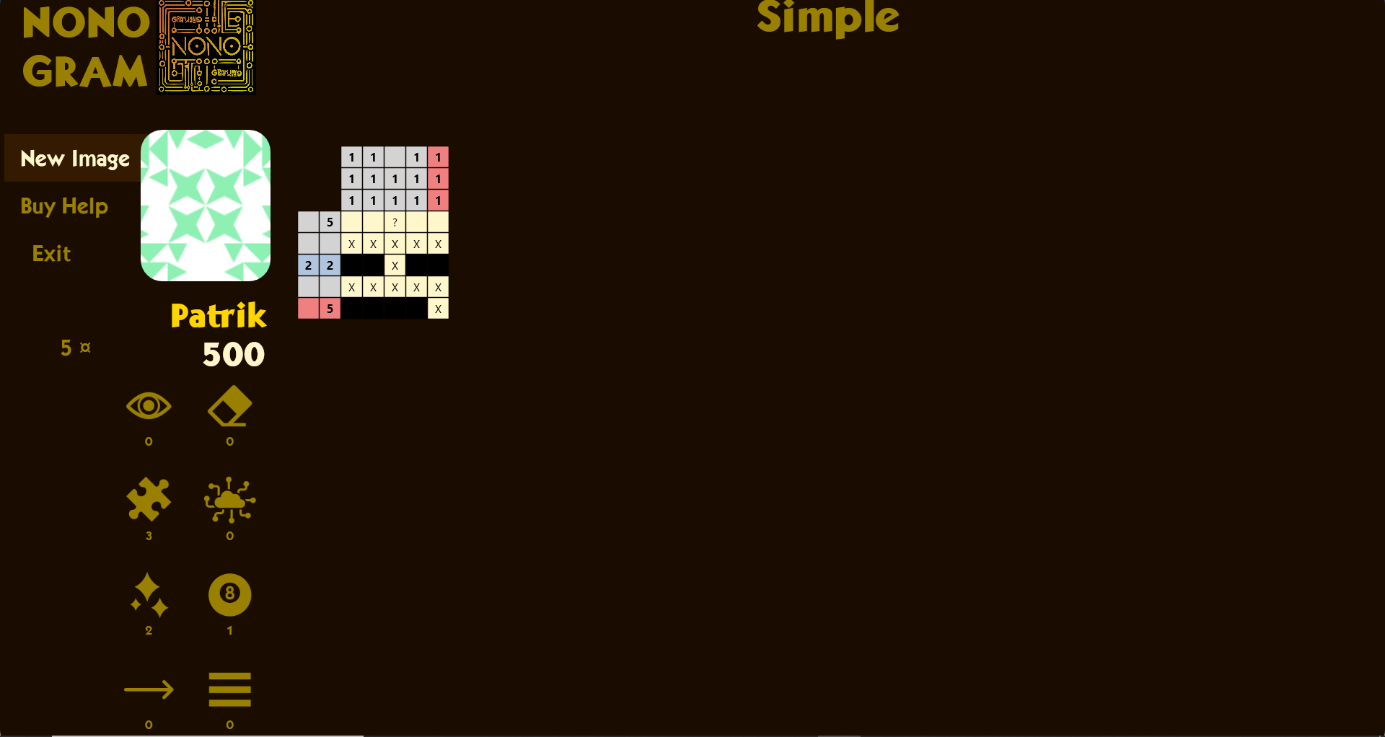
9. ábra Ez a fő ablak. Innen lehet elérni az összes funkciót.

A sikeres bejelentkezés után a játékos egyből a választható képek oldalra érkezik, ahol eldöntheti melyik képet szeretné felfedni. Ezt a 9. ábra is mutatja. A játék kezdése előtt lehetősége van a felhasználónak segítséget vásárolnia amit felhasználhat a játék során, ami segíti őt a kép felfedésében. Ezeket a segítségeket úgynevezett tokenekkel veheti meg. A 10. ábra be is mutatja hogyan is néz ki a vásártér.



10. ábra Segítségek vásárlása a kép felfedéséhez

Innentől kezdve, hogy megvásárolta a szükséges segítségeket el is kezdheti a játékot. A 11. ábrán már egy kiválasztott kép menetét látjuk. A képernyő felső részében a kiválasztott kép a neve található jelen esetben SIMPLE, bal oldalon pedig a felhasználó névnek és a pontok alatt jelenik mega a segítségek tábla, ahol majd fel lehet használni a különböző segítségek mennyiségét itt jelenik mega mennyiségben az, amit megvásároltunk. Ettől jobbra egy oszlopos és soros elrendezésű mátrixot láthatunk, ahol meg vannak adva számok és egy nagyobb tér, ahol majd a képet kell kiraknunk. Az adott kép mérete, hogy hányszor hányas rács alkotja azt még a főmenüben látjuk.

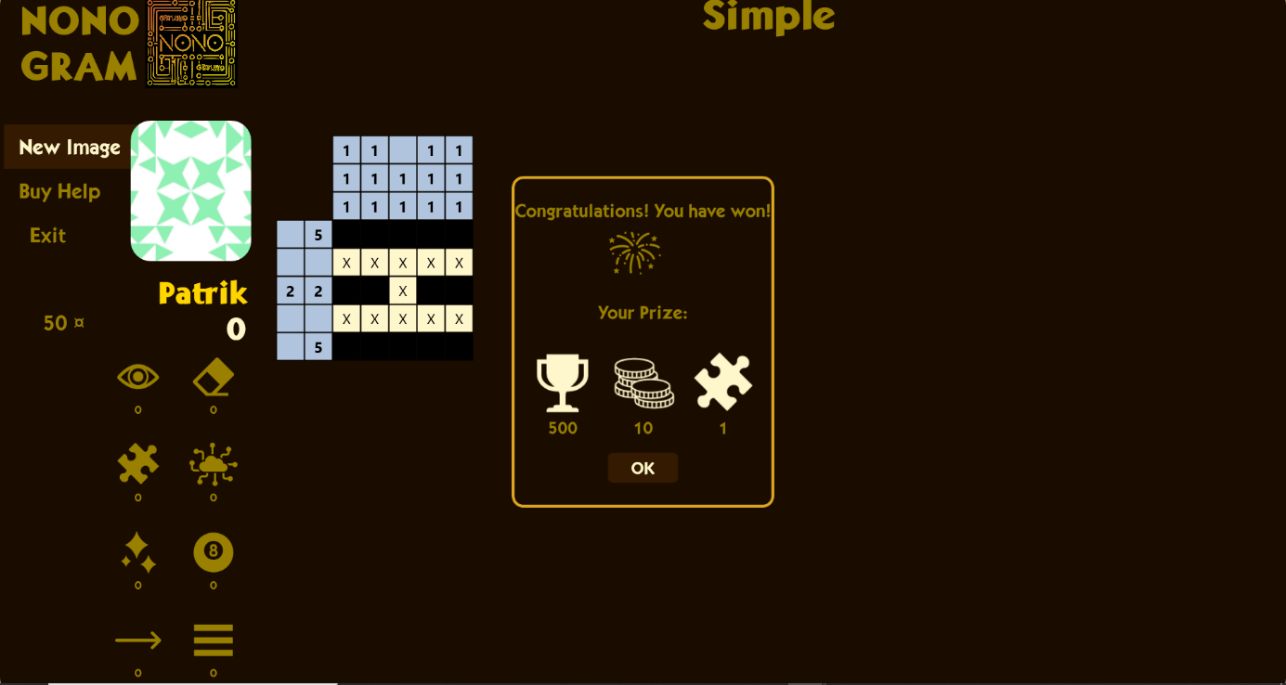


11. ábra Játék tér, ahol a képet lehet megfejteni a számok alapján

A 11. ábrán látható a már megkezdett kép megfejtése. A vajszínűleg rácsos részen első kattintással tudjuk bejelölni a képet alkotó fekete négyzettel. Ha bizonytalanok vagyunk abban, hogy egy adott négyzetet ki kellett tölteni az kérdőjel le tudjuk jelölni viszont amit biztosan kizárunk azt egy x-el jelöljük, illetve újra üresen is tudjuk hagyni az adott négyzetet lássuk is hogy ezt hogyan tudjuk változtatni. Egy adott négyzetre való kattintással először feketével jelölibe, második kattintással egy X-et rak ki, harmadik kattintással egy ?-jelet rak ki, negyedik kattintással pedig újra üres négyzetet kapunk. így tudjuk változtatni hogy mit szeretnénk jelölni az adott négyzetben.

Láthatjuk, hogy az ábrán a számoknál a sorok és oszlopok számánál pirossal és kékkel színezi be az adott négyzetet. A piros azt jelöli, hogy a megadott oszlop, vagy sorszámhoz képest Mennyi négyzet van bejelölve. Láthatjuk hogy ahol ötlete négyzetet kéne kijelölni otthoni a sorban lévő négyzetek száma is öt, és mivel csak négy van bejelölve és az ötödik ki van X-elve ezért jelzi a program hogy hibásan van kitöltve az a sor, ugyanez vonatkozik az oszlopra is. Amennyiben kékkel van bejelölve úgy az adott sor vagy oszlop helyesen van bejelölve.

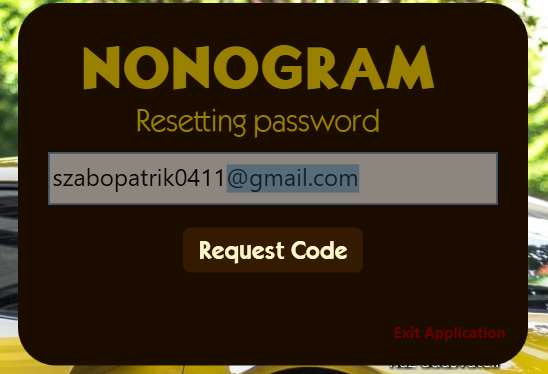
A következő 12. ábrán látható a kép helyes kitöltése tehát ha megoldottuk a játékot a következő ablak fog minket fogadni.



12. ábra Sikeresen kitöltött játék esetén megjelenő gratuláló ablak ahol jutalmat is kapunk.

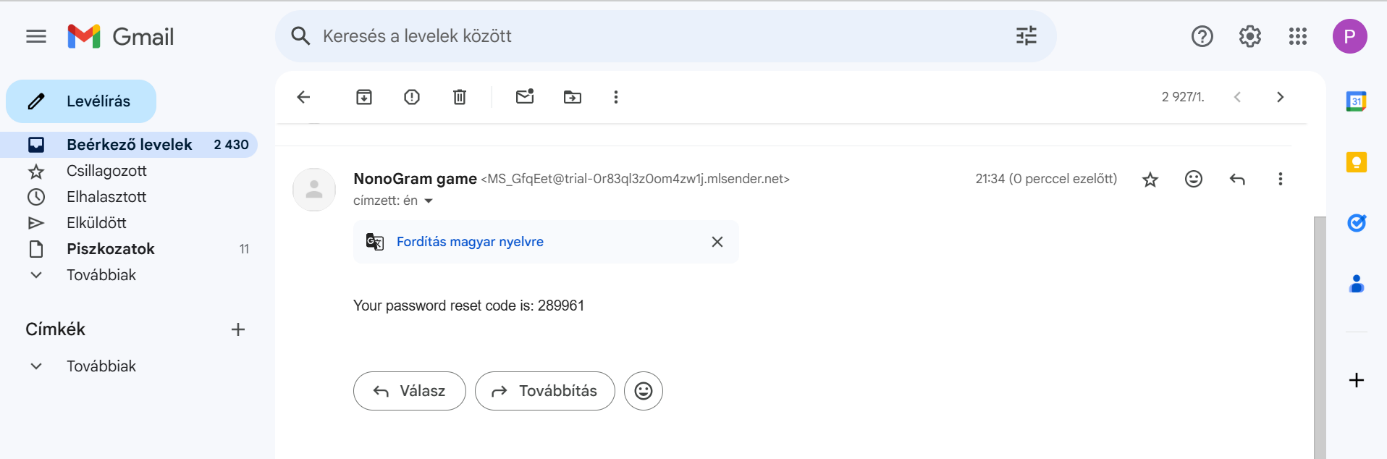
Ha sikeresen megoldottuk a képet, akkor egy külön ablak jelenik meg egy gratulációval és egy tűzijáték animációval és nem maradunk jutalom nélkül, ugyanis az adott képhez járó pontokat, tokent és egy random ajándék segítséget kapunk.

Amennyiben félbehagytuk a játékot és kiléptünk a programból a félbehagyott játék elmentődik. Ezt tudjuk folytatni a legközelebbi bejelentkezéskor. Ha elfelejtettük a jelszavunkat nem kell aggódnunk ugyanis a bejelentkező felületen az elfelejtett jelszóra kattintva kérhetünk újra jelszót ami a következőképp történik. Ha rákattintunk az elfelejtett jelszó linkre egy másik oldal fog megjelenni ahol az e-mail címünket be kell írnunk ugyanis e-mailben fogunk egy megerősítő kódot kapni. ezt a következő ábra szemlélteti.



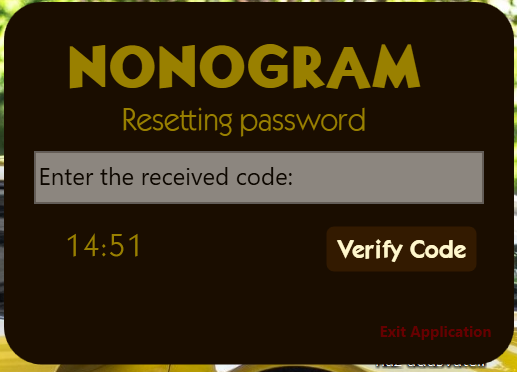
13. ábra E-mail cím beírása az elfelejtett jelszó kéréshez. ide küldi a megerősítő kódot.

Ha beírtuk a jelszót és rákattintottunk a kód kérdés a gombra akkor a megadott e-mail címre fogunk kapni egy automatikusan legenerált 6 számjegyű kódot, amit a 12. ábra szemléltet.



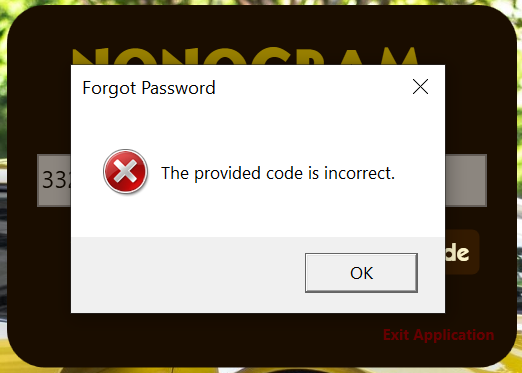
14. ábra A kód megérkezett az e-mailre.

Ezt a hat számjegyű kód arra szolgál, hogy más ne tudjon jelszót kérni ami profilunkhoz épp ezért küldi ki a megadott e-mail címre, ahonnan ki másolva be tudjuk illeszteni abba az ablakba ami a kód kérése ablak után jelent meg. ezt a 13. ábra szemlélteti.



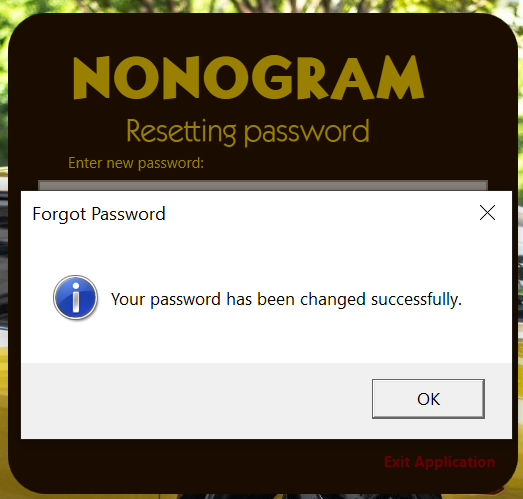
15. ábra Megerősítő kód beírása felület 15 perc áll rendelkezésre a beíráshoz.

Miután kértük a kódot 15 percünk van azt beírni. Ha helytelenül írjuk be akkor a következő hibaüzenet fog minket figyelmeztetni ezt a 16. ábra szemlélteti.



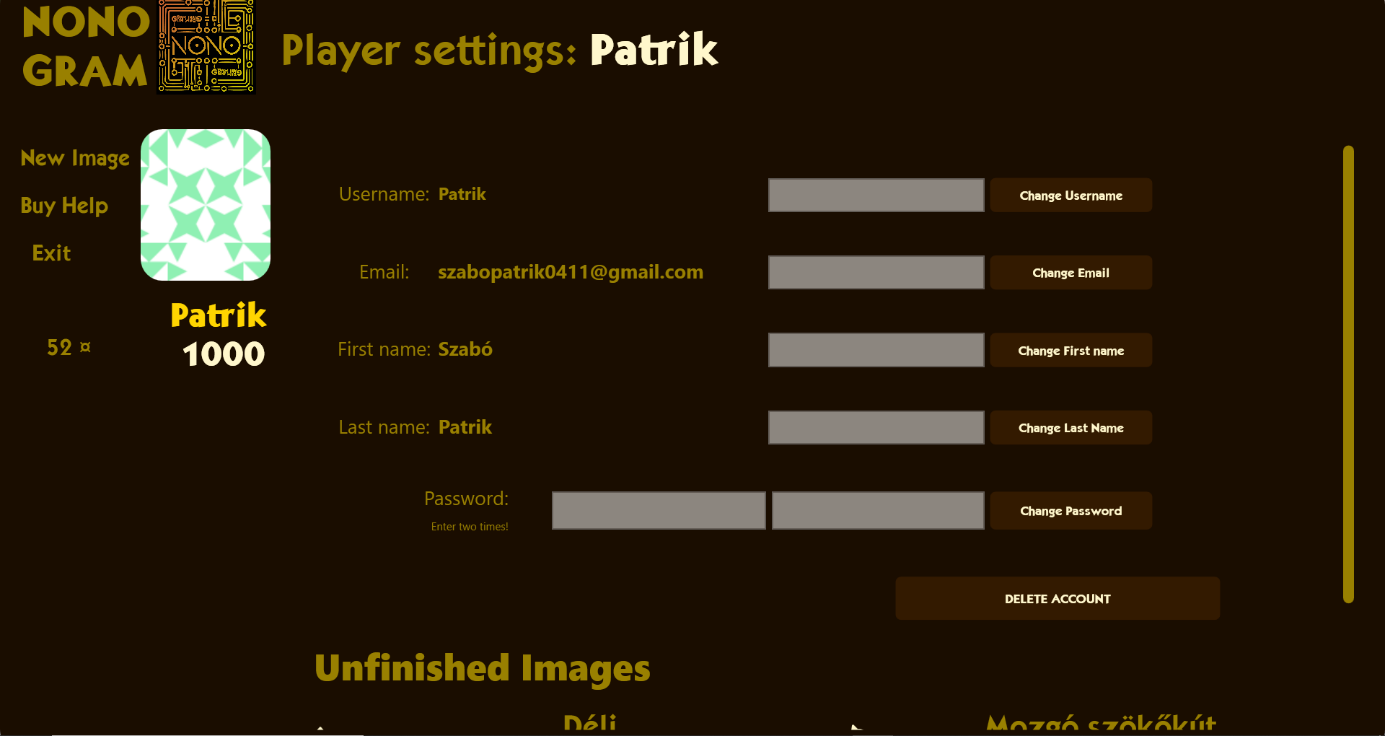
16. ábra Nem megegyező ellenőrző kód.

