

3. Beadandó feladat dokumentáció

Készítette:

Hallgató Harald

E-mail: haha@inf.elte.hu

Feladat:

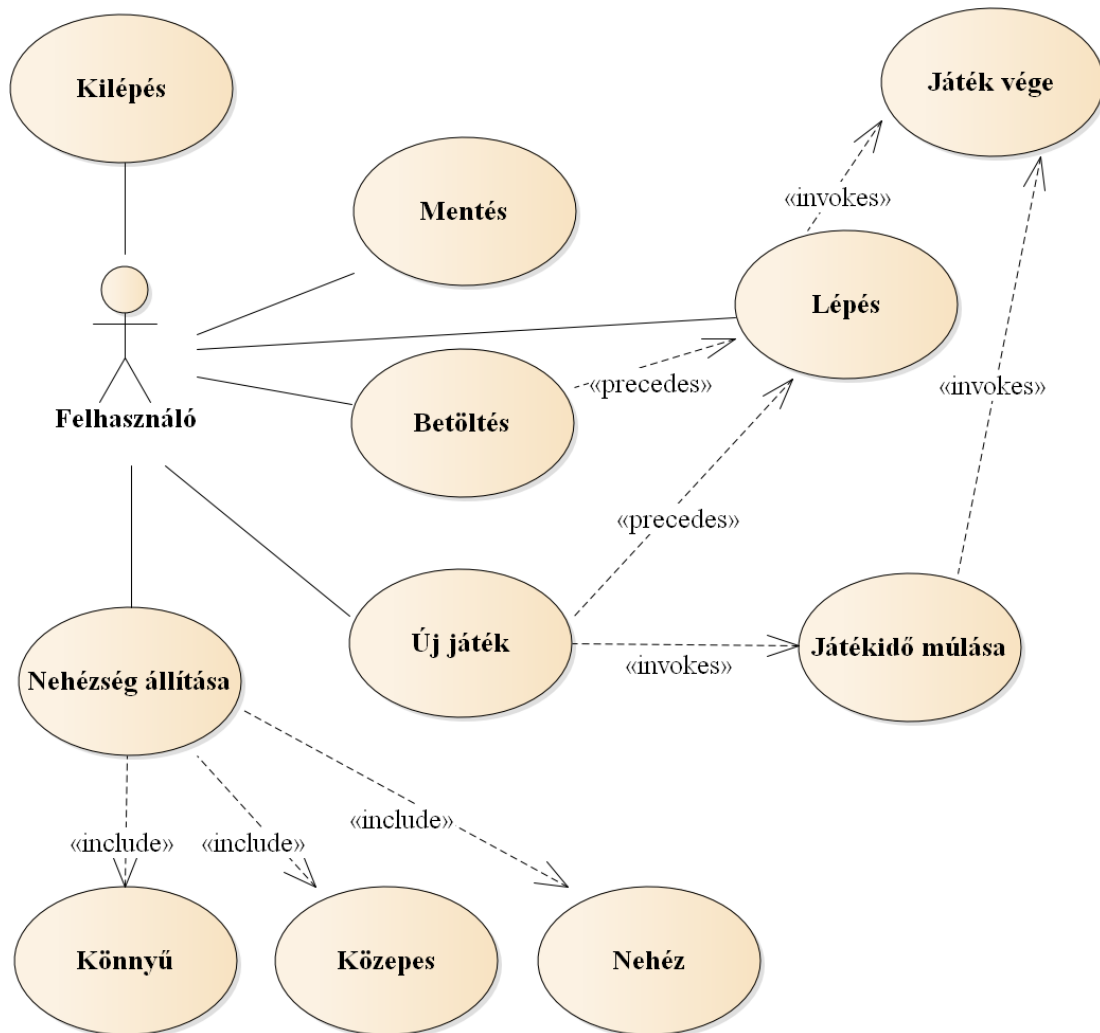
Készítsünk egy Sudoku játékprogramot. A Sudoku egy olyan 9×9 -es táblázat, amelyet úgy kell a 0-9 számjegyekkel kitölteni, hogy minden sorában, minden oszlopában és minden házában egy számjegy pontosan egyszer szerepeljen. (Házaknak a 9×9 -es táblázatot lefedő, de egymásba át nem érő kilenc darab 3×3 résztáblázatot nevezünk.) A 81 darab kis négyzet bármelyikére kattintva a négyzet felirata változzon meg: az üres felirat helyett 1-esre, az 1-es helyett 2-esre, és így tovább, végül a 9-es helyett üresre. Ennek megfelelően bármelyik négyzeten néhány (legfeljebb kilenc) kattintással egy tetszőleges érték állítható be. Egy adott négyzeten történő kattintgatás esetén soha ne jelenjék meg olyan szám, amely az adott négyzettel egy sorban, egy oszlopban vagy egy házban már szerepel.