Ha helyesen írjuk be akkor pedig egy visszaigazoló ablak jönnek a számok megegyezéséről és megjelenik egy ablak, ahol új jelszót tudunk beállítani a profilunkhoz. Itt is kétszer kell megadni az új jelszót. Ennek sikerességét a 17. ábra szemlélteti.



17. ábra új jelszó sikeres beállítása

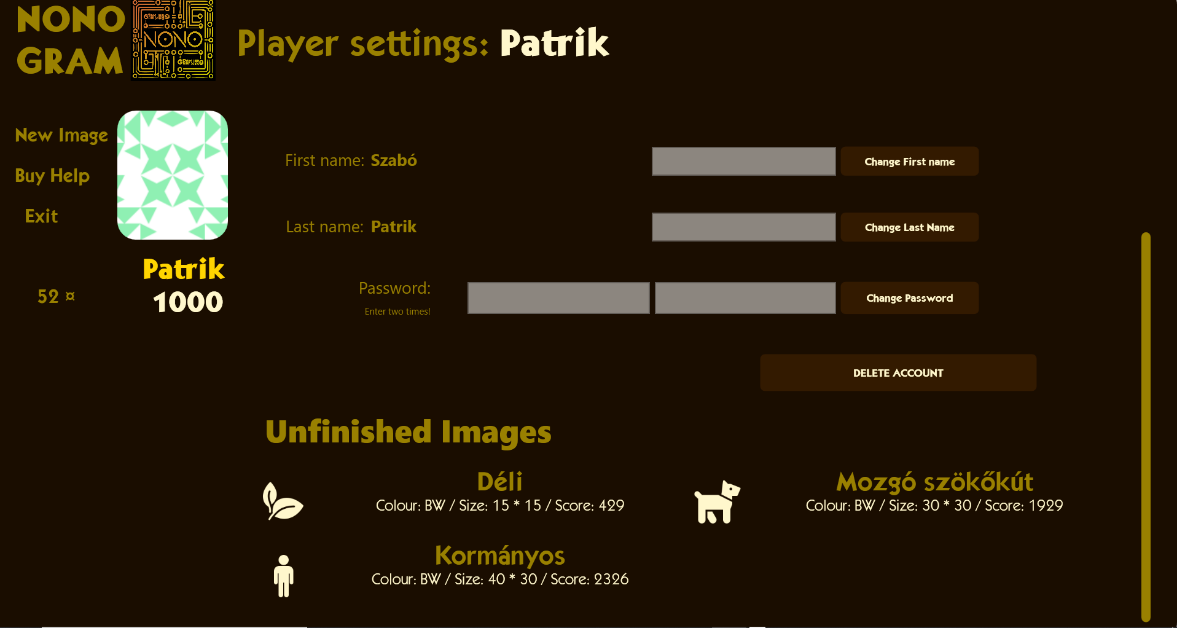
A profil-beállításokba úgy juthatunk, hogy az avatarra kattintunk. Erről egy TooTip is értesít minket. Az avatar az a kép, ami a menü sáv mellett található, és véletlenszerűen kerül beállításra a Gravatar szolgáltatásán keresztül. A profil beállításokat a 18. ábra szemlélteti.



18. ábra Profil szerkesztő felület.

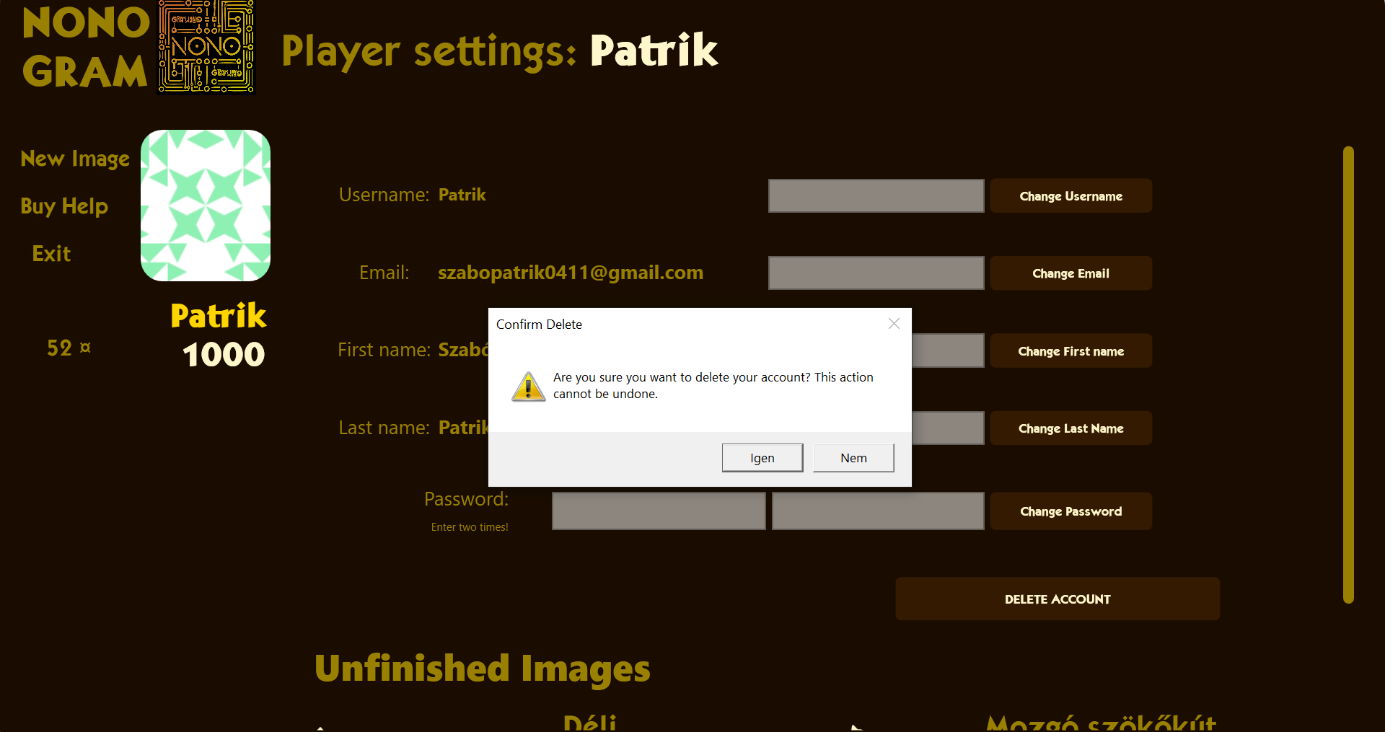
A profil beállításokban lehetőségünk van megváltoztatni a felhasználó nevünket, vezeték és keresztnevünket, az email címünket és a jelszavunkat. Lehetőségünk van még a teljes felhasználó törlésére is.

Ha lejjebb görgetünk, akkor ott jelennek meg a félbehagyott képek, melyek természetesen innen is folytathatók. Ezt a 19. ábra szemlélteti.



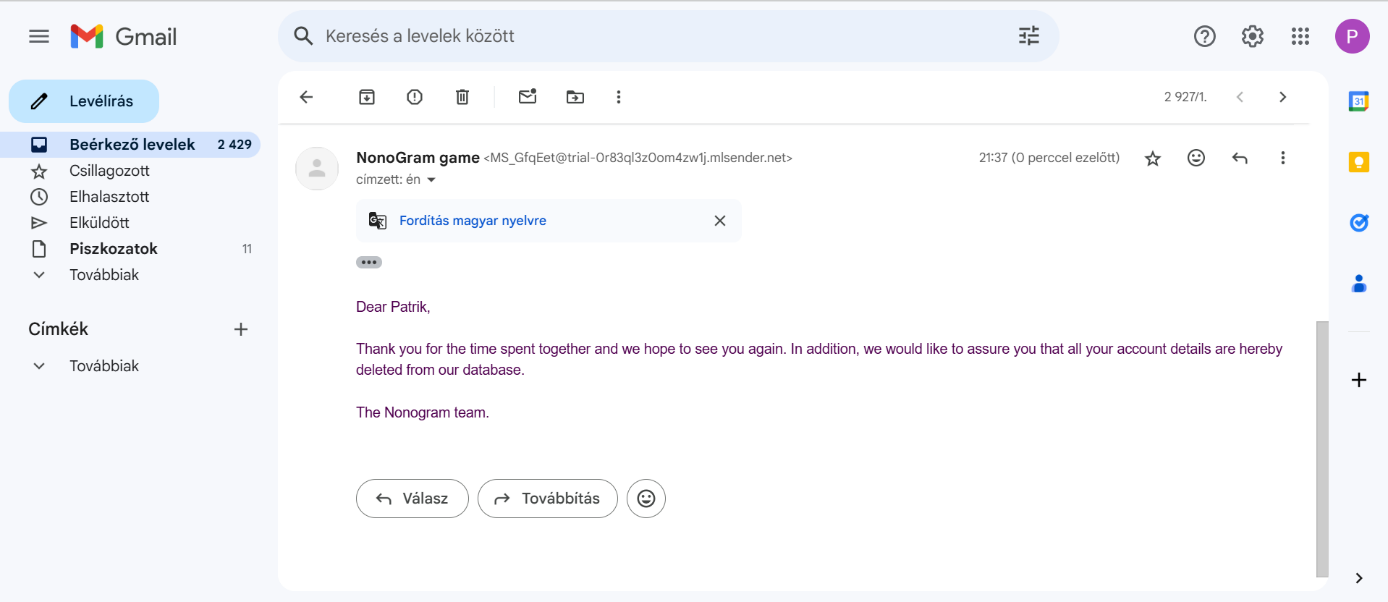
19. ábra Profil beállítások és félbehagyott játék

A következő, 20. ábrán pedig töröljük a meglévő felhasználót. Ha rákattintunk a felhasználó törlésére, a következő ablak jön fel egy figyelmeztető üzenettel, amiben még egyszer megkérdezi, biztosan törölni szeretnénk-e a profilt, mert az már nem visszavonható.



20. ábra A profil törlése, megerősítő, fihyelmeztető ablak.

Ha töröljük akkor a játék kilépett a bejelentkező felületre, és küld egy emailt a felhasználónak, amiben megköszönni a játék az együtt töltött időt és biztosítja a felhasználót arról, hogy minden adata törlésre kerül az adatbázisból. Ez a 21. ábrán látható.



21. ábra E-mail a törölt fiókról

# **Fejlesztői dokumentáció**

## **6. Témaválasztás indoklása**

A projektünk témájának kiválasztásakor több szempontot vettünk figyelembe, amelyek végül a Nonogram játék mellett döntettünk. Elsődlegesen asztali alkalmazás fejlesztésében gondolkodtunk, mivel ez az irányvonal lehetőséget kínál arra, hogy a programozási ismereteinket és készségeinket egy komplex, ugyanakkor jól strukturált keretek között kamatoztathassuk.

Úgy döntöttünk, hogy egy játékot készítünk, mivel a játékfejlesztés során számos izgalmas és változatos kihívással találkozhatunk, amelyek lehetővé teszik számunkra a kreativitás és a problémamegoldó készségek fejlesztését. A Nonogram játék kiválasztása különösen vonzó volt számunkra, mert ez egy olyan logikai játék, amelyben a játékosok számára a gondolkodás és a stratégiai tervezés kerül előtérbe. A játék egyszerűsége és kompaktsága lehetőséget ad arra, hogy a játék mechanikájára és a felhasználói élményre összpontosítsunk.

Ezen kívül fontos szempont volt a rendelkezésünkre álló idő és erőforrások hatékony felhasználása. A Nonogram játék megvalósítása viszonylag könnyen kivitelezhető az általunk megszabott időkereten belül, miközben lehetőséget ad arra, hogy a fejlesztés során alaposan elmélyedjünk a játék logikai összefüggéseiben és a programozási kihívásokban.

## **7. Alkalmazott fejlesztői eszközök**

### 7.1 Felhasznált nyelvek

Minthogy a játék alapvetően egy WPF alkalmazásként készült, a fejlesztés során a C#, a XAML és a MySQL nyelveket használtuk, illetve az adatok újra felhasználható tárolása érdekében a csv és a kiküldendő e-mailek formázásánál egy minimális mennyiségű HTML-t.

* **C#**: A fő programozási nyelve a projektünknek, amelyet a WPF (Windows Presentation Foundation) alkalmazás fejlesztéséhez használtunk. A nyelv erőteljes típusellenőrzést és objektumorientált programozási lehetőségeket biztosít. A mi programunkban a C# kód tartalmazza az üzleti (ügyviteli) logikát, a ViewModel-eket, a Model-eket és a különböző segédosztályokat. Ilyen kódok felelnek a
  + **Adatkezelésért**: Az adatbázisokkal való interakció, beleértve az adatok lekérdezését, módosítását és mentését. Például a DbManager osztályban található metódusok, amelyek az adatbázis inicializálását, táblák létrehozását és adatok betöltését végzik.
  + **Felhasználói műveletek kezelése**: A felhasználói műveletek, mint például a bejelentkezés, regisztráció, jelszó visszaállítás és felhasználói adatok módosítása. Ezeket a műveleteket a ViewModel-ekben található metódusok kezelik, mint például a LoginViewModel, RegisterViewModel és ForgotPasswordViewModel.
  + **Játéklogika**: A játék működéséhez szükséges logika, mint például a képállapot mentése, a játék megnyerésének ellenőrzése és a segítségek alkalmazása. Ezt a logikát a GameViewModel és a GameGrid osztályok tartalmazzák.
  + **Segítségek kezelése**: A játékban elérhető segítségek vásárlása és alkalmazása. Ezt a logikát a BuyHelpViewModel és a HelpTableViewModel osztályok tartalmazzák, amelyek a felhasználói tokenek kezelését és a segítségek frissítését végzik.
  + **E-mail küldés**: Az e-mail küldés logikája, például a jelszó visszaállításához szükséges kód küldése. Ezt a SmtpServer osztály kezeli.
* **XAML** (Extensible Application Markup Language): A felhasználói felület (**UI**) leírására és tervezésére használják. Lehetővé teszi a WPF-ben a felhasználói felület (a vizuális elemek és az adatkötések) deklaratív módon történő meghatározását és testreszabását. Ez elősegíti a UI és a logika szétválasztását.
  + **Adatkötések**: A XAML lehetővé teszi az adatkötések (data bindings) deklaratív módon történő meghatározását, azt, hogy a UI elemek közvetlenül kapcsolódhatnak a ViewModel-ekben vagy Model-ekben található adatokhoz. Ez elősegíti az MVVM (Model-View-ViewModel) minta alkalmazását, amely elválasztja a UI-t az üzleti logikától.
  + **Stílusok és sablonok**: A XAML lehetőséget biztosít stílusok és sablonok (templates) definiálására, amelyek segítségével egységes megjelenést és viselkedést biztosíthatunk az alkalmazás különböző elemei számára. Ezeket a stíluslapokat a Style mappa tartalmazza.
  + **Erőforrások kezelése**: A XAML támogatja az erőforrások (resources) kezelését, amelyek lehetnek színek, stílusok, sablonok, képek stb. Az erőforrásokat központilag definiálhatjuk és újra felhasználhatjuk az alkalmazás különböző részein (App.xaml).
  + **Eseménykezelés**: A XAML lehetőséget biztosít eseménykezelők (event handlers) deklaratív módon történő meghatározására, amelyek segítségével a felhasználói interakciókra reagálhatunk. Az eseménykezelők a C# kódban definiált metódusokra hivatkoznak, amelyek végrehajtják a szükséges műveleteket.
* **SQL**: Az adatbázis-kezeléshez használják, különösen a MySQL adatbázisban tárolt adatok lekérdezéséhez és módosításához. Az SQL parancsok segítségével történik az adatbázis létrehozása, a táblák kezelése és az adatok manipulálása. Ezen parancsok alapvetően a DBManager osztályban találhatóak.
  + **CSV**: Az adatok importálásához és exportálásához használják, különösen a DB mappában található CSV fájlokban tárolt adatok kezeléséhez. A CSV fájlok egyszerű szöveges formátumban tárolják az adatokat, amelyeket könnyen lehet olvasni és írni.

### 7.2 Fejlesztői környezet

A projektünk során két fő fejlesztői eszközt használtunk: a Visual Studio 2022-t és a MySQL-t. Az alábbiakban részletesen ismertetjük ezeknek az eszközöknek a szerepét és fontosságát a fejlesztési folyamatban.

**Visual Studio 2022**

A Visual Studio 2022 egy integrált fejlesztőkörnyezet (IDE), amelyet elsősorban a .NET alapú alkalmazások, köztük a Windows Presentation Foundation (WPF) alkalmazások fejlesztésére használtunk. Azért választottuk ezt az eszközt, mert számos előnyt és funkciót kínál a fejlesztők számára, többek között:

* **Kódkezelés és hibakeresés**: A Visual Studio 2022 fejlett kódkezelési és hibakeresési funkciókkal rendelkezik, amelyek lehetővé teszik a kód gyors és hatékony írását, ellenőrzését és javítását.
* **Beépített eszközök és kiegészítők**: A Visual Studio számos beépített eszközt és kiegészítőt kínál, mint például a NuGet csomagkezelő, amely megkönnyíti a külső könyvtárak és csomagok integrálását a projektbe. Ezek közül néhányat igénybe is kellett vennünk, például a **MySql.Data**-t az adatbázis-kezeléshez, a **BouncyCastle.Cryptography**-t a jelszavak hash számításához, valamint pár adattömörítéshez és adatkommunikációhoz szükséges dependency csomagot is
* **IntelliSense**: Az IntelliSense funkció automatikus kódkiegészítést és szintaktikai javaslatokat kínál, amelyek segítenek a kód olvashatóságának és minőségének javításában.
* **Projekt- és megoldáskezelés**: A Visual Studio 2022 lehetővé teszi a komplex projektek és megoldások hatékony kezelését, így könnyedén kezelhetjük a különböző projektfájlokat és erőforrásokat.
* **Test Explorer**: A beépített Test Explorer lehetővé teszi az egységtesztek írását és futtatását, amelyek biztosítják a kód minőségét és stabilitását.

A Visual Studio 2022 segítségével gyorsan és hatékonyan fejleszthettük a Nonogram játékunkat, kihasználva az IDE nyújtotta számos előnyös funkciót és eszközt. Különösen hasznosnak találtuk a Visual Studio 2022 támogatását a WPF alkalmazások fejlesztéséhez, amely lehetővé tette számunkra a felhasználói felület intuitív és látványos kialakítását.

**MySQL**

A MySQL egy nyílt forráskódú relációs adatbázis-kezelő rendszer, amelyet az alkalmazás adatkezelési feladataihoz használtunk. Azért választottuk a MySQL-t, mert kiváló teljesítményt és megbízhatóságot kínál az adatbázis-kezelés terén. A MySQL főbb jellemzői és előnyei közé tartoznak:

* **Adatkezelés**: A MySQL lehetővé teszi az adatok hatékony tárolását, lekérdezését és kezelését, ami elengedhetetlen a játék adatainak kezeléséhez, mint például a felhasználói fiókok, eredmények és játékmenetek.
* **Skálázhatóság és teljesítmény**: A MySQL kiválóan alkalmas nagy mennyiségű adat kezelésére, és képes a teljesítmény optimalizálására, ami fontos szempont a nagyobb játékosbázis kezelésénél.
* **Biztonság**: A MySQL magas szintű biztonsági funkciókkal rendelkezik, amelyek biztosítják az adatok védelmét és integritását.
* **Könnyű integráció**: A MySQL könnyedén integrálható a Visual Studio 2022-vel és a .NET alapú alkalmazásokkal, ami megkönnyíti az adatbázis kezelését a fejlesztési folyamat során.

**GitHub**

A GitHub a legnépszerűbb felhőalapú Git verziókezelő platform, amely lehetővé teszi a fejlesztők számára, hogy együtt dolgozzanak a projektjeiken. A GitHub segítségével a csapatok könnyen kezelhetik a változásokat, nyomon követhetik a fejlesztéseket és együttműködhetnek a kód megosztásában és felülvizsgálatában. A platform számos funkciót kínál, többek között verziókezelést, probléma nyomon követést, ágkezelést, együttműködést és automatizált munkafolyamatokat.

**GitHub Projects**

A projektmenedzsment feladatokat a GitHub Projects keretében valósítottuk meg, mely lehetővé teszi a fejlesztők számára, hogy tervezzenek, együttműködjenek és nyomon kövessék munkájukat a GitHub Issues segítségével. A GitHub Projects táblázatokat, táblákat, egyedi mezőket, diagramokat és automatizálást kínál, amelyek segítségével a csapatok hatékonyan kezelhetik projektjeiket és alkalmazkodhatnak a különböző munkafolyamatokhoz.

**ERDPlus**

Az ERDPlus egy ingyenesen elérhető, egyszerű és hatékony adatmodellező eszköz, mellyel könnyen készíthetünk ERD-ket. Az ERD (Entity-Relationship Diagram) egy ábrázolási forma, amely relációs adatbázisok ábrázolására és tervezési céljaira szolgál. Az ERD célja az adatbázis-elemek struktúrájának és kapcsolatainak grafikus ábrázolása, hogy könnyebben megérthető legyen az adatbázis tervezése és felépítése. Az adatmodell rész ERD-i is innen származnak.

**XAMPP** ill. **MySQL Server 9.1 CE**

Az adatbázis és szerverműveletekhez egy localhost-ot használtunk, melyet a XAMPP ill. a MySQL Server 9.1 CE alkalmazások biztosítottak. A XAMPP egy könnyen telepíthető Apache disztribúció, amely tartalmazza a MariaDB-t, a PHP-t és a Perl-t is. Azért fejlesztették, hogy egyszerűsítse a fejlesztők számára a helyi webszerver környezet létrehozását, amely lehetővé teszi a webalkalmazások fejlesztését és tesztelését anélkül, hogy internetkapcsolatra lenne szükség. A XAMPP tartalmazza a phpMyAdmin-t is, amely egy webalapú eszköz a MySQL adatbázisok kezelésére. MySQL Server 9.1 az egyik legnépszerűbb nyílt forráskódú adatbázis-kezelő rendszer, és mivel a XAMPP-pal akadtak időközben problémák Pete Krisztián ezen szoftver ingyenesen elérhető community edition-jének telepítése mellett döntött (nem utolsósorban azért is, mert a LinqPad is ezt használja). Az adatbázis-műveletek ellenőrzésére ő a MySQL Workbench alkalmazást használta, így a dokumentáció megfelelő részei (táblák képei adatbázismodell-digaram is ebből az alkalmazásból származik).

Az alkalmazott fejlesztői eszközök segítségével olyan hatékony és megbízható rendszert hoztunk létre, amely biztosítja a Nonogram játék zavartalan működését és élvezetes játékélményt nyújt a felhasználók számára.

## **8. Tervezési módszer**

A Nonogram játék fejlesztése során különböző tervezési módszereket és elveket alkalmaztunk annak érdekében, hogy a projekt jól strukturált és könnyen kezelhető legyen. Az alábbiakban részletesen ismertetjük a legfontosabb tervezési módszereinket.

### 8.1 Objektumorientált Programozás (OOP)

Az objektumorientált programozás (OOP) alapelveit követve a kódunkat osztályok és objektumok segítségével szerveztük. Az OOP lehetővé teszi a kód újra felhasználhatóságát, karbantarthatóságát és skálázhatóságát, ami különösen fontos egy olyan komplex projekt esetében, mint a Nonogram. Az alábbi OOP alapelveket alkalmaztuk:

* **Encapsulation (Kapszulázás)**: Az adatokat és a hozzájuk tartozó funkciókat egyetlen osztályba zártuk, így biztosítva, hogy az adatok csak az osztályon belül legyenek elérhetők és módosíthatók.
* **Inheritance (Öröklődés)**: Az öröklődés segítségével a közös funkciókat és tulajdonságokat az ősosztályokban definiáltuk, majd azokat az alosztályok örökölték. Ez lehetővé tette a kód újra felhasználását és a komplexitás csökkentését.
* **Polymorphism (Polimorfizmus)**: A polimorfizmus segítségével azonos nevű metódusokat hoztunk létre különböző osztályokban, amelyek a konkrét osztálytól függően eltérő módon működhetnek. Ez lehetővé tette a rugalmas és bővíthető kód írását.

### 8.2 MVVM Tervezési Minta

A Nonogram játék fejlesztése során a Model-View-ViewModel (MVVM) tervezési mintát alkalmaztuk, amely különösen alkalmas a WPF alkalmazások fejlesztésére. Az MVVM minta szétválasztja a felhasználói felület logikáját a mögöttes adatkezeléstől és üzleti logikától, ezáltal tisztább és karbantarthatóbb kódot eredményez. Az MVVM minta három fő komponensből áll:

* **Model**: Az adatokat és az üzleti logikát reprezentálja. A Model osztályokban tároltuk az adatokat és a hozzájuk tartozó műveleteket.
  + A nonogram.MVVM.Model névtér tartalmazza a Model osztályokat, mint például a GameGrid, amely a játék rácsának adatait és logikáját kezeli. A GameGrid osztály felelős a rács inicializálásáért, a tippek kiszámításáért és a játékállapot kezeléséért.
* **View**: A felhasználói felületet (UI) reprezentálja. A View elemek felelősek a felhasználói interakciók megjelenítéséért és kezeléséért.
  + A nonogram.MVVM.View névtér tartalmazza a View osztályokat és XAML fájlokat, mint például a GameView.xaml, amely a játék rácsának megjelenítéséért felelős. A View elemek XAML-ben vannak definiálva, és a felhasználói interakciókat kezelik.
* **ViewModel**: A View és a Model közötti kapcsolatot biztosítja. A ViewModel tartalmazza a felhasználói felület logikáját, és összekapcsolja a View elemeket a Model osztályokkal.
  + A nonogram.MVVM.ViewModel névtér tartalmazza a ViewModel osztályokat, mint például a GameViewModel, amely a játék logikáját és a felhasználói interakciókat kezeli. A GameViewModel osztály összekapcsolja a GameGrid modellt a GameView nézettel, és biztosítja az adatkötéseket és a parancsok kezelését.

### 8.3 Agilis Fejlesztési Módszertan

Az agilis fejlesztési módszertan alapelveit követve a projektet iteratív és inkrementális módon fejlesztettük. Az agilis módszertan lehetővé tette számunkra, hogy gyorsan reagáljunk a változásokra és folyamatosan javítsuk a terméket a felhasználói visszajelzések alapján. Az agilis fejlesztési módszertan főbb jellemzői közé tartoznak:

* **Iteratív fejlesztés**: A projektet kisebb iterációkra bontottuk, amelyek során új funkciókat és javításokat integráltunk a meglévő kódba.
* **Folyamatos visszajelzés**: Rendszeresen teszteltük és értékeltük a terméket a felhasználói visszajelzések alapján, hogy biztosítsuk a magas minőséget és a felhasználói igényeknek való megfelelést.
* **Csapatszintű együttműködés**: Az agilis módszertan elősegíti a csapattagok közötti együttműködést és kommunikációt, ami növeli a hatékonyságot és a projekt sikerességét.

### 8.4 Verziókezelés

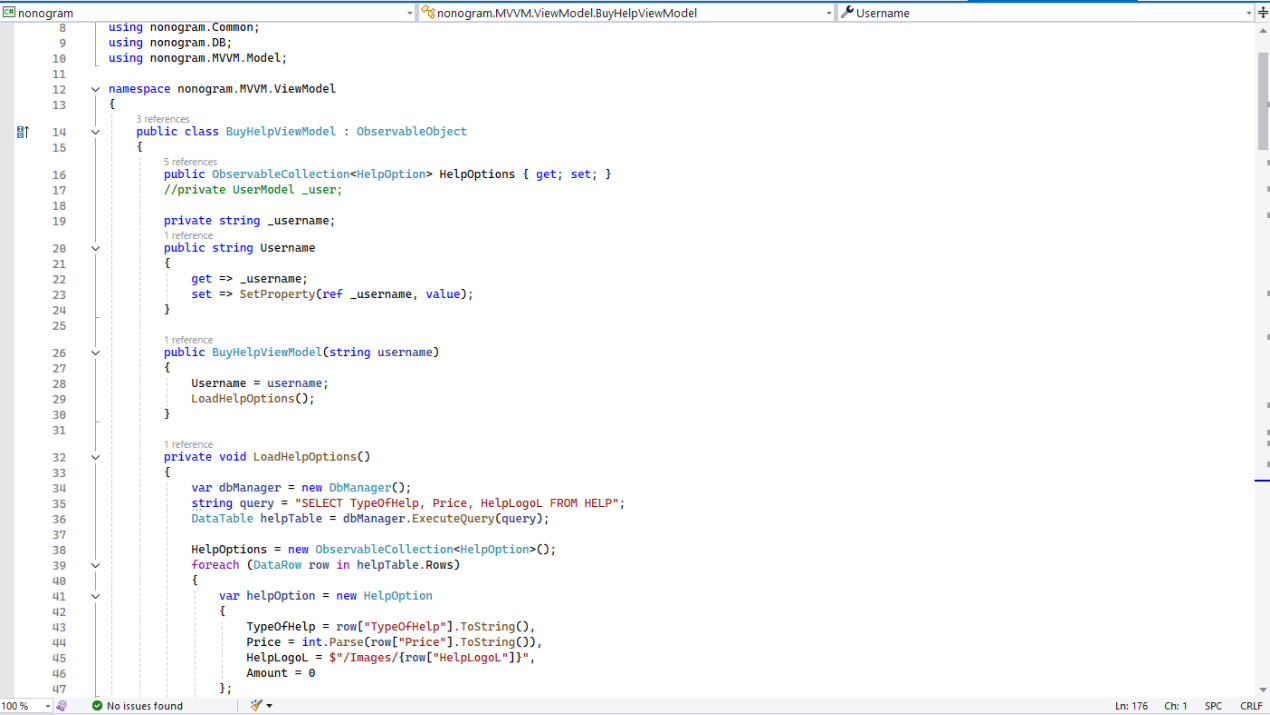
Mint a feljesztési környezet bemutatásánál már jeleztük, a projekt során a Git verziókezelési rendszert használtuk a kód nyomon követésére és kezelésére. A verziókezelés lehetővé tette számunkra, hogy a kód különböző verzióit nyomon kövessük, együttműködjünk a csapattagokkal, és könnyedén visszaállítsuk a korábbi állapotokat szükség esetén.

## **9. OOP megvalósítása**

A Nonogram játék fejlesztése során nagy hangsúlyt fektettünk az objektumorientált programozás (OOP) alapelveinek alkalmazására. Az OOP lehetővé teszi a kód újrafelhasználhatóságát, karbantarthatóságát és skálázhatóságát, amely különösen fontos egy komplex projekt esetében. Az alábbiakban bemutatom, hogyan alkalmaztuk az OOP alapelveit a projekt során.

### 9.1 Absztrakció

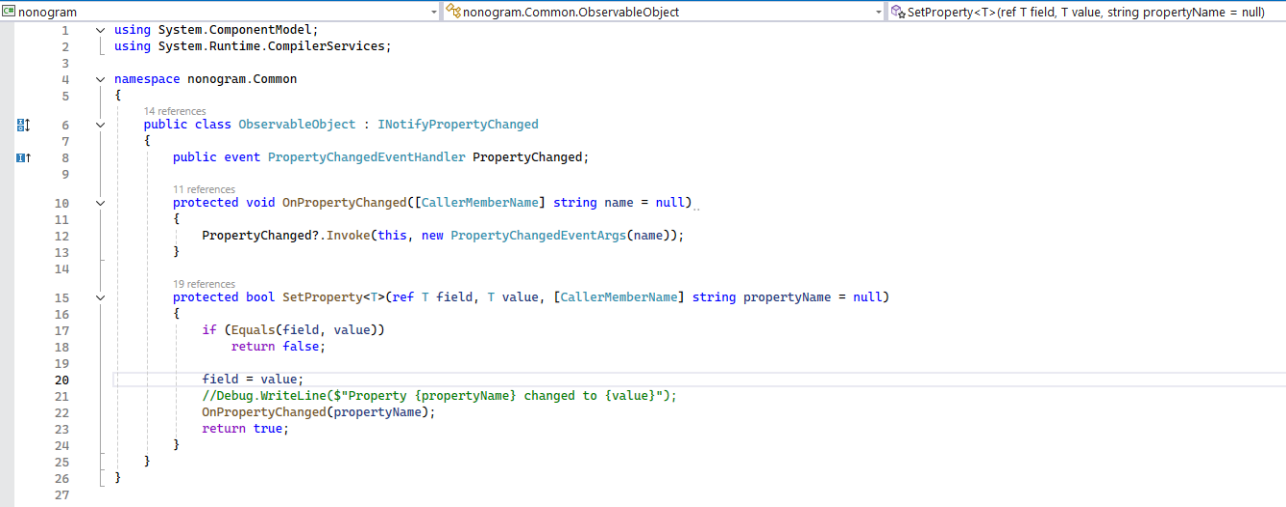
Az absztrakció lehetővé teszi a komplexitás elrejtését és egyszerű interfészek biztosítását a kód használatához. (Erre láthatunk példát a 22. ábrán)



22. ábra: A BuyHelpViewModel VM interfésze

### 9.2 Inheritance (Öröklődés)

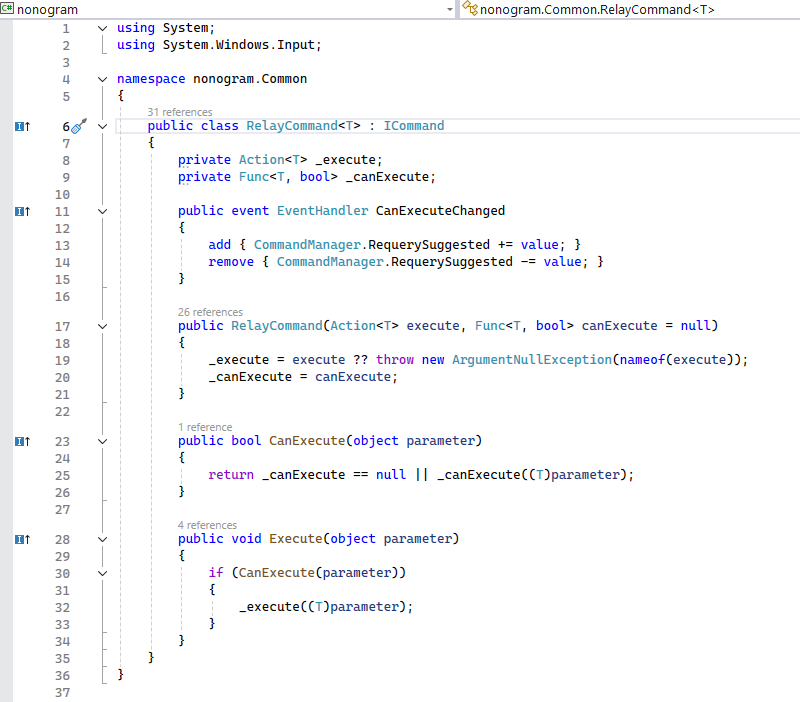
Az osztályok tulajdonságokat és metódusokat örökölhetnek más osztályoktól. Például az ObservableObject osztályból örökölt osztályok (23. ábra).



23. ábra: A Common folder egyik utility osztálya

### 9.3 Polymorphism (Polimorfizmus)

A különböző osztályokban azonos névvel rendelkező metódusok különböző módon viselkedhetnek. Például A RelayCommand osztály az ICommand interfészt valósítja meg, ami a futásidejű polimorfizmus egyik példája. Az ICommand interfész definiálja a CanExecute és Execute metódusokat, amelyeket a RelayCommand osztály implementál.