A program számolja a játékos lépéseit, valamint az eltelt időt. A programnak támogatnia kell új játék kezdését, játék betöltését, valamint mentését. Betöltéskor a már kitöltött mezőket ne lehessen állítani. Hasonlóan, új játék kezdésekor legyen véletlenszerűen kitöltve pár mező, amelyeket utólag nem lehet módosítani. Ezen felül a program nehézségi szinteket is kezeljen, amely meghatározza a játék maximális idejét (ezért a hátralévő időt jelenítsük meg), valamint új játék esetén az előre beállított mezők számát.

Elemzés:

- A feladatot .NET Avalonia alkalmazásként, elsődlegesen asztali alkalmazásként és Android platformon valósítjuk meg, amely egy fő nézetből fog állni. Az alkalmazás portré tájolást támogat.
- A játék nézete tartalmazza a játéktáblát, a játék állását (lépések száma, fennmaradó idő) a lap alján, az új játék, valamint a beállítások gombjait a nézet tetején. Játék betöltésére és mentésére az adott operációs rendszer dialógusablakain keresztül van lehetőség.
- A játékot három nehézségi szinttel játszhatjuk: könnyű (60 perc, 6 generált mező), közepes (20 perc, 12 előre generált mező), nehéz (10 perc, 18 előre generált mező). A program indításkor közepes nehézséget állít be, és automatikusan új játékot indít.
- A játéktábla 9×9 mezőből áll, amelyeket érintéssel kezelhetünk. Az érintés hatására a játék lépteti az adott mezőn megjelenő számot. A számot egyesével növeljük, és csak azokat az értékeket engedjük, amelyek még nem szerepelnek az adott sorban, oszlopban és házban. A táblán az előre betöltött, illetve generált mezőket nem engedjük megváltoztatni, ezeket sárga színnel jelöljük.

- A játék automatikusan jelzi előugró üzenettel, ha vége a játéknak (kiraktuk a táblát, vagy letelt az idő).
- A használati esetek az 1. ábrán láthatóak.

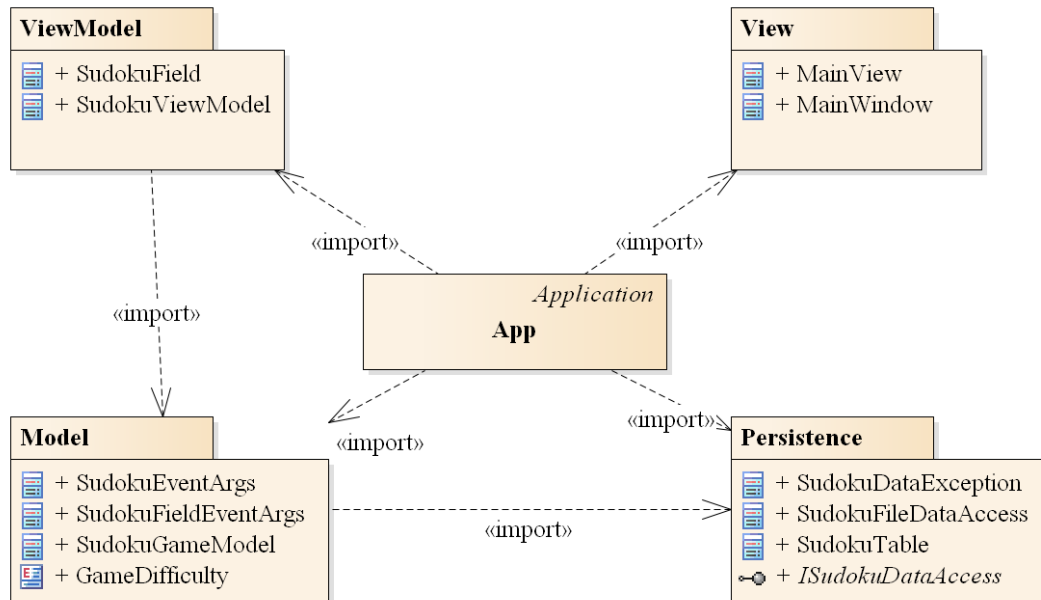


1. ábra: Használati esetek diagramja

Tervezés:

- Programszerkezet:
 - A szoftvert négy projektből építjük fel: a modellt és a perzisztenciát tartalmazó osztálykönyvtárból (.NET Standard Class Library), valamint a .NET Avalonia projektjeiből (platformfüggetlen osztálykönyvtár és platformfüggő végrehajtható projektek), amelyet így Windows és Android operációs rendszerekre is le tudunk fordítani.
 - A programot MVVM architektúrában valósítjuk meg, ennek megfelelően **View**, **Model**, **ViewModel** és **Persistence** névttereket valósítunk meg az alkalmazáson belül.