24. ábra: A RelayCommand osztály

**9.4 MVVM Tervezési Minta**

A Nonogram játék fejlesztése során a Model-View-ViewModel (MVVM) tervezési mintát alkalmaztuk, amely különösen alkalmas a WPF alkalmazások fejlesztésére. Az MVVM minta szétválasztja a felhasználói felület logikáját a mögöttes adatkezeléstől és üzleti logikától, ezáltal tisztább és karbantarthatóbb kódot eredményez. Az MVVM minta három fő komponensből áll:

* **Model:** Az adatokat és az üzleti logikát reprezentálja. Például a HelpOption osztályban tároltuk a játék segítség opcióinak adatait.
* **View:** A felhasználói felületet (UI) reprezentálja. A BuyHelpView.xaml fájlban definiáltuk a felhasználói felületet és annak elemeit.
* **ViewModel:** A View és a Model közötti kapcsolatot biztosítja. A BuyHelpViewModel osztályban tartalmaztuk a felhasználói felület logikáját, és összekapcsoltuk a View elemeket a Model osztályokkal.

### 9.5 Példa az MVVM Mintára

**Model:**

**A képen szöveg, képernyőkép, Betűtípus, szám látható

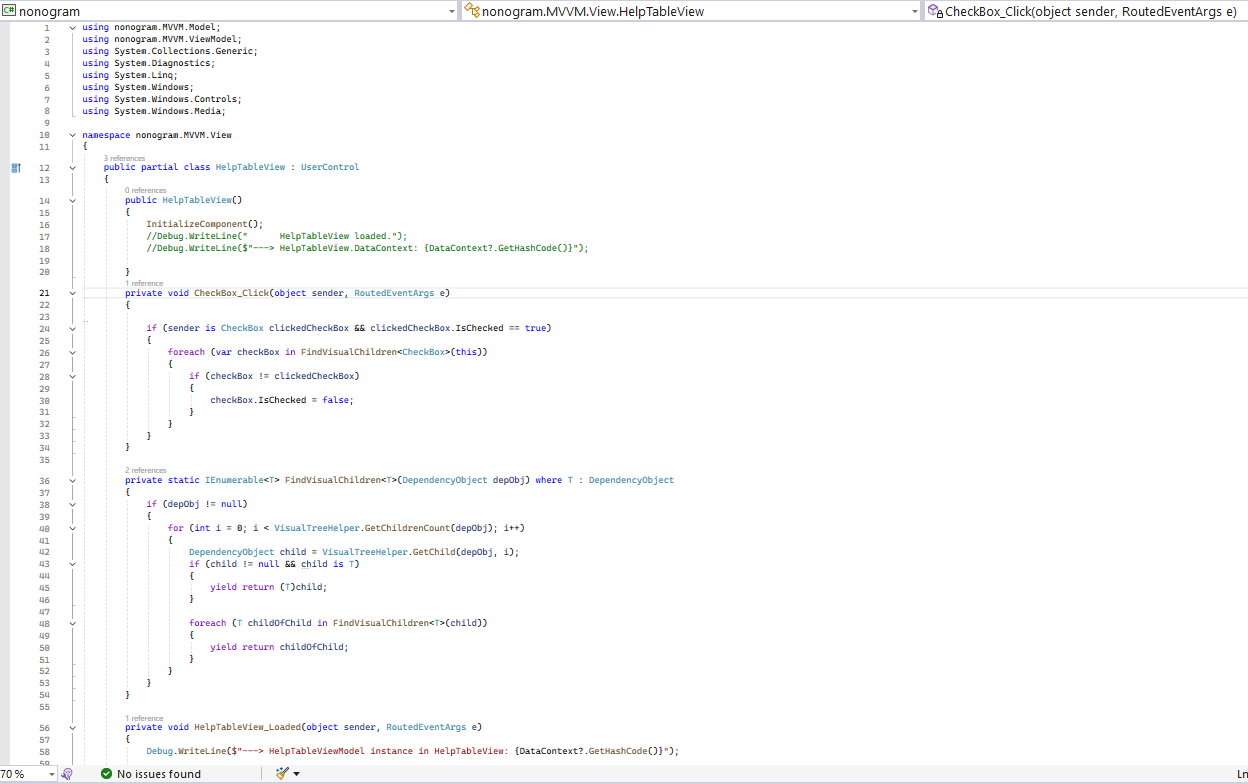
Automatikusan generált leírás**

25. ábra: A HelpOption Modell, amit több ViewModel is hivatkozik

**View** (xaml és code-behind)**:**

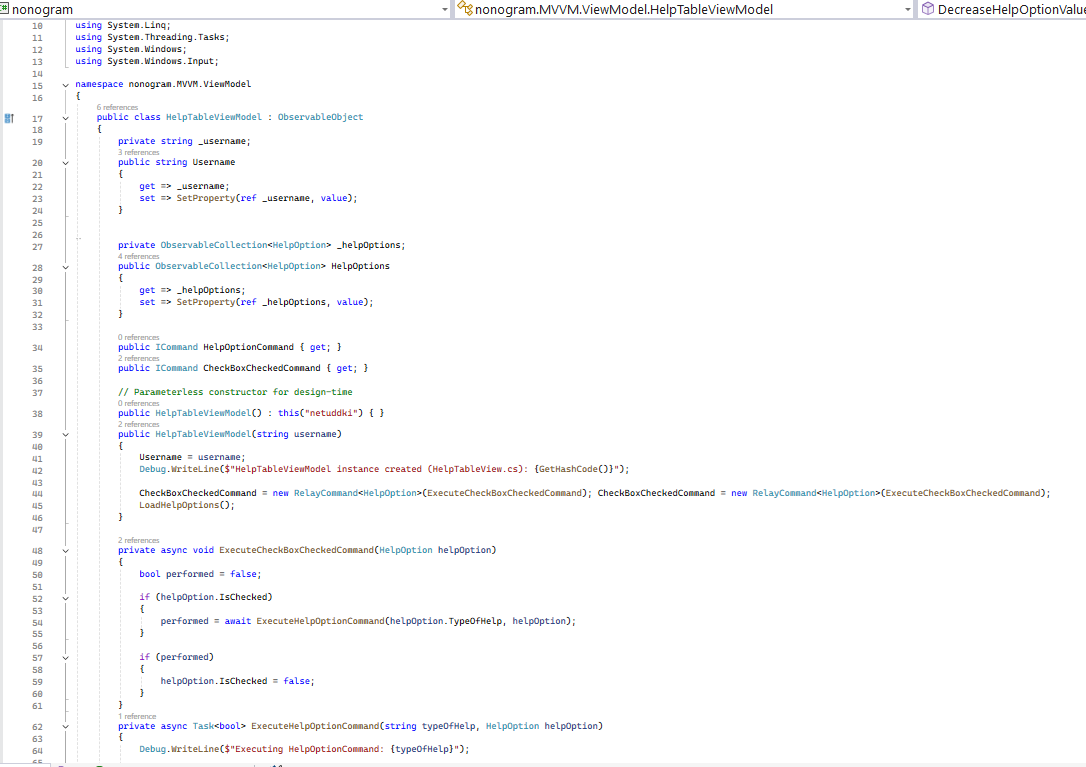
****

26. ábra: HelpTableView.xaml

****

27. ábra: HelpTableView.xaml.cs

**ViewModel:**



28. ábra: HelpTableViewModel

Ezek az OOP alapelvek segítettek abban, hogy a Nonogram játék kódját jól strukturált, könnyen karbantartható és bővíthető módon fejlesszük. Az objektumorientált megközelítés lehetővé tette számunkra, hogy a kódotok átlátható és moduláris legyen, így megkönnyítve a fejlesztést és a jövőbeni bővítéseket.

## **10. Projektmenedzsment szoftver**

Ugyan a projekt indulásakor nem látszott megoldhatatlan feladatnak a csapatmunka megszervezése, a feladatok kiosztása, vagy épp a határidők kijelölése és betartása, hiszen mindössze két fős csapatban dolgoztunk, mégis szerettünk volna valamiféle projektmenedzsment szoftvert használni, ha másra nem, hát a feladatok és a kontribúciók nyomon követésére, így azt a döntést hoztuk, hogy a lehető legegyszerűbb és legnyilvánvalóbb opciót választjuk, és a GitHub saját projectmenedzsment eszközét használjuk.

A GitHub Projects egy igen hasznos és széleskörűen használható eszköz, amely lehetőséget biztosít a csapatok számára, hogy hatékonyan követhessék a munkafolyamatokat és koordinálják a fejlesztési feladatokat. A GitHub, amelyet elsősorban forráskód-tárolásra és verziókövetésre használnak, a Projects funkcióval kibővítette kínálatát, hogy teljes körű projektmenedzsment-támogatást nyújthasson. A rendkívül rövid és feszes fejlesztési idő sajnos nem tette lehetővé számunkra, hogy maradéktalanul kihasználjuk a Projects-ben rejlő lehetőségeket, de egyrészt így is hasznos volt számunkra, másrészt szeretnénk azért bemutatni a szoftver funkcióit.

**A GitHub Projects főbb funkciói:**

A GitHub Projects alapvetően egy kanban-stílusú projektmenedzsment eszköz, amely kártyákra osztott táblákon keresztül követi nyomon a feladatokat. A Kanban egy vizuális projektmenedzsment módszer, amely az agilis módszertanok közé tartozik. Az 1940-es évek végén Japánban, a Toyota gyárban fejlesztették ki, és azóta számos iparágban alkalmazzák. A Kanban 4 fő alapelve: *a meglévő munkafolyamatokból induljunk ki* (Más, strukturáltabb agilis módszerekkel, például a SCRUM-mal, ellentétben a Kanban a csapat meglévő munkafolyamatait veszi alapul.); *fokozatos változásokkal haladjunk előre* (Kicsi és inkrementális kiigazításokkal kezdjünk, ne pedig nagy és átható változtatásokkal, így finomítsuk a projekt előrehaladását); *tartsuk tiszteletben a kiosztott szerepeket, felelősségeket és választott módszereket* (A Kanban más módszerekkel ellentétben nem ír elő konkrét csapatszerepeket.); *ösztönözzük a csapat minden tagjának kezdeményezését* (A Kanban nem hierarchikus struktúrájú, elismeri, hogy a változás a csapattagoktól is eredhet, nem csak a vezető pozícióban lévőkből.) (Morales 2023). Minden Kanban tábla három fő oszlopból áll: „Elvégzendő feladatok” (Todo), „Folyamatban lévő feladatok” (In Progress) és „Kész” (Done) feladatok. Ez a vizuális megközelítés segít a csapatoknak nyomon követni a feladatok állapotát és azonosítani a szűk keresztmetszeteket. A Kanban *előnyei* közé tartozik a *rugalmasság*, a *folyamatos fejlesztés ösztönzése* és a munkafolyamatok *átláthatósága*. Ugyanakkor *hátrányai* is vannak, melyek a mi a tényleges működésünkben is megmutatkoztak:

* a Kanban egyszerű, de nem tartalmaz részletes időterveket
* a Kanban táblák egészen addig működnek jól, amíg nem válnak túl bonyolulttá. A komplex projektek menedzselésére rétegezni a munkafolyamatokat.
* ahhoz, hogy hatékony munkaszervezési módszer legyen, a Kanban táblát rendszeresen frissíteni kell, ami nagyfokú fegyelmet követel a csapattól.

A GitHub Projects erre a Kanban alapra épülő szoftverfejlesztési, munkaszervezési eszköz, ami tökélesen integrált a GitHub-ba. Az alábbiakban röviden bemutatjuk a GitHub Projects legfontosabb funkcióit:

1. *Táblák és oszlopok*: magát a projektet egy táblában szervezhetjük, mely a fent részletezett oszlopokat tartalmazza (Todo, In Progress, Done - Tennivaló, Folyamatban, Kész”) Az oszlopok szabadon testreszabhatók, lehetővé téve az egyéni munkafolyamatok kialakítását.

2. *Kártyák*: Az oszlopok kártyákkal tölthetők fel, ahol minden kártya egy adott feladatot vagy tárolóhoz (repository) rendelt feladatot (issue) jelöl. A kártyákhoz hozzáadhatók leírások, hozzárendelhetők ismétlődések, felelősök, a munka mennyigét jelző méretjelzők, és határidők is. A kártyák átkonvertálhatók a tárolók feladataivá, sőt azok mérföldköveihez is hozzárendelhetők.

3. *Automatizáció:* Bizonyos műveletek automatizálhatók is, pl. egy feladat (issue) automatikusan átkerül az In Progress oszlopba, ha elkezdenek dolgozni rajta.

4. *Testreszabhatóság:* A tábla nézetei (Hátralék, Csapat kapacitás, Jelenlegi ismétlés, Útiterv, Saját elemek) szűrhetők és rendezhetők különböző attribútumok alapján, így a csapattagok könnyen megtalálják a számukra releváns információkat.

5. *Integráció:* A GitHub Projects zökkenőmentesen integrálható a GitHub más funkcióival, valamint külső eszközökkel, mint például Slack vagy Trello.

A képen szöveg, szoftver, szám, Weblap látható

Automatikusan generált leírás

29. ábra: A Nonogram project táblája (Backlog nézet)

És hogy mindez hogyan nézett ki a mi saját projektünk, a NonoGram alkalmazás kapcsán? Sajnos a képzési idő rövidsége és a mi saját egyéb irányú elfoglaltságaink miatt (mégiscsak felnőttképzésről van szó, ráadásul annak is egy csonkolt verziójáról) nem tudtuk maximálisan kihasználni a projektmenedzsment eszközben rejlő lehetőségeket. Abban mindenképp segítségünkre volt, hogy egyértelműen jelezni tudtuk egymás felé a feladatokat, amiken éppen dolgoztunk, sőt azt is jelezni tudtuk előre, hogy milyen feladatokat vállalunk magunkra. Ezeket a szempontokat leolvashatjuk a projekt táblájáról, melyet a 29. ábra tartalmaz. Ebben mindenképp nagy segítség volt a GitHub Projects. Mivel a Kanban rendszernek köszönhetően ez a módszer nem a munkakörök kiosztásán alapul, hanem a feladatokon, így a szervezésnek nem volt több szintje, és az idő rövidsége miatt arra sem volt lehetőségünk, hogy utánanézzünk, ezt hogyan tehetjük meg. Mivel a módszer nem tartalmaz részletes időtervet (noha a tárolóban (repository) kijelölhetők mérföldkövek, amivel némi komplexitás vihető az időtervekbe) ezért nem is sikerült pontosan tartanunk az amúgy sem eléggé pontosan kijelölt határidőinket. Sajnos a feladatok meghatározása sem volt egyenletes, hol nagyon részletes, hol pedig sokkal pontatlanabb leírásokkal dolgoztunk. És pontosan a fejlesztési idő rövidsége miatt (az első találkozásunk a WPF-fel szeptemberben történt, és december végére lényegében meg kellett valósítanunk a projektet), illetve a munka melletti tanulás miatt a GitHub Project tábláinak karbantartásai is el-elmaradozott.

## **11. Adatmodell**

Az adatmodell határozza meg az adatbázis szerkezetét, beleértve az adatok típusát, kapcsolatait, a rájuk vonatkozó feltételeket és a velük végezhető műveleteket. Az adatmodell célja, hogy a valós világ eseményeit és objektumait olyan formában írja le, amely számítógéppel könnyen kezelhető, valamint lehetővé tegye az adatok hatékony keresését és lekérdezését. Fontos megjegyezni, hogy a modellek nem képesek a valós világ minden jellemzőjét tökéletesen visszaadni, hiszen nem minden tulajdonság egyformán lényeges, és a túl sok információ felesleges adatok tárolásához vezetne. Ezért elengedhetetlen meghatározni, hogy mely tulajdonságokat kívánjuk rögzíteni az adatbázisban, majd ennek megfelelően definiálni az egyedtípusokat és az adatbázis szerkezetét.

A képen szöveg, képernyőkép, Betűtípus, embléma látható

Automatikusan generált leírás

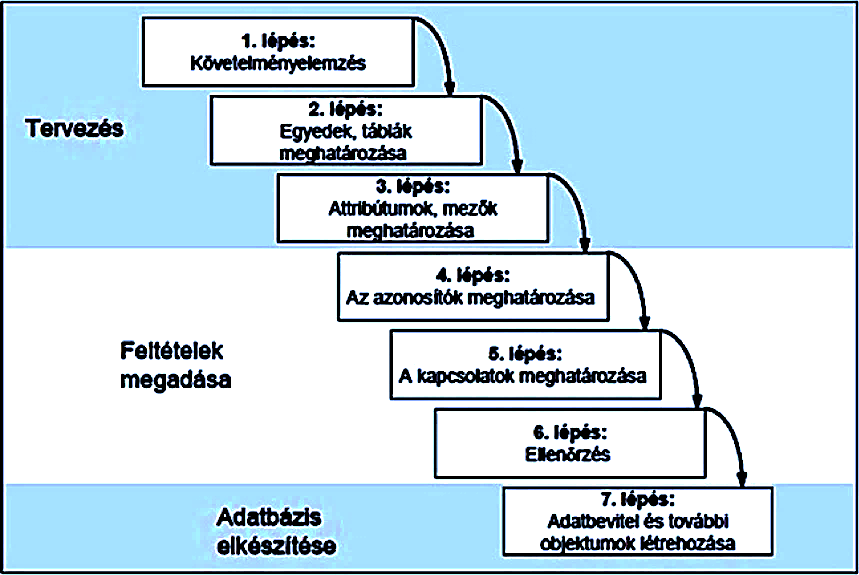
30. ábra: Adatmodell-típusok

Napjainkban négy fő adatmodell van használatban (Kardos 2008: 5-6.):

* a **hierarchikus** adatmodell: Egyszerűen alkalmazható fa szerkezettel bíró irányított gráfként szemléltethető adatmodell. Nagy hátránya, hogy bizonyos kapcsolatok nem szemléltethetők vele (például a több:több {N:M} kapcsolatok nem szemléltethetők ezzel az adatmodellel)
* a **hálós** adatmodell: A hálós adatmodellben az adatokat mutatókon keresztül tudjuk elérni, és kapcsolataikat is mutatókon keresztül tudjuk bemutatni. Ezen adatmodellben már lehetséges az N:M kapcsolatok szemléltetése is.
* az **objektumorientált** adatmodell:Az objektumorientált adatmodell az objektumorientált programozás módszertanának egy része. Olyan adatbázis, amely intelligens elemekből épül fel, azaz objektumok az adatokat és a hozzájuk tartozó műveleteket egy egységbe foglalják (lényegében „tudják”, hogy kik ők, mire használhatók, s miként kapcsolódnak a többi adatbáziselemhez). Az elemeket jellemzi az öröklődés, a polimorfizmus és adat absztrakció jellemzi (azaz a felhasználók csak a lényeges információkat látják, míg a részletek rejtve maradnak előttük). Az objektumorientált adatbázisok az objektumorientált programozási nyelvek térhódításával terjedtek el, de hatékonyságban még alulmaradnak a relációs adatbázisokkal szemben.
* a **relációs** adatmodell szerkezetét alapvetően az adatok tulajdonságai szabják meg. Olyan adatmodell, amelynek legfontosabb eleme a matematikai reláció fogalma. A relációs adatbázisban az adatokat táblákban tároljuk. A táblák kapcsolódhatnak egymáshoz, így a tábla a matematikában használt reláció egy lehetséges megvalósítási formája. Innen az adatmodell neve is. Az adatelemek megnevezett, összetartozó csoportjából kialakított táblázatot relációnak nevezzük. A tábla sorokból és oszlopokból áll, az oszlopok pedig egy-egy tulajdonságot írnak le. Ily módon a relációs adatmodell egy kétdimenziós táblázat. Ugyanakkor a relációs adatbázis csak logikai viszonyokat tartalmaz, az adatok tényleges tárolásának „megszervezése”, a folyamatok gyakorlati lebonyolítása nem feladata.

### 11.1 Tervezés

Projektünkben mi is a relációs adatbázis modell alkalmazását követtük, így a továbbiakban csak ezzel foglalkozunk. A következőkben a Kardos Zoltán (2008: 8.) által bemutatott adatbázis tervezési modellen keresztül mutatjuk be a mi ***nonogram*** adatbázisunk tervezését.



31. ábra: Az adatbázis tervezésének lépései (Kardos 2008: 11)

1. *Követelményelemzés*: Ebben a lépésben kell meghatározni a megoldandó feladatot, definiálni az adatbázis elkészítésének paramétereit. Tulajdonképp a mit és hogyan meghatározása történik itt. A mi esetünkben az adatbázis keretfeltételei adottak voltak, ugyanis bár a választott alkalmazás megvalósítható lett volna adatbázis nélkül is, kikötés volt, hogy egy legalább 3 táblát tartalmazó adatbázisra épüljön a vizsgaremek. Mivel pedig a projekt maga egy Nonogram puzzle, így adta magát, hogy kell egy táblát létrehoznunk a játékosoknak vagy felhasználóknak, és kell egy táblát létrehozni a megoldandó képeknek is. E két entitástípusról mindenképp kell adatokat tárolni és létrehozni. Ezen felül kellenek segítségek is, amelyeket a képek kitöltése során igénybe vehetnek a felhasználók. Ezen kívül nyilván kell tartanunk a már kitöltött képeket, és a még folyamatban lévő képeket is.
2. *Egyedek, táblák meghatározása*: Itt történik az összegyűjtött adatok rendszerezése, azon információk keretrendszerének meghatározása, ahol ezeket használni fogjuk, az egyedtípusok meghatározása, a redundancia elkerülése. Az imént említett 3 entitás-típus tehát külön táblákat fog kapni, az egyiket fogjuk **USER** táblának nevezni – itt tárolódnak a felhasználókhoz kapcsolódó adatok, a másodikat **IMAGE** táblának – ebben a képekkel kapcsolatos információkat fogjuk szerepeltetni, míg a harmadik tábla, a **HELP** tábla a segítségek különböző típusait fogja tartalmazni. Ezen egyedek közötti kapcsolatokat kell modelleznie az adatbázisunknak.
3. *Attribútumok, mezők meghatározása*: Itt történik az egyes táblák tényleges meghatározása, a bennük tárolandó attribútumok meghatározása, és a bennük kötelezően szerepló mezők kijelölése. Lényegében az egyed-típusok konkrét definiálása. Meg kell határoznunk az attribútumokat vagy tulajdonságokat, melyek lehetnek egyszerűek (egy részből áll, és tovább nem bontható) és összetettek (több részből tevődik össze); lehetnek egyértékűek (egy előfordulásnál csak egy értéke van) és lehetnek többértékűek (több értéket is felvehet egy előfordulásnál). Minderről részletesebben olvashatunk Juhász Tibornál (2005: 146-7.). Az előző pontban meghatározott egyedek vagy entitások esetében a mindegyikre találhatunk példát. A USER neve például összetett, hiszen van egy családneve és egy keresztneve is, de például az e-mail címe egyszerű, hiszen csak egyetlen adatból áll. Ugyanez lesz igaz a HELP attribútumaira, hiszen az, hogy milyen típusú, és mennyibe kerül, mennyi ponttal csökkenti az elérhető pontszámot egyszerű adat. Az IMAGE tábla minden adata egyszerű, hiszen a kitöltendő kép tartalma maga nem változik, legfeljebb az, hogy éppen milyen kitöltési állapotban van, azonban ez az adat is mindig a USER-hez rendelve jelenik csak meg. A USER azonban rendelkezik egyértékű és többértékű tulajdonságokkal is, hiszen az azonosító adatai egyértékűek (ha változnak is, akkor is csak egy értéket vehetnek fel), de már az, hogy mely képeket oldotta meg egy többértékű tulajdonság, ahogyan az is, hogy milyen HELP-ekkel rendelkezik.
4. *Az azonosítók meghatározása*: Noha neve szerint ebben a lépésben az egyes táblák azonosítóinak, vagy kulcsainak a megadása történik, lényegében ez a felvett táblák közötti relációk kialakítását is jelenti, melyek az egyes táblák kulcsaitól függnek, illetve ezek segítségével adhatók meg a táblák kapcsolódásai. Mivel a táblák egyes sorai (a rekordok) beazonosítása a cél – csak így köthetjük össze a táblákat egymással – azért e feladatra azonosítókat kell kijelölni. Így nem kell minden tulajdonságot leírnunk egy rekord azonosításához, hanem csupán egyet, vagy néhányat. Álljon itt néhány definíció:

**„Kulcs**: azoknak a tulajdonságoknak (mezőknek) a legszűkebb halmaza, amelyek minden sort (rekordot) egyértelműen meghatároznak.

**Elsődleges tulajdonság**: szerepel a kulcsban. [nevezzük elsődleges kulcsnak is]

**Másodlagos tulajdonság**: nem szerepel a kulcsban.

**Egyszerű kulcs**: egyetlen tulajdonságból (mezőből) áll.

**Összetett kulcs**: több tulajdonság (mező) alkotja.” (Juhász 2005: 149.)

Minden táblának kell legyen elsődleges kulcsa, és ez így van a mi tábláinkkal is: a USER tábla elsődleges kulcsa a UserName lett, az IMAGE tábla elsődleges kulcsa a ImageId lett, a HELP tábla rekordjainak elsődleges kulcsa pedig a TypeOfHelp lett. Ezek mind egyszerű kulcsok, de végül az adatbázisunk tartalmazott olyan táblát is (USERIMAGE), amelyben összetett kulcsot használtunk, az egyes rekordokat a USER tábla és az IMAGE tábla azonosítóinak kombinációja határozza meg.

1. *A kapcsolatok meghatározása*: Itt kell megadni az elsődleges kulcsokat, a kiemelten fontos mezőket, illetve a táblák relációban álló rekordjainak az összekapcsolása, melyhez nemritkán külső vagy idegen (*foreign key*) kulcsokat kell használnunk. A mi adatbázisunk esetében a USERIMAGE tábla azonosítóinak mindegyike idegen kulcs lett, ugyanis sem a UserName, sem pedig az ImageId nem a USERIMAGE tábla tulajdonsága. Részben ugyanez lett a sorsa a USERHELP táblának is mely az egyes felhasználókhoz tartozó segítségeket tartalmazza, és csupán azért lett külön tábla, mert többértékű tulajdonságról van szó, amelyet az adatbázis-normalizáció miatt le kellett választani a USER tábláról.
2. *Ellenőrzés*: A táblák, mezők, kapcsolatok definiálása és elkészítése után ellenőriznünk kell, hogy minden fontos szempontot, adatot, relációt szerepeltetünk-e? A mi esetünkben ez még viszonylag késői stádiumban is előfordult, és módosítanunk kellett az adatbázis tábláit. Ilyen volta például az, amikor a HELP táblába utólag bekerült a Weight tulajdonság, amivel eredetileg nem számoltunk, de egyszerűbbé tette az adott kép megoldása után kapott jutalmak súlyozását, hogy nem kellett külön definiálni ezt, hanem a tábla rekordjai már tartalmazták azt. Ugyanez történt az IMAGE tábla két attribútumával a RowFinished és ColumnFinished tulajdonságokkal, melyeket azért iktattunk be, hogy az egyes képek megnyitásakor ne kelljen minden sorra és oszlopra lefuttatni azt a keresést, mely megnézi, hogy vannak-e olyan sorok és oszlopok, melyek kitöltésében nincs szabadsága a felhasználónak, mert azokat csak egyféleképp lehet kitölteni (a játék ezeket a megnyitáskor alapból kitölti).
3. *Adatbevitel és további objektumok létrehozása*: Az ellenőrzés és javítás után feltölthetjük a táblákat adatokkal. illetve elkészíthetjük a lekérdezéseket, melyekkel az adatbázisunk adataihoz férünk hozzá, illetve módosítjuk azokat. A mi programunkban mind az adatbázis létrehozást, mind a táblák létrehozását, azok feltöltését és módosítását is egyetlen DBManager.cs osztály végzi.

Az eredetileg tervezett adatmodellünkön több változtatást is végre kellett hajtani így az eredeti ERD diagrammunk (32. ábra) nem teljesen egyezik meg a végleges adatbázis struktúrával (33. ábra), noha az alapvető szerkezetén nem sok módosítást kellett végrehajtani. És ezeknek a változtatásoknak is alapvetően normalizálási okai vagy épp programozási okai voltak.

A képen diagram, rajz, vázlat, minta látható

Automatikusan generált leírás

32. ábra: Eredeti ER diagram

A képen rajz, vázlat, diagram, Vonalas grafika látható

Automatikusan generált leírás

33. ábra: A végső ER diagram

### 11.2 Normalizálás

Az adatbázis-normalizálás az adatbázisok tervezésének egyik alapvető folyamata, amelynek célja a redundancia csökkentése és az adatok konzisztenciájának biztosítása. A normalizálás során az adatokat táblákba rendezzük, és kapcsolatokat hozunk létre e táblák között, miközben szigorú szabályokat alkalmazunk. Ennek eredményeként elkerülhető az adatok felesleges ismétlődése (redundancia), csökkenthető a tárhelyigény, és minimalizálható a karbantartási hibák lehetősége. Ha ugyanis egy információ több helyen is előfordul, akkor annak frissítése bonyolultabbá válik, hiszen minden előfordulást egyformán kell módosítani, ellenkező esetben ellentmondásos (inkonzisztens) adatok keletkezhetnek.

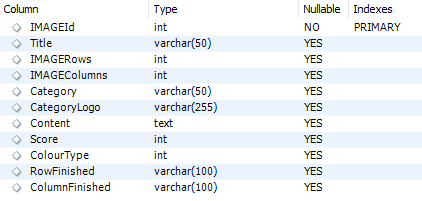
A normalizálási folyamat több **normálformából** áll, amelyeket egymás után alkalmazva elérhető egy optimális adatszerkezet. Végső soron ez egy adatbázis optimalizálás. Az első három normálforma az alapvető követelményeket fogalmazza meg: az **1NF** biztosítja, hogy minden attribútum atomi értéket tartalmazzon; a **2NF** kiküszöböli a részleges függőségeket; míg a **3NF** megszünteti a tranzitív függőségeket (a másodlagos tulajdonságok között fennálló funkcionális függőségeket), ezáltal tovább csökkentve a redundanciát. Az adatbázisokat általában legalább **harmadik normálformába** (3NF) kell hozni, hogy elkerüljük a felesleges adat-duplikációt és biztosítsuk a hatékony adatkezelést.

A megfelelően normalizált adatbázisok előnye, hogy rugalmasabbak, könnyebben módosíthatók, és kevesebb hibalehetőséget hordoznak magukban. Ugyanakkor túlzott normalizálás esetén a sok táblaközötti kapcsolat miatt az adatok lekérdezése bonyolultabbá válhat, ezért a normalizálást mindig az adott rendszer igényeinek megfelelően kell alkalmazni. Így mindenképp optimális szintre kell hozni a normalizálást.

A nonogram alkalmazás adatbázisának normalizálásának a menete azalábbiak szerint alakulhatott volna, ha bizonyos táblák már nem lettek volna a megfelelő normálformában tervezéskor is. A bemutatás során végig megyünk az egyes normálformákon és az adatbázis tábláin is:

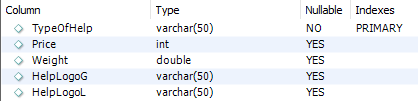
1. *első normálforma*
   1. összetartozó adatok egy táblába kerüljenek;
   2. az egyedi adatelemeket kulccsal jelöljük meg;
   3. szűrjük ki az ismétlődő adatokat.

Ennek megfelelően kell egy IMAGE tábla a megoldandó képek adatainak, kell egy USER tábla a játékosok adatainak, és kell egy HELP tábla a kitöltés során igénybe vehető segítségeknek. Az egyedi azonosítók ennek megfelelően az IMAGE táblában egy „*ImageId*” lehetne, a USER táblában a *„UserName”*, hisz ez egyedi, a HELP táblában pedig a segítségek típusai („*TypeOfHelp”*). Az IMAGE tábla önmagában létrehozható, hiszen idegen kulcsot nem kell tartalmaznia, és módosítani sem kell a játék során. Ez különösen fontos, ugyanis az IMAGE (34. ábra) tábla elsődleges kulcsa automatikusan előállított tulajdonság, és más táblák is használják idegen kulcsként.



34. ábra: Az IMAGE tábla

A HELP (35. ábra) tábla a segítségek típusait, az árát, annak súlyozását, hogy milyen valószínűséggel nyerhető meg, illetve a mouseover és ischecked effektusok által használ képek nevét tartalmazza minden egyes típus esetében:



35. ábra: A HELP tábla

A USER tábla már nem ennyire egyszerű ugyanis abban rögzíteni kell azt, hogy melyik segítségtípusból mennyi áll rendelkezésre a játékos számára, azt azonban már nem kell rögzíteni a USER táblában, hogy ezek mennyibe kerülnek, és milyen hatással vannak az épp kitöltött képre. Ugyanakkor az 1NF-nek megfelelne az is, ha a HELP tábla elsődleges kulcsait további attribútumokként (oszlopokként) betennénk a USER táblába.

A megoldott képeket is a felhasználóhoz csatoltan kell tárolni, azonban egy-egy felhasználóhoz biztosan több megoldott kép tartozik, így ez nem elégítené ki az első normálforma 3. kívánalmát. A megoldás az, hogy a megoldott képeket és azok állapotát, ha félbehagyott, akkor a félbehagyott állapotát egy külön táblában kell tárolni, amelynek egy sorában a „*UserName*”, a kép „*ImageId*”, a befejezettség és a pillanatnyi tartalom szerepel, az előbbi kettő összetett kulcsként meghatározva minden rekordot:

|  |  |
| --- | --- |
| 36. ábra: A USERIMAGE tábla | . ábra: A DBManager.cs sql string-je |

A fenti tábla (36. ábra) mutat némi eltérést ahhoz képest, ahogyan a DBManager.cs (37. ábra) létrehozza azt. Ugyanis egyrészt a MySQL tinyint(1)-ként interpretálja a BOOLEAN adattípust, másrészt a MySQL Workbench, ahonnan a tábla képe származik, automatikusan hozzáad egy (KEY IMAGEId (IMAGEId)) indexet annak érdekében, hogy optimalizálja az „*ImageId*” oszlopot érintő kéréseket.

Ha tehát a fenti táblázatokat létrehoznánk, de a USER táblában továbbra is szerepeltetnénk az 1NF-nek megfelelően a játékos rendelkezésére álló helpjeit, akkor az valahogy így nézne ki:



38. ábra: Az 1NF-nek megfelelő USER tábla

1. *második normálforma* (**2NF**):
   1. első normálformában van;
   2. és minden nem elsődleges mező (nem kulcs attribútum) teljesen függ a kulcstól.

Az 1NF alapján összeállított tábláink megfelelnek a 2NF-nek is, hiszen mindegyik tábla oszlopai teljesen a kulcstól függnek. Például a *„FullName”* attribútum közvetlenül a *„UserName”*-től függ, mivel a *„FirstName”* és *„LastName”* is közvetlenül a *„UserName”*-től függ. Tehát a *„FullName”* megfelel a 2NF-nek. Ugyanakkor felmerülhet a gyanú, hogy mégsem teljes mértékben elégíti ki a 2NF-t, hiszen az 1H…Erase attribútumok a HELP tábla elsődleges kulcsai, tehát attól is függni látszanak. Ez azonban nincs így, hiszen ezen oszlopok értékei teljesen a „UserName”-től függnek, ugyanis csak az egyes segítségtípusok darabszáma (ami a felhasználó rendelkezésére áll) van eltárolva bennük. Ugyanakkor felvethető, hogy túl sok attribútum kerül bele a USER tábla rekordjaiba ezzel, így célszerűbb lenne külön táblába rendezni ezeket. Ennek a táblának az egyedi kulcsa a „UserName” és minden segítség egy-egy oszlop lesz benne, és az ott szereplő szám adja meg, hogy a felhasználónak melyik segítségből mennyi áll rendelkezésére. Ebben a táblában lehet növelni az értéket vásárláskor, az összeget a USER tábla Tokens oszlopából levonni a HELP táblában tárolt ár alapján. Tehát ez a USERHELP tábla (39. ábra) így nézne ki:



39. ábra: A USERHELP tábla

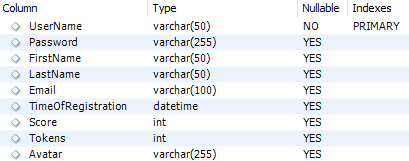
1. *harmadik normálforma:*
   1. második normálformában van,
   2. nem tartalmaz tranzitív függést.