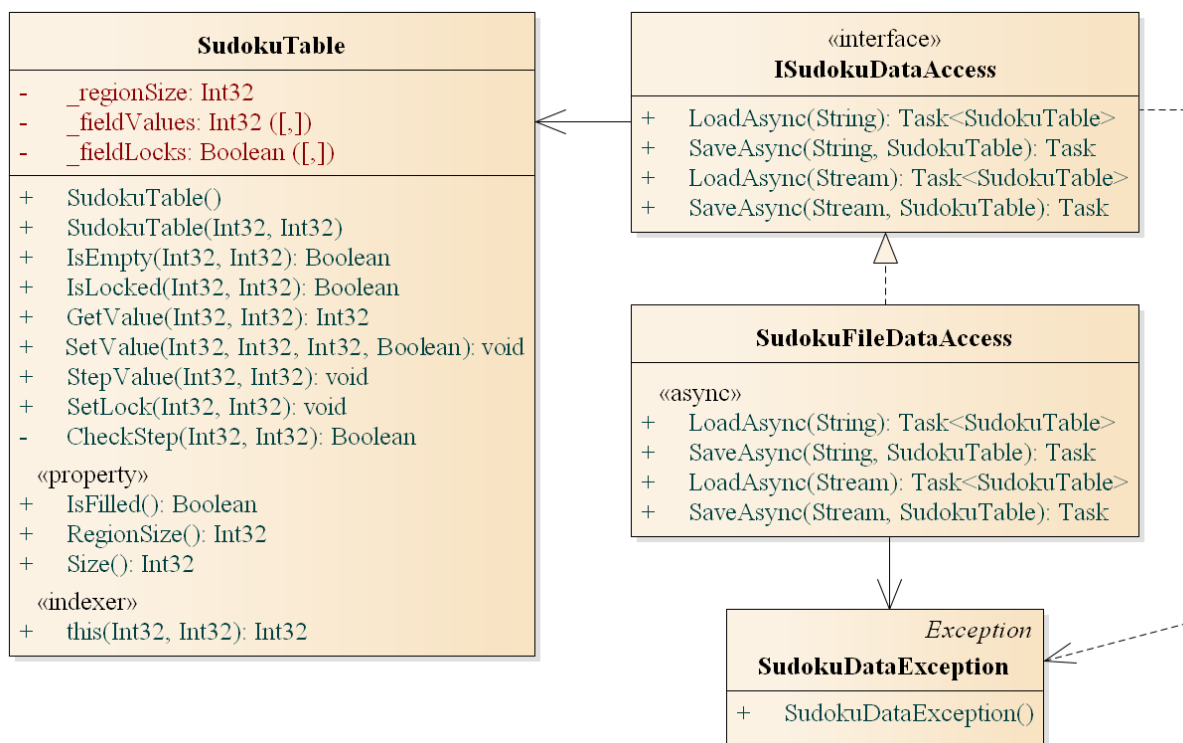
- A program vezérlését az alkalmazás osztály (**App**) végzi, amely példányosítja a modellt, a nézetmodell és a nézetet, biztosítja a kommunikációt, valamint felügyeli az adatkezelést. Az alkalmazás környezeti réteg feladata a platformfüggetlen alkalmazás életciklus helyénvaló kezelése is.
- A program csomagdiagramja a 2. ábrán látható.



2. ábra: Az alkalmazás csomagdiagramja

- Perzisztencia (3. ábra):
 - Az adatkezelés feladata a Sudoku táblával kapcsolatos információk tárolása, valamint a betöltés/mentés biztosítása.
 - A **SudokuTable** osztály egy érvényes Sudoku táblát biztosít (azaz mindig ellenőrzi a beállított értékek), ahol minden mezőre ismert az értéke (**_fieldValues**), illetve a zároltsága (**_fieldLocks**). Utóbbit a játék kezdetekor generált, illetve értékekre alkalmazzuk. A tábla alapértelmezés szerint 9×9 -es 3×3 -as házakkal, de ez a konstruktorban paraméterezhető. A tábla lehetőséget az állapotok lekérdezésére (**IsFilled**, **IsLocked**, **IsEmpty**, **GetValue**), valamint szabályos léptetésre (**StepValue**), illetve direkt beállítás (**SetValue**, **SetLock**) elvégzésére.
 - A hosszú távú adattárolás lehetőségeit az **ISudokuDataAccess** interfész adja meg, amely lehetőséget ad a tábla betöltésére (**LoadAsync**), valamint mentésére (**SaveAsync**). A műveleteket hatékonysági okokból aszinkron módon valósítjuk meg. A mobilos platformok támogatása érdekében a műveleteket adatfolyamok (**Stream**) támogatásával is megvalósítjuk.