A USER tábla, amiről leválasztottuk a USERHELP táblát így nézne tehát most ki (40. ábra):



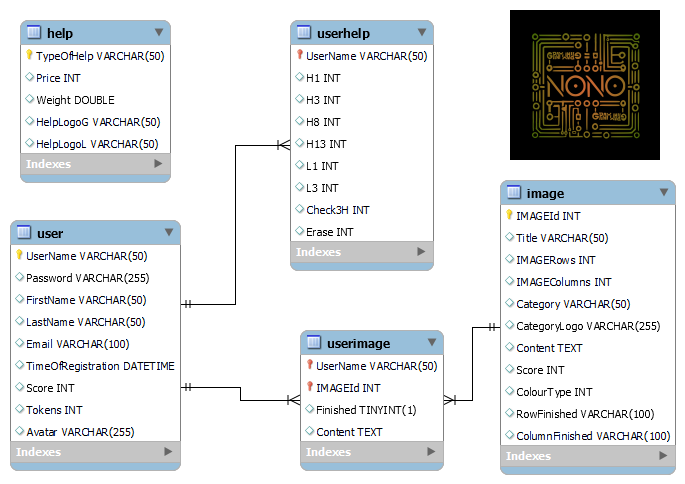
40. ábra: A módosított és 2NF-nek megfelelő USER tábla

Ez a tábla azonban nem felel meg a 3NF-nek, mert a 3NF megköveteli, hogy minden nem kulcs attribútum közvetlenül az elsődleges kulcstól függjön, és ne függjön más nem kulcs attribútumtól. Mivel a *„FullName”* függ a *„FirstName”* és *„LastName”* attribútumoktól, ez egy tranzitív függőséget hoz létre, ami megsérti a 3NF-et. Ez az adat azonban nyilvánvalóan redundáns is, hiszen semmi mástól nem függ, csupán a „*FirstName”* és *„LastName”* attribútumoktól, így egyáltalán nincs is szükség ezen adat eltárolására. Ha kivesszük a kérdéses attribútumot a USER táblából, akkor az már megfelel a 3NF-nek is, így az adatbázis minden táblája megfelel a harmadik normál formának. Az új USER tábla (41. ábra) tehát végül így néz ki:



41. ábra: A USER tábla

A végleges és a 3NF-nek megfelelő adatbázisunk adatbázismodell-diagramja tehát így fog kinézni (42. ábra):



42. ábra: A nonogram adatbázismodell-diagramja

Még néhány megjegyzést tennénk az egyes táblákban található adatokról és adattípusokról:

**USER** tábla:

* **UserName**: a felhasználó által regisztrációkor választott felhasználónév. Nincsen semmiféle megkötés vele kapcsolatban, csupán annyi, hogy legfeljebb 50 karakter hosszúságú lehet.
* **Password**: a jelszőt is a felhasználó választja, majd ezt egy SHA256 kódolás után stringként tároljuk.
* **FirstName**, **LastName**, **Email:** ezekre is csak hosszúsági megkötés van érvényben.
* **TimeOfRegistration**: DATETIME formátumú adat, és a regisztráció idejét adja meg.
* **Score** és **Token**: a felhasználó összesített pontszámát és tokenjainak mennyiségét adják meg, egész számként tárolva.
* **Avatar**: ez alapvetően egy fejlesztési lehetőséget előlegez meg, ti. azt, hogy a felhasználó megváltoztathassa a ***Gravatar*** által biztosított és sok más cég (pl. GitHub) által is alkalmazott szolgáltatáson alapuló (e-mail cím alapján véletlenszerűen generált) képét. Így ez az attribútum a felhasználó képfájljának nevét tartalmazná.

**IMAGE** tábla:

* **ImageId**: a megoldandó kép automatikusan előállított azonosítója, a tábla elsődleges kulcsa.
* **Title**: a megoldandó kép címe stringként tárolva.
* **ImageColumns, ImageRows:** a megoldandó kép méretét adják meg (mivel olykor összeveszett az SQL szintaxissal a korábbi Rows és Columns elnevezés, megváltoztattuk őket. Egész számként tárolt adatok.
* **Category, CategoryLogo**: Az ImageListView-ban megjelenített adat a kategória, és az ehhez tartozó logo képfájl neve stringként tárolva.
* **Content**: ez a legfontosabb tulajdonsága minden képnek, ugyanis ez tartalmazza a kép tartalmát egy soronkénti leolvasással kapott string formátumában. Mivel a tartalom string, így nemcsak fekete-fehér, hanem más színeket is könnyen tárolhatunk ugyanezen adattípusban. Bár ez is csak egy további fejlesztési lehetőség.
* **Score**: a képhez tartozó pontszám, amit a felhasználó megnyer, ha megoldja a képet (egész számként tárolva).
* **ColourType**: ez is előre beépített fejlesztési lehetőség. Most ugyan csak fekete-fehér képek szerepelnek a játékban az egész szám változtatásával könnyen tárolhatjuk, hány színt is használ egy kép.
* **RowFinished, ColumnFinished**: e két string később került bele a táblába, hogy a megnyitáskori kitöltési ellenőrzést hatékonyabban hajthassuk végre. Ez adja meg, melyik sorokat kell (1) ellenőrizni és kitölteni, és melyeket nem kell (0).

**USERIMAGE** tábla:

* **UserName**: az egyik elsődleges kulcs a USER táblából (idegen kulcs). A felhasználóhoz rendeli a kitöltött és félbehagyott képeket.
* **ImageId**: azt adja meg melyik képről is van szó, szintén elsődleges és szintén idegen kulcs az IMAGE táblából.
* **Finished:** egy boolean típusú adat, ha befejezett a kép, akkor igaz, ha félbehagyott, akkor hamis.
* **Content**: Amennyiben befejezett a kép, egy x szerepel itt, amennyiben félbehagyott, akkor a félbehagyott kitöltés egy string formájában, az IMAGE tábla Content attribútumához hasonlóan, de egyéb karakterekkel dúsítva (pl. x jelzi azt, hogy az adott mező még „szűz”, azaz sem üresnek, sem telinek nincs jelölve, sem pedig „?”-lel megjelölve).

**USERHELP** tábla:

* **TypeOfHelp**: a felhasználható segítség típusát adja meg. Elsődleges kulcs.
* **Price**: a segítség tokenben kifejezett árát adja meg, ennyiért lehet vásárolni a BuyHelpView-ban.
* **Weight:** Mivel a képek befejezésekor nemcsak pontot lehet szerezni, hanem véletlenszerűen segítségeket is, noha minél nagyobb értékű a segítség és minél kevesebb pontot ér a megoldott kép annál valószínűtlenebb a megnyerése. Az ehhez szükséges súlyozást tartalmazza ez a mező DOUBLE formátumban.
* **HelpLogoG, HelpLogoL**: A segítségek és azok mouseover és ischecked tulajdonságainak vizuális megjelenítéséhez szükséges képfájlok nevei string formátumban tárolva.

### 11.3 Adatvédelem (GDPR és SQL-injektálás)

Mivel a felhasználók bizonyos adatait használjuk, illetve tároljuk, ezért az alkalmazásban történő regisztrációkor a felhasználónak bele kell egyezni, hogy elfogadja az adatkezelési tájékoztatót. Ugyanakkor az alkalmazás csak nagyon kevés, és nem különösebben érzékeny adatot használ (mindössze a felhasználó nevét és e-mail címét, amelyeke közül egyiket sem ellenőrizzük), így nincs szükségünk egy teljes GDPR[[1]](#footnote-1) adatvédelmi tájékoztatóra. A szükséges beleegyezést regisztrációkor kell megadni, az Adatvédelmi Tájékoztatót pedig pdf formátumban lehet letölteni és elolvasni.

Nemcsak a GDPR-re kell kitérnünk az adatvédelem kapcsán, hanem az adatbázisműveletekre leselkedő legnagyobb veszélyre, az SQL-injektálásra is.

“Az SQL-injektálásos támadások alighanem a legveszélyesebb olyan támadások, amelyek érvénytelen adatokat használnak. A támadó az SQ-injektálással egy SQL-utasítást ágyaz be az űrlapmezők adataiba, abban reménykedve, hogy az alkalmazás ellenőrzés nélkül továbbítja az adatokat az adatbázisszervernek, és hogy az adatbázisszerver futtatni is fogja. A beágyazott SQL-utasításáltalában valamilyen rosszindulatú műveletet hajt végre, például töröl egy táblát, a táblában található összes adatot.” (Blum 2020: 390.)

A NonoGram alkalmazás védve van az SQL-injektálásos támadások ellen, mert a kód paraméterezett lekérdezéseket (parameterized queries) használ az adatbázis műveletekhez. A paraméterezett lekérdezések lehetővé teszik azt, hogy az SQL parancsokat és az adatokat külön kezeljük, így az adatokat nem közvetlenül az SQL parancs részeként adjuk át. Ez megakadályozza, hogy a felhasználói bemenetek manipulálják az SQL parancsokat, mivel a bemeneteket az adatbázis motor külön kezeli, és nem értelmezi őket SQL parancsként.

Például a DbManager osztályban található metódusokban a paramétereket külön adják meg a lekérdezésekhez, mint például a MySqlCommand.Parameters.AddWithValue (43. és 44. ábra) metódus használatával. Ez biztosítja, hogy a felhasználói bemenetek nem tudnak kártékony SQL parancsokat végrehajtani, mivel az adatbázis motor automatikusan megvédi a bemeneteket az SQL injection támadások ellen. Ezenkívül a bemenetek validálása és szűrése további védelmet nyújt az ilyen típusú támadásokkal szemben.

A képen szöveg, képernyőkép, szoftver látható

Automatikusan generált leírás

43. ábra: A DBManager.cs ExecuteQuery metódusa

A képen szöveg, Betűtípus, szám, képernyőkép látható

Automatikusan generált leírás

44. ábra: User feltöltése a USER táblába a RegisterVM-ben

## **12. Részletes feladatspecifikáció, algoritmusok**

A Nonogram játék fejlesztése során célunk volt, hogy egy jól strukturált, könnyen karbantartható és bővíthető alkalmazást hozzunk létre. Az alábbiakban bemutatjuk a részletes feladatspecifikációt és az alkalmazott algoritmusokat, amelyek hozzájárultak a projekt sikeres megvalósításához.

### 12.1 Feladatspecifikáció

A Nonogram egy logikai játék, amelynek célja, hogy a játékos kitöltse a rácsot a sorok és oszlopok melletti számok alapján. A játék fejlesztése során több kulcsfontosságú követelményt határoztunk meg:

* **Felhasználói Felület (UI):** A játék felhasználói felületét intuitív és felhasználóbarát módon kellett kialakítani. A felületnek támogatnia kellett különböző képernyőméreteket és felbontásokat, hogy asztali számítógépeken és mobil eszközökön egyaránt jól használható legyen.
* **Interaktív Elemek:** A játék elemeinek interaktívnak kellett lenniük, lehetővé téve a játékosok számára a rács celláinak kattintásos kitöltését.
* **Adatkezelés:** A játék során a játékos által elért eredmények és a játékállapot tárolása és visszatöltése fontos szempont volt. Az adatokat biztonságosan és hatékonyan kellett kezelni.

A játék fejlesztése sajnos akkor kezdődött, amikor még nem voltunk a megfelelő ismeretek birtokában ahhoz, hogy világosan végig gondolhassuk a játék végső funkcionalitását, és annak megvalósítási módját. A fő UI megvalósításához a kiindulópontunk egy **WPF C# Professional Modern Flat UI Tutorial** (Payload 2021) című Youtube videóból származott, melynek egy részét felhasználtuk a NonoGram alkalmazás alap UI interfészéhez is. Ez sajnos egy sokkal egyszerűbb template, mint amire nekünk szükségünk volt végül, és ennek köszönhetően maradtak egyenetlenségek, redundanciák és inkonzisztenciák a projekt jelenlegi verziójában is. Tudatában vagyunk annak, hogy a CleanCode értelmében egységes MVVM kezelést kellene megvalósítania az applikációnak, miközben az előbb elkészült fő UI interfész ViewModel logikája nem úgy épül fel, mint a később implementált bejelentkezési modulé, amelynek UI logikáját egy ViewModelFactory osztály vezérli. Természetesen summary-ket is érdemes lett volna írnunk a kódba, sokkal nagyobb számban, mint most van, de a fejlesztési idő rövidsége miatt még a legutolsó napokban is dolgoztunk a kódon.

A fejlesztés során - a rendelkezésre álló információk hiányossága és a képzés, talán kevésbé átgondolt struktúrájának köszönhetően későn vagy egyáltalán nem feldolgozott ismeretköreinek köszönhetően - kénytelenk voltunk segítségül hívni a mesterséges intelligencia segítségét is, mely sokszor azért nem annyira segítségként funkcionált, mintsem félrevezetésként. Azonban összességében azt hiszem a Copilot, a ChatGPT és társaik segítsége nélkül sokkal nehezebben haladt volna a fejlesztés. Nagy segítséget jelentett a fejlesztés során rengetegszer igénybevett hibakeresésben és az ötletek tesztelésében. Ahogyan a “favágó” munkában is. Sajnos a legnagyobb gondokat okozó DataContext instanciálásokban és az adatkötésekben korántsem volt olyan nagy segítségre ez a forrás, mint arra előzetesen számítottunk.

## **13. Forráskód**

A Nonogram játék fejlesztése során számos algoritmust alkalmaztunk a játék logikájának és működésének biztosítására. A fontos részeket kétfelé tudjuk osztani az egyik a bejelentkező és a regisztrációs felület a másik pedig a fő ablak ami a bejelentkezés után jelenik meg. Ezen részek néhány főbb forráskódjait Szeretnénk elmagyarázni és bemutatni.

### 13.1 LoginViewModel.cs

Ez az osztály a bejelentkezési folyamatot kezeli az MVVM (Model-View-ViewModel) architektúrában. A LoginViewModel () osztály a felhasználói bejelentkezést, regisztrációt és jelszó-emlékeztetőt kezeli, miközben biztonságosan kezeli a felhasználói adatokat.

*Tulajdonságok*

* UserName: A bejelentkezett felhasználó nevét tárolja.
* IsLoginSuccessful: Tárolja, hogy a bejelentkezés sikeres volt-e.

*Parancsok*

* LoginCommand: A bejelentkezés végrehajtására szolgál.
* RegisterCommand: A regisztrációs nézet megjelenítésére szolgál.
* ForgotPasswordCommand: A jelszó emlékeztető nézet megjelenítésére szolgál.

#### 

*Konstruktor*

A konstruktor inicializálja a parancsokat a megfelelő műveletekkel és beállítja a IsLoginSuccessful tulajdonságot hamis értékre.

A képen szöveg, képernyőkép, Betűtípus, sor látható

Automatikusan generált leírás

45. ábra: A LoginVM

*Parancs Műveletek*

* + - 1. **ShowRegisterView**

Ez a metódus a ViewModelFactory segédosztály segítségével megjeleníti a regisztrációs nézetet. (46. ábra)

A képen szöveg, képernyőkép, Betűtípus, sor látható

Automatikusan generált leírás

46. ábra: ShowRegisterView

1. **ShowForgotPasswordView**

Ez a metódus megjeleníti a jelszó emlékeztető nézetet, szintén a ViewModelFactory segédosztály segítségével. (47. ábra)

A képen szöveg, képernyőkép, Betűtípus, sor látható

Automatikusan generált leírás

47. ábra: ShowForgotPasswordView

1. **Login**

Ez a metódus kezeli a bejelentkezési folyamatot. (48. ábra)

A képen szöveg, képernyőkép, dokumentum látható

Automatikusan generált leírás

48. ábra: Login

### 13.2 DBManager.cs

Az adatbázis kapcsolatot a DbManager osztály kezeli, amely végrehajtja az SQL lekérdezéseket. Főbb funkciói és feladatai a következők:

* + **Adatbázis inicializálása és táblák létrehozása**: Az InitializeDatabaseAndTables létrehozza az adatbázist és a szükséges táblákat, valamint opcionálisan feltölti azokat CSV fájlokból. A CreateDatabase létrehozza az adatbázist, ha az még nem létezik. A CreateTables ugyanezt teszi a táblákkal. A GetTableCreationSql visszaadja az SQL parancsokat a táblák létrehozásához.
  + **Táblák feltöltése CSV fájlokból**: A PopulateTablesFromCsv feltölti a táblákat a megfelelő CSV fájlokból. Az IsTablePopulated ellenőrzi, hogy egy tábla már fel van-e töltve adatokkal. Az InsertRecordFromCsvLine beszúr egy rekordot a megadott táblába egy CSV sor alapján.
  + **SQL parancsok végrehajtása**: Az ExecuteNonQuery hajtja végre a nem lekérdezés típusú SQL parancsokat (pl. INSERT, UPDATE, DELETE). Az ExecuteQuery dig a lekérdezés típusú SQL parancsokat, és visszaadja az eredményt egy DataTable objektumban.
  + **Adatok exportálása CSV fájlokba**: ExportTableToCsv exportálja a megadott táblát egy CSV fájlba, ezzel téve lehetővé az adatbázis műveletek elmentését a későbbi munkamenetekre.
  + **Adatok lekérdezése és manipulálása**: A GetImageById lekérdezi egy kép adatait az IMAGE tábla alapján azonosítóval. Az AddParameters hozzáadja a paramétereket egy MySqlCommand objektumhoz.

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

49. ábra: Példa a DBManager metódusaira

### 13.3 HashHelper.cs

A **HashHelper** osztály a jelszó hash-értékének kiszámítására és ellenőrzésére szolgál.

A képen szöveg, képernyőkép, szoftver, képernyő látható

Automatikusan generált leírás

50. ábra: A HashHelper segédosztály

### 13.4 MainViewModel Osztály

A MainViewModel osztály az egyik legfontosabb része a programnak, mert központi szerepet játszik az alkalmazás működésében és felhasználói élményében. Az alábbiak miatt kiemelkedően fontos:

1. **Nézetek és nézetmodellek kezelése**: A MainViewModel felelős az alkalmazás különböző nézeteinek és nézetmodelljeinek koordinálásáért és kezeléséért. Biztosítja, hogy a megfelelő nézetmodellek jelenjenek meg a felhasználói interakcióknak megfelelően.
2. **Felhasználói adatok kezelése**: A felhasználói adatok betöltése és kezelése is a MainViewModel feladata, így biztosítja, hogy a bejelentkezett felhasználó megfelelő adatokkal rendelkezzen az alkalmazás használata során.
3. **Parancsok és interakciók kezelése**: A MainViewModel különböző parancsokat és interakciókat kezel, amelyek lehetővé teszik a felhasználóknak, hogy navigáljanak az alkalmazás különböző részei között, és végrehajtsák a szükséges műveleteket (pl. játék indítása, segítség kérése).
4. **Adatbázis integráció**: Az osztály felelős az adatbázissal való kapcsolat létrehozásáért és kezeléséért, ami lehetővé teszi a felhasználói adatok betöltését, frissítését és mentését.
5. **Felhasználói élmény javítása**: A MainViewModel biztosítja, hogy a felhasználói élmény gördülékeny és intuitív legyen, mivel kezeli a nézetek közötti navigációt, a felhasználói adatokat és az interakciókat.

Ez az osztály felelős a Nonogram játék fő nézetének kezeléséért az MVVM (Model-View-ViewModel) architektúrában. A MainViewModel osztály koordinálja a különböző nézetmodelleket és parancsokat, amelyek a felhasználói interakciókat kezelik.

#### *Tulajdonságok*

* UserName: A bejelentkezett felhasználó nevét tárolja.
* IsHelpEnabled: Tárolja, hogy a súgó funkció engedélyezve van-e.
* CurrentViewMain: Az aktuálisan megjelenített fő nézet objektuma.
* CurrentViewTitle: Az aktuálisan megjelenített cím nézet objektuma.
* CurrentViewHelp: Az aktuálisan megjelenített súgó nézet objektuma.
* AvatarUrl: A felhasználó avatarjának URL-je.
* ZoomLevel: A nagyítási szint értéke.
* User: A felhasználói adatok objektuma.

#### *Parancsok*

* ImageListViewCommand, BuyHelpViewCommand, GameViewCommand, UserMenuViewCommand: ICommand interfészek, amelyeket a megfelelő nézetek megjelenítésére és műveletek végrehajtására használunk.

#### *Konstruktor*

A konstruktor inicializálja a nézetmodelleket és a parancsokat, valamint beállítja az alapértelmezett nézeteket és a felhasználói adatokat.

#### A képen szöveg, képernyőkép, dokumentum látható Automatikusan generált leírás

51. ábra: A MainViewModel konstruktora

#### *Parancs Műveletek*

##### OpenGameView

Ez a metódus kezeli a játék-nézet megnyitását. (52. ábra)

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

52. ábra: A Játéknézetet megnyitó metódus

### Adatbázis Kapcsolat (LoadUserData 53. ábra)

Az adatbázis kapcsolatot a DbManager osztály kezeli, amely végrehajtja az SQL lekérdezéseket a felhasználói adatok betöltésére.

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

53. ábra: A MainVM számára fontos user tulajdonságok

### 13.5 Játékállapot Kezelés

A játékállapot kezelése kulcsfontosságú volt a játék folyamatos működésének biztosításához. A játék állapot tartalmazza a rács aktuális kitöltését és a játékos által elért eredményeket. Az állapotot egy adatbázisban tároltuk és szükség szerint visszatöltöttük.

**GameView Felépítése**

* **XAML Felépítés**: A GameView.xaml fájlban definiált UI felület tartalmazza a játék rácsának megjelenítéséhez szükséges elemeket. A fő elemek közé tartozik a Grid, amely tartalmazza a sor- és oszlop-tippek megjelenítésére szolgáló ItemsControl elemeket, valamint a játék rácsának celláit megjelenítő ItemsControl elem. A Grid két sorból és két oszlopból áll, ahol az első sor és oszlop a tippek megjelenítésére szolgál (a Grid.Row="0" Grid.Column="1" mezőben kaptak helyet az oszlop-segítségek és a Grid.Row="1" Grid.Column="0" mezőben a sor-segítségek), míg a második sor és oszlop közös mezője (Grid.Row="1" Grid.Column="1") szolgál a játék rácsának megjelenítésére.
* **Adatkötések és Konverterek**: Az ItemsControl elemek adatkötésekkel kapcsolódnak a GameViewModel megfelelő tulajdonságaihoz, mint például a ColumnTableElements (az oszlop-segítségeket tartalmazó táblázat), RowTableElements (az sor-segítségeket tartalmazó táblázat) és ImageCellTableElements (maga játékrács).

A BoolToBrushConverter és ZeroToVisibilityConverter konverterek segítségével a cellák megjelenése dinamikusan változik a játék állapotának megfelelően. Míg a ZeroToVisibilityConverter felel azért, hogy a segítség-táblák 0-kat tartalmazó cellái ne jelenjenek meg, addig a BoolToBrushConverter felel a highlight megjelenítésért.

Az ImageCellTableElements rács minden cellájának 4 kattintási állapota lehetséges, lehet kitöltetlen (“szűz”), kitöltött, üres, és kérdőjeles. Ezek a kattintásállapotok a mezők ClickState értékéhez vannak kapcsolva adatkötéssel, és így jelenítődnek meg. Lásd 54. ábra.

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

54. ábra: GameView.xaml

**GameView Működése**

* **Adatkötések és Interakciók**: A GameViewModel osztály biztosítja az adatokat és a logikát a GameView számára. Az adatkötések segítségével a GameView elemei automatikusan frissülnek, amikor a GameViewModel adatai változnak.   
  A felhasználói interakciókat, mint például a cellákra való kattintást, a GameView code-behind osztálya kezeli és továbbítja a GameViewModel megfelelő metódusainak, mint például a CheckRowsAndColumns és ExecuteHelpOption metódusoknak.
* **Játékállapot Kezelése**: A GameViewModel tartalmazza a játék logikáját, beleértve a rács inicializálását, a tippek kiszámítását és a játékállapot ellenőrzését.

A GameViewModel metódusai, mint például a CheckRowsAndColumns és CheckIfGameIsWon, biztosítják, hogy a játék megfelelően működjön, és a felhasználói interakciók megfelelően legyenek kezelve.

* **Zoom Funkció**: A GameView támogatja a zoom funkciót, amely lehetővé teszi a felhasználók számára, hogy nagyítsák vagy kicsinyítsék a játék rácsát. Ezt a PreviewMouseWheel eseménykezelő és a ScaleTransform segítségével valósítják meg.

**GameViewModel Felépítése**

A GameViewModel a Nonogram játék központi eleme, ez tartalmazza a játék logikáját, és kezeli a felhasználói interakciókat is. Ennek megfelelően messze ez a legnagyobb és legösszetettebb osztály az alkalmazásban. Az említetteken kívül ez kezeli a segítségek alkalmazását is. Adatkötések és események segítségével biztosítja a játék megfelelő működését és a felhasználói élményt.

* **Tulajdonságok**: A GameGrid tartalmazza a játék-rács adatait és logikáját. A GameGrid osztály példánya, amely a rács celláit, sor- és oszlop-tippek listáit, valamint a játékállapotot kezeli. A ColumnTableElements, RowTableElements, és ImageCellTableElements ObservableCollection típusú tulajdonságok, amelyek a rács oszlopainak, sorainak és celláinak megjelenítéséhez szükséges adatokat tartalmazzák. A RowFinished és ColumnFinished tulajdonságok olyan egyeseket és nullákat tartalmazó tömbök, amelyek a sorok és oszlopok befejezettségi állapotát jelzik. A GuessGrid és a TempList pedig egyaránt kétdimenziós karakterlisták, amelyek a játékos jelöléseit és az ideiglenes állapotokat tárolják.
* **Események**: Az egyetlen fontos esemény az, amikor a játék-rács kitöltésre került, így a GameWon esemény akkor kerül meghívásra, amikor a játékos megnyeri a játékot.
* A GameViewModel konstruktora inicializálja a GameGrid példányát a kiválasztott kép alapján, és beállítja a szükséges tulajdonságokat, mint például a sorok és oszlopok tippjeit, valamint a játékállapotot.

#### GameViewModel Működése

* **Rács Inicializálása**: A GameGrid inicializálása során a rács cellái, sor- és oszlop-tippek kiszámítása és beállítása történik. A DrawTable metódus segítségével a rács elemei megjelenítésre kerülnek a megfelelő ObservableCollection tulajdonságokban.
* **Felhasználói Interakciók Kezelése**: A CheckRowsAndColumns metódus kezeli a felhasználói kattintásokat a rács celláin. A metódus frissíti a GuessGrid állapotát, ellenőrzi a sorok és oszlopok helyességét, és szükség esetén frissíti a RowFinished és ColumnFinished tömböket. Az ExecuteHelpOption metódus kezeli a különböző segítségek alkalmazását, mint például a cellák felfedését vagy a hibás találgatások javítását.
* **Játékállapot Kezelése**: A CheckIfGameIsWon metódus ellenőrzi, hogy a játékos megnyerte-e a játékot azáltal, hogy összehasonlítja a GuessGrid és a GameGrid állapotát. Ha minden sor és oszlop helyesen van kitöltve, a GameWon esemény meghívásra kerül. A SaveGameState metódus menti a játék aktuális állapotát az adatbázisba, hogy a játékos később folytathassa a játékot.
* **Segítségek Kezelése**: Az ExecuteHelpOption metódus különböző segítségeket alkalmaz a játékos számára, mint például a cellák felfedése, sorok és oszlopok ellenőrzése, valamint hibás találgatások javítása. A metódus frissíti a GuessGrid állapotát és ellenőrzi a sorok és oszlopok helyességét.

A továbbiakban néhány a játéknézet működése szempontjából kiemelten fontos osztály és metódus ismertetésére térnénk rá.

**GameGrid.cs**

A GameGrid.cs az osztály a játék rácsának adatait és logikáját reprezentálja, az MVVM rendszer egyik modelljeként funkcionál. Inicializálja a GameGrid példányát a megadott IMAGE objektum alapján. Beállítja a rács méreteit, inicializálja a cellákat, és kiszámítja a sorok és oszlopok tippeit a megjelenítéshez.

Az InitializeGrid metódus inicializálja a rács celláit a megadott tartalom (IMAGE.Content) alapján. A tartalom karakterlánc formátumban van megadva, és a metódus ezt a karakterláncot bontja le sorokra és oszlopokra, így hozva létre egy karaktereket (cellák/oszlopok) tartalmazó listákat (sorok) tartalmazó listát (kép).

A CalculateHints kiszámítja a sorok és oszlopok tippeit (a számokat amelyek a sorok és oszlopok mellett megjelennek) a rács celláinak állapota alapján, méghozzá a CalculateConsecutiveCells metódus segítségével.

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

55. ábra: A CalculateConsecutiveCells metódus

A metódus egy paraméterként kapott stringben (line) található egymást követő '1' karakterek számát számítja ki és adja vissza egy listában. Először létrehoz egy új listát, majd a .Aggregate metódus egy iteratív műveletet hajt végre a line string minden egyes karakterén (c). Az acc az akkumulátor, amely a listát tartalmazza, és amelyet minden iterációban frissít. Ha a karakter '1', akkor az aktuális lista utolsó elemét (acc[acc.Count - 1]) megnöveli eggyel. Ez azt jelenti, hogy egy újabb '1' karaktert találtunk az aktuális sorozatban. Ha a karakter nem '1' és az aktuális sorozat nem üres, akkor hozzáad egy új 0 értéket a listához. Ez azt jelenti, hogy az aktuális egymást követő '1' sorozat véget ért, és egy új sorozat kezdődik. Ezek után kiszűri azokat az elemeket a listából, amelyek az értéke 0, mivel ezek nem relevánsak az egymást követő '1' sorozatok szempontjából. Az eredményt pedig egy listaként adja vissza, mely a RowHints és a ColumnHints listákat fogja alkotni. Ezután a GetRowHints, GetHorizontalColumnHints metódusok hozzák olyan formára a listákat, hogy azok táblázatban megjeleníthetők legyenek, azaz a lista minden lista-eleme ugyanolyan hosszú legyen, ezt ‘0’-kal való feltöltéssel érik el. A GetHorizontalColumnHintsazonban mindezt még vizszintes alakra is kell hozza, hogy a GameViewModel táblázatrajzoló metódusa könnyen ki tudja azt rajzolni.

#### GameViewModel.cs

A GameViewModel konstruktora elsőként inicializál egy felhasználónevet, amit a későbbi műveletek felhasználhatnak. Létrehozza a GameGrid egy példányát a szintén paraméterként megkapott Image objektum alapján, és beállítja az ImageId tulajdonságot is. Ezután inicializálja a három táblázat megrajzolásához szükséges ColumnTableElements, RowTableElements és ImageCellTableElements gyűjteményeket (ObservableObjectCollection). Inicializálja a RowFinished és ColumnFinished tömböket GameGrid befejezettségi állapotai alapján, majd ezt követően inicializálja a GuessGrid és TempList rácsokat is, amelyek a játékos jelöléseit és az ellenőrzéshez szükséges ideiglenes állapotokat tárolják. Aztán a DrawTable metódus segítségével a rács elemei megjelenítésre kerülnek a megfelelő ObservableCollection tulajdonságokban. Ez biztosítja, hogy a rács és a tippek megfelelően jelenjenek meg a felhasználói felületen. Végül a CheckUnfinishedImage metódus ellenőrzi, hogy van-e befejezetlen játékállapot, és ha igen, betölti azt. Ez lehetővé teszi a játékos számára, hogy folytassa a játékot ott, ahol abbahagyta.

A játéknézet egyik legfontosabb metódusa a CheckRowsAndColumns, ami a kattintások során ellenőrzi, hogy az GuessGrid adott mezőben egymást metsző sora és oszlopa a mező kattitntási állapotával (clickState) frissítve kompatibilis-e a kitöltésre váró kép megfelelő sorával és oszlopával. Nagyon fontos megértenünk e metódus működését illetően, hogy nem azt ellenőrzi, hogy a GuessGrid és az ImageCells listák megfelelő elemei megegyeznek-e vagy akár csak azt, hogy kompatibilisek-e. Amit ellenőriz, az az, hogy a sor és oszlop segítségeivel A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás(hints) kompatibilis-e a GuessGrid jelenlegi sora ill. oszlopa. Mindez nem függ magának az ImageCells-nek a tartalmától, csak a ColumnTableElements, RowTableElements tartalmától. Maga a metődus több almetódust hív meg működés közben. Előbb be1rja a megfelelő elemet a GuessGrid-be a clickState alapján, majd leellenőrzi a sort (CheckRow) és az oszlopot (CheckColumn), és ha minden sor megvalósította a RowHints-eket, akkor leellenőrzi, hogy a GuessGrid megegyezik-e az ImageCells-el (CheckIfGameIsWon).

56. ábra A CheckRowsAndColumns metódus

A metódus külön ellenőrzi a sorokat és az oszlopokat, azonban a meghívott metódusok (CheckRow és CheckColumn) működése és felépítése azonos, így ezek is azon kódrészletek sorába illeszkednek, amelyeket egységesítenénk, ha rendelkezésünkre állna még egy kis idő ennek kivitelezésére. Mivel pedig a metódusok működése azonos, most csak a CheckRow (57. ábra) metódust mutatjuk be.

**A képen szöveg, képernyőkép, Betűtípus, dokumentum látható

Automatikusan generált leírás**

**57**. ábra A CheckRow metódus

A metódus először létrehozza a vizsgálni kívánt sor stringjét a GuessGrid-ből, majd átalakítja ezt úgy, hogyaz 'x' és '?' karaktereket '0'-ra cseréli, ugyanis alapvetően azt vizsgálja, hogy a kitöltött mezők kompatibilisek-e sor-segítségek számaival. Nem egyezést vizsgál, csupán azt, hogy megfelelhet-e a kívánalmaknak a GuessGridben jelenleg szereplő sor. Azért változtatja 'x' és '?' karaktereket '0'-ra, mert a sor-segítségek csak az 1-eseket tartják számon, és ezen csere mentén az eredményül kapott string csak '1'-ket és '0'-kat fog tartalmazni, és ezen stringekben kell ellenőrizni az '1'-k előfordulásait. Elsőként létrehozza a GuessGrid-nek megfelelő sorsegítségeket, majd a GuessGrid sor-segítségeit összehasonlítha az eredeti rács sor-segítségeivel. Ha a kettő List<int> megegyezik, akkor a sor teljes mértékben megfelel a sorsegítségeknek, és ezt a RowTableElements sorának háttérváltoztatásával jelzi. Azaz a sor celláinak háttérszínét világoskékre állítja, és ezzel együtt az összes cellát a játékrácsban '0'-ra, kivéve azokat, amelyek már '1'-esek. Ekkor frissíti is a cellák kattintásállapotát és ez megjeleníti 'x'-ként az ImageCellTableElements-ben (tulajdonképp kitölti a sor maradékát üres mezőkkel). Ha a sor nem helyes, akkor ellenőrzi, hogy a sor karakterlánca kompatibilis-e a sor-segítségekkel. Ha nem, akkor a RowTableElements sor celláinak háttérszínét pirosra állítja, jelezve a hibát. Ha a sor részben helyes, akkor a sor celláinak háttérszínét szürkére állítja (ez az eredeti szín és azért van erre szükség, hogy az inkompatibilis sor javítása után visszaváltson az eredeti színre, jelezve, hogy a sor még nincs teljesen kész.

Ennek a kompatibilitási feltételnek az eldöntése jelentette az egyik legnagyobb kihívást. A CheckRow metódus ezt a CheckString meghívásával és a GuessGrid sorának, illetve a feltételként szolgáló List<int> paraméterként való átadásával teszi meg.

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

58. ábra: A CheckString metódus

A CheckString metódus feladata annak ellenőrzése, hogy egy adott karakterlánc megfelel-e a sor-segítségeknek. Ehhez először átalakítja a sor-segítségeket egy string listává, ahol minden elem egy '1'-esekből álló karakterlánc. Ez a lista reprezentálja azokat a mintákat, amelyeket az input karakterláncban keresni fog. Tehát ha pl. <3,2,6> listát kapja a metódus, akkor ebből csinál egy <”111”,”11”,”111111”> listát. E lista egyes elemei reprezentálják a keresett folyamatos karakterláncokat. Majd átalakítja az input stringet, hogy az 'x' karaktereket (üres cellák) '?' karakterekre cseréli. Ez az átalakított karakterlánc lesz az, amit összehasonlít a mintákkal. Valójában csak az egyszerűség kedvéért van szükség erre az átalakításra, hogy ne kelljen or kapcsolatokkal két karaktert vizsgálni és visszaállítani. Ezután ellenőrzi, hogy az input karakterlánc tartalmaz-e elegendő '?' és '1' karaktert a mintákhoz képest. Ha nem, akkor az input nem kompatibilis a hintákkal. Ha a fenti mintánál maradunk, akkor az input stringnek minimum 11 '?' és '1' karaktert kell tartalmaznia a '0'-kon kívül. Ez egy minimum feltétele a kompatibilitásnak. Ha megfelelő számú '1'-es vagy arra felcserélhető '?' karakter van az input stringben, akkor meg kell vizsgálni, hogy elhelyezhetők-e az '1' karakterekből álló stingek az input stringben. Ezt rekurzívan ellenőrzi a CanPlacePatterns segédmetódus meghívásával, amely megpróbálja elhelyezni a mintákat az input stringben. Ha sikerül neki, akkor az input string kompatibilis a sor-segítségekkel. Ha nem sikerül neki, akkor az inkompatibilitást adja vissza.

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

59. ábra: A CanPlacePattern metódus

A CanPlacePatterns metódus feladata annak ellenőrzése, hogy a minták (az '1'-esekből álló karakterláncok) elhelyezhetők-e az input karakterláncban, illetve behelyettesíthetők-e annak valamely részébe. Az alapértelmezett visszatérési érték false, így csak akkor ad vissza true-t, ha valóban képes elhelyezni a minden mintát az input karakterlácba. Ha sikerült elhelyeznie minden '1'-esekből álló mintát, akkor a metódus ellenőrzi, hogy az input karakterláncban található '1'-esek száma megegyezik-e a minták összesített hosszával (remainingOnes). Ha igen, akkor az input karakterlánc kompatibilis a hintákkal. Erre azért van szükség, mert az elhelyezés az input string elejéről indul, és előfordulhat, hogy az összek minta elhelyezésre került az input stringben (mondjuk mert nincs sok kitöltött mező benne, vagy nem sok mintát kell elhelyezni benne), de az input string végén is vannak '1'-esek, ameddig a rekurzív elhelyezés el sem jutott. Amennyiben ez a helyzet áll elő, az input stringben található '1'-esek száma nagyobb lesz az összesen engedélyezhető '1'-esek számával, ami azt jelenti, hogy a kérdéses elhelyezés nem kompatibilis a sor-segítségekkel. A mintákat egyesével próbálja elhelyezni, és amint sikerül elhelyezni egyet, lép tovább a következőre. Az adott minta elhelyezése úgy történik, hogy a metódus megpróbálja elhelyezni az aktuális mintát az input karakterláncban minden lehetséges pozícióban (startIndex-től az input karakterlánc végéig). A CanPlacePatternAt segédmetódus ellenőrzi, hogy egy minta elhelyezhető-e az input karakterlánc adott pozícióján. Ha a minta bármelyik karaktere '0'-ra esik, akkor a minta nem helyezhető el. A PlacePattern segédmetódus elhelyezi a mintát az input karakterlánc adott pozícióján. Az input karakterlánc megfelelő pozícióit '1'-re állítja. Ha a minta elhelyezése után a következő minta nem helyezhető el, akkor a metódus visszaállítja az input karakterlánc eredeti állapotát, és folytatja a próbálkozást a következő pozícióval.

Egy példán keresztül érzékeltethető, hogyan is működik ez az ellenőrző metódus. Tegyük fel, hogy egy adott rácsban megtörténik egy kattintás, ami egy cellát telire állít (Cell clicked at Row: 1, Column: 13, ClickState: 1) a kérdéses rácsban a kérdéses sorhoz a <3,2> sor-segítség tartozik, és ezzel a kattintással a GuessGrid kérdéses sora ilyen értéket vesz fel: ?0xxxxxxxxxxx1x. Ezután meghívásra kerül a CheckRowsAndColumns, ami meghívja a CheckString-et. A debug által adott kimeneti logok világosan mutatják a CheckString metódus rekurzív működését:

CheckString called. Input: ?0xxxxxxxxxxx1x, Substrings: 3, 2

Total required 1's: 5, Available 1's (including ?): 14

Trying to place pattern '111' at position 2: ?0???????????1?

Placed pattern '111' at position 2: ?0111????????1?

Trying to place pattern '11' at position 6: ?0111????????1?

Placed pattern '11' at position 6: ?0111?11?????1?

Final check: Current 1's in string: 6, Remaining 1's: 5

Reverted to previous state after trying '11' at position 6: ?0111????????1?

Trying to place pattern '11' at position 7: ?0111????????1?

Placed pattern '11' at position 7: ?0111??11????1?

Final check: Current 1's in string: 6, Remaining 1's: 5

Reverted to previous state after trying '11' at position 7: ?0111????????1?

Trying to place pattern '11' at position 8: ?0111????????1?

Placed pattern '11' at position 8: ?0111???11???1?

Final check: Current 1's in string: 6, Remaining 1's: 5

Reverted to previous state after trying '11' at position 8: ?0111????????1?

Trying to place pattern '11' at position 9: ?0111????????1?

Placed pattern '11' at position 9: ?0111????11??1?

Final check: Current 1's in string: 6, Remaining 1's: 5

Reverted to previous state after trying '11' at position 9: ?0111????????1?

Trying to place pattern '11' at position 10: ?0111????????1?

Placed pattern '11' at position 10: ?0111?????11?1?

Final check: Current 1's in string: 6, Remaining 1's: 5

Reverted to previous state after trying '11' at position 10: ?0111????????1?

Trying to place pattern '11' at position 11: ?0111????????1?

Placed pattern '11' at position 11: ?0111??????111?

Final check: Current 1's in string: 6, Remaining 1's: 5

Reverted to previous state after trying '11' at position 11: ?0111????????1?

Trying to place pattern '11' at position 12: ?0111????????1?

Placed pattern '11' at position 12: ?0111???????11?

Final check: Current 1's in string: 5, Remaining 1's: 5

CheckString called. Input: x1xxxx1xxxxxxxx, Substrings: 9

Total required 1's: 9, Available 1's (including ?): 15

Trying to place pattern '111111111' at position 0: ?1????1????????

Placed pattern '111111111' at position 0: 111111111??????

Final check: Current 1's in string: 9, Remaining 1's: 9

Természetesen még lehetne számos eljárásról, például az itt található is szintén fontos szerepet betöltő ExecuteHelpOption metódusról, mely a rács-kitöltésben alkalmazható segítségek alkalmazásáért felel. De sajnos már így is hely és időszűkében vagyunk, ezért mindössze nagyon vázlatosan tudunk írni ennek a funkciónak a működéséről.

Maguk a segítségek a HelpTableView-ból kerülnek meghívásra, mely a főablaknak egy segédnézete. A **HelpTableView**-ból hívott segítségek (Help) közül néhány közvetlenül végrehajtható, míg mások csak a **GameView**-n keresztül hívhatók meg, mivel ezekhez szükség van paraméterekre, melyeket vagy egy cella koordinátái adnak meg, vagy pedig a sor-segítségeket tartalmazó tábla sorának ill. az oszlop-segítségeket tartalmazó tábla oszlopának indexe. Az igénybe vehető segítségek végrehajtása ettől függetlenül a GameViewModel-ben történik. A következő segítségekről beszélhetünk:

**H1:** Egy választott cella felfedése.

**H3:** Három rejtett vagy hibásan jelölt véletlen cella felfedése.

**H8:** Nyolc rejtett vagy hibásan jelölt véletlen cella felfedése.

**H13:** Tizenhárom rejtett vagy hibásan jelölt cella felfedése gyémánt alakban egy választott cella körül.

**L1:** Egy választott sor vagy oszlop teljes felfedése.

**L3:** Három egymás melletti sor vagy oszlop felfedése a középső megjelölésével.

**Check3H:** Az egész rács ellenőrzése, és maximum három hibás jelölés javítása. Ha nincs hibás jelölés, akkor nem használódik el a segítség.

**Erase:** Minden hibás jelölés törlése.

A HelpTableView-ból hívott segítségek a következők felosztása tehát aszerint is lehetséges, hogy a GameViewModel ExecuteHelpOption metódusa közvetlenül, vagy a GameView-n keresztül kerül meghívásra:

* Közvetlenül végrehajtható segítségek: H3, H8, Check3H, Erase
* GameView-n keresztül végrehajtandó segítségek: H1, H13, L1, L3

A közvetlenül végrehajtható segítségek (H3, H8, Check3H, Erase) esetén a **HelpTableViewModel** közvetlenül hívja meg az **ExecuteHelpOption** metódust a **GameViewModel**-ben. A GameView-n keresztül végrehajtandó segítségek (H1, H13, L1, L3) esetén a **HelpTableViewModel** beállítja a **SelectedHelpOption** tulajdonságot a **GameView**-ban, és várja, hogy a felhasználó kiválassza a cellát. A **GameView** osztályban a **SelectedHelpOption** tulajdonságot használjuk a segítségek végrehajtásához. Amikor a felhasználó kiválaszt egy cellát, a **Cell\_MouseLeftButtonDown** eseménykezelő megnézi, hogy be van-e állítva valamilyen SelectedHelpOption, és ha igen, akkor meghívja **ExecuteHelpOption** metódust a **GameViewModel**-ben. Az ExecuteHelpOption metódus

paraméterei a következők: row (sor), column (oszlop) és typeOfHelp (segítség típusa). Azt a problémát, hogy a közvetlenül hívható segítségeknek nincs szükségük sor és oszlop paraméterre úgy hidalja át a HelpTableViewModel, hogy beállít egy 0 default értéket mindkettőre. Az algoritmus pedig így halad előre:

1. Paraméterek ellenőrzése: Ellenőrzi, hogy a paraméterek érvényesek-e.
2. Segítség típusának azonosítása: A metódus switch-case szerkezetet használ a segítség típusának azonosítására és a megfelelő logika végrehajtására.
3. Segítség végrehajtása: A segítség típusától függően különböző logikát hajt végre, például cellák felfedése, hibák javítása stb.
4. Sorok és oszlopok ellenőrzése: A segítség végrehajtása után ellenőrzi a sorok és oszlopok helyességét.
5. Játék megnyerésének ellenőrzése: Ellenőrzi, hogy a játékos megnyerte-e a játékot a segítség végrehajtása után.

## **14. Tesztelési dokumentáció**

A játék fejlesztése során a tesztelés kiemelt szerepet kapott. Ez magába foglalta a pályák és a menü különböző elemeinek tesztelését, valamint a bejelentkezési folyamat alapos ellenőrzését annak érdekében, hogy a rendszer megfelelően kezelje a megadott adatokat. A manuális tesztek részletes leírását az „Alkalmazott fejlesztői eszközök” szakaszban található. Az alábbiakban bemutatjuk, hány és milyen típusú egységteszt készült a projekthez, különös tekintettel a DbManagerTests.cs fájlra.

**Constructor\_ShouldInitializeUserName**:

* **Mit tesztel?** Ellenőrzi, hogy a MainViewModel konstruktora helyesen inicializálja-e a UserName tulajdonságot.
* **Teszt részletei**:
  + A konstruktor meghívása után ellenőrzi, hogy a UserName értéke megegyezik-e a testUser értékkel.

**LoadUserData\_ShouldSetUserProperty**:

* **Mit tesztel?** Ellenőrzi, hogy a LoadUserData metódus helyesen tölti-e be az adatokat, és beállítja-e a User tulajdonságot.
* **Teszt részletei**:
  + A LoadUserData metódus meghívása után ellenőrzi, hogy a User tulajdonság nem null és a UserName megegyezik a testUser értékkel.

A képen szöveg, elektronika, képernyőkép, képernyő látható

Automatikusan generált leírás

60. ábra: A DbManager.cs osztály tesztelése

## **15. Továbbfejlesztési lehetőségek**

A Nonogram játék továbbfejlesztésére számos innovatív lehetőség kínálkozik, amelyek fokozhatják a játékélményt és új kihívásokat nyújthatnak a játékosoknak. Ezek az újítások változatosabbá, izgalmasabbá és tartalmasabbá tehetik a játékot, mind vizuális, mind stratégiai szempontból.

Az egyik legérdekesebb fejlesztési lehetőség az új játékmódok bevezetése. Például az időkorlátos mód, ahol a játékosoknak adott időn belül kell teljesíteniük a rejtvényt. Ez növeli a játék feszültségét és izgalmát, mivel a játékosoknak gyorsan kell gondolkodniuk és dönteniük. A versenymód is izgalmas lehetőség, amely lehetővé teszi, hogy több játékos versenyezzen egymással, hogy ki tudja gyorsabban megoldani a rejtvényt. A kihívás mód szintén új dimenziót adhat a játéknak, ahol nehezített pályák speciális feltételekkel vagy extra nehézségi szintekkel jelentkeznek. Ez a mód különösen azoknak a játékosoknak lehet érdekes, akik már jártasak a játékban és újabb kihívásokat keresnek.

A színes kitöltés bevezetése is nagy előrelépés lehet a játék számára. Jelenleg a játék csak egyszínű kitöltést használ, de a színes kitöltés lehetősége még látványosabbá és élvezetesebbé teheti a játékot. A játékosok különböző színeket használhatnának a rejtvény megoldása során, ami új stratégiákat és taktikai megközelítéseket igényelne, és vizuálisan is gazdagabb élményt nyújtana.

A pályakészlet bővítése szintén fontos fejlesztési lehetőség. Különböző tematikus pályacsomagok bevezetésével, például évszakok, ünnepek vagy híres helyszínek témájában, a játék még változatosabbá válhat. Emellett a felhasználók által készített pályák lehetősége is nagyban növelheti a játék közösségi aspektusát. A játékosok saját pályákat készíthetnek és megoszthatják azokat másokkal, ami egy folyamatosan bővülő pályakészletet eredményezhet, és hosszú távon is friss élményt nyújthat a játékosok számára.

A jutalmazási rendszer bevezetése szintén hozzájárulhat a játék élvezetéhez és motivációjához. Naponta változó kihívások és jutalmak kínálása a játékosok számára pontokat vagy virtuális tárgyakat nyerhetnek, ami arra ösztönzi őket, hogy rendszeresen visszatérjenek a játékhoz. A szintezés és rangsor rendszerével a játékosok szinteket léphetnek és versenyezhetnek egymással, ami hosszabb távon is fenntartja a motivációt.

A grafikai és hanghatások fejlesztése szintén fontos szerepet játszhat a játék élvezeti értékének növelésében. Látványos animációk bevezetése a rejtvények megoldásakor vizuális élményt nyújtanak, míg a különböző hangok a játékos műveleteihez, például a helyes és helytelen lépésekhez, fokozzák az immersziót és a játékosok élményét.

Az oktatási és tanulási módok bevezetése szintén nagy segítség lehet a kezdő játékosok számára. Útmutatók és tippek beépítése a játékba segíthet megtanulni a játék szabályait és stratégiáit, míg a gyakorló pályák lehetőséget adnak a játékosoknak, hogy fejlődjenek és gyakoroljanak. Ez különösen fontos lehet azok számára, akik még újonnan ismerkednek a játékkal és szeretnék jobban megérteni annak működését.

A társadalmi és közösségi funkciók bevezetése is jelentősen növelheti a játék élményét. Lehetőség barátok hozzáadására és egymás eredményeinek követésére közösségi élményt nyújt, míg a közös játékok lehetősége, ahol barátokkal vagy csapatban lehet megoldani a rejtvényeket, fokozza a közösségi interakciókat és együttműködést.

Összességében ezek a fejlesztési irányok nemcsak gazdagabbá és szórakoztatóbbá tehetik a Nonogram játékot, hanem új kihívásokat is nyújtanak, miközben közösségi élményt is biztosítanak a játékosok számára. Az új játékmódok, a színes kitöltés, a bővített pályakészlet, a jutalmazási rendszer, a grafikai és hanghatások fejlesztése, az oktatási módok és a közösségi funkciók mind hozzájárulhatnak ahhoz, hogy a Nonogram játék még vonzóbb és élvezetesebb legyen minden játékos számára.

# **IV. Összegzés**

A projektmenedzser jelentős fejlődést mutatott a csapattagok koordinálásában és a feladatok hatékony szétosztásában, valamint azok betartatásában. Rendszeresen szervezett és vezetett megbeszéléseket a csapattagokkal mind az iskolában, mind online platformokon, mint például a Discord és az e-mail. A csapat tagjai sokat fejlődtek ezekből a készségekből, mivel az előadások során mindenkinek prezentálnia és beszélnie kellett a feladatairól.

A projekt számára rendelkezésre álló idő meglehetősen korlátozott volt, különösen azért, mert a tanmenet szerint később kezdtük el az asztali alkalmazás GUI felületének fejlesztését, és szinte nulla ismerettel vágtunk neki. Ennek ellenére sikerült összehoznunk a projektet, rengeteg utánajárással és kitartó munkával.

## **Hivatkozások:**

Morales, J. (2023): Kanban Methodology Review, Principles, Uses, & How to Create It, In: <https://www.mindonmap.com/blog/what-is-kanban/> (Hozzáférés: 2025.01.02)

Juhász T. (2005): Adatbázis-kezelés. In Devecz Ferenc, Jónás Katalin, Juhász Tibor, Kévés Rita, Reményi Zoltán, Seigler Gábor, Takács Barnabás (2005): *Irány az ECDL! - Irány a középszintű érettségi!* Budapest: Nemzeti Tankönyvkiadó, 145-198.

Kardos Z. (2008): *Adatbázis kezelési alapismeretek.* Budapest: Nemzeti Szakképzési és Felnőttképzési Intézet. In: <https://www.nive.hu/Downloads/Szakkepzesi_dokumentumok/Bemeneti_kompetenciak_meresi_ertekelesi_eszkozrendszerenek_kialakitasa/17_0061_011_101030.pdf> (Hozzáférés: 2025.01.05)

Blum, R (2020): *PHP, MySQL & JavaScript 7 könyv 1-ben.* Budapest: Taramix Kiadó

1. A GDPR rendelet az Európai Parlament által létrehozott jogszabály, az elnevezés rövidítése az angol General Data Protection Regulation kifejezés szava, ami magyarul annyit jelent, hogy: Általános Adatvédelmi Rendelet. [↑](#footnote-ref-1)