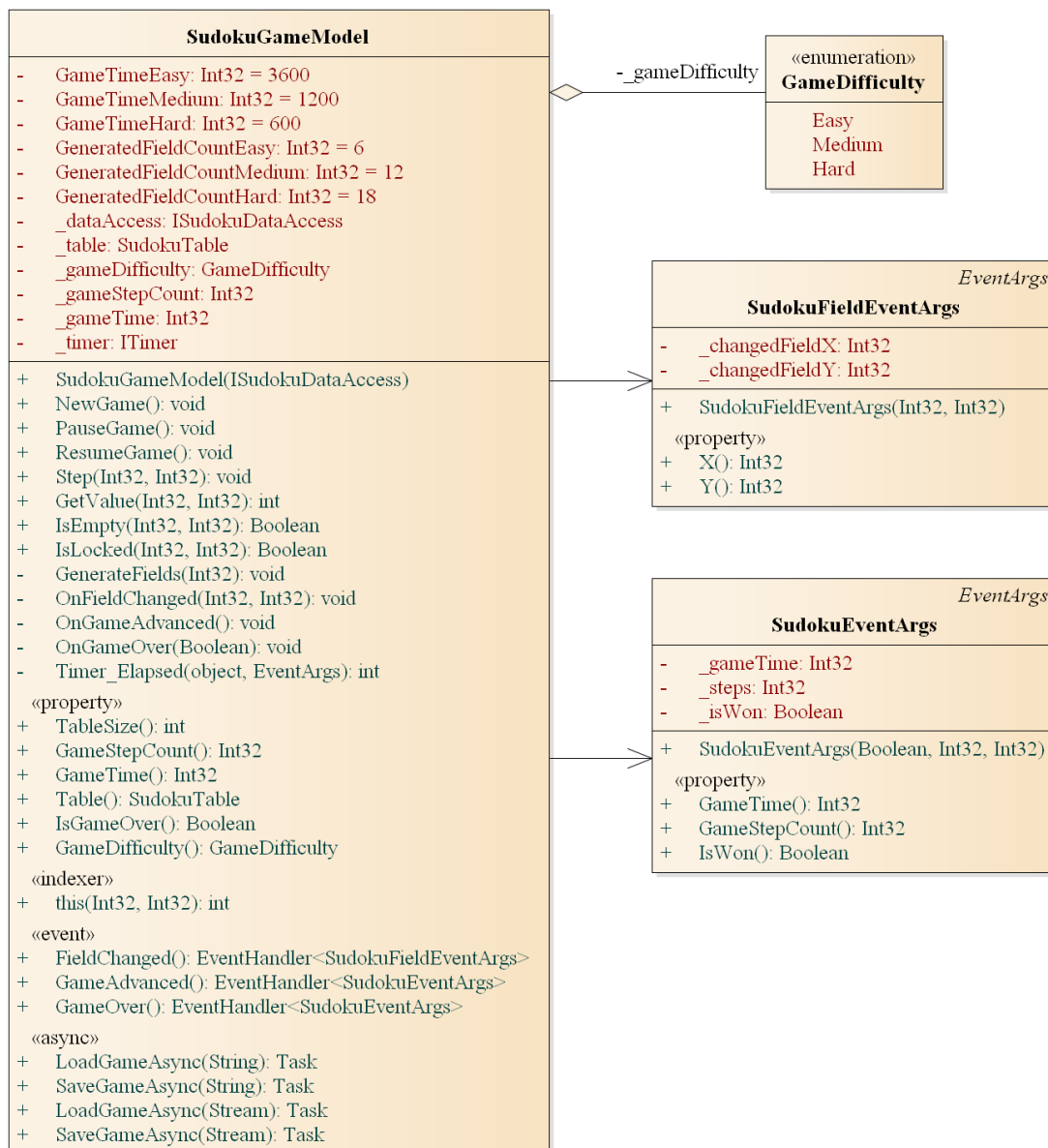
- Az interfészt szöveges fájl alapú adatkezelésre a **SudokuFileDataAccess** osztály valósítja meg. A fájlkezelés során fellépő hibákat a **SudokuDataException** kivétel jelzi.
- A program az adatokat szöveges fájlként tudja eltárolni, melyek az **stl** kiterjesztést kapják. Ezeket az adatokat a programban bármikor be lehet tölteni, illetve ki lehet menteni az aktuális állást.
- A fájl első sora megadja a tábla méretét, valamin a házak méretét (ami alapértelmezés szerint 9 és 3). A fájl többi része izomorf leképezése a játéktáblának, azaz összesen 9 sor következik, és minden sor 9 számot tartalmaz szóközzel választva. A számok 0-9 közöttiek lehetnek, ahol 0 reprezentálja a még üres mezőt.



4. ábra: A Persistence csomag osztálydiagramja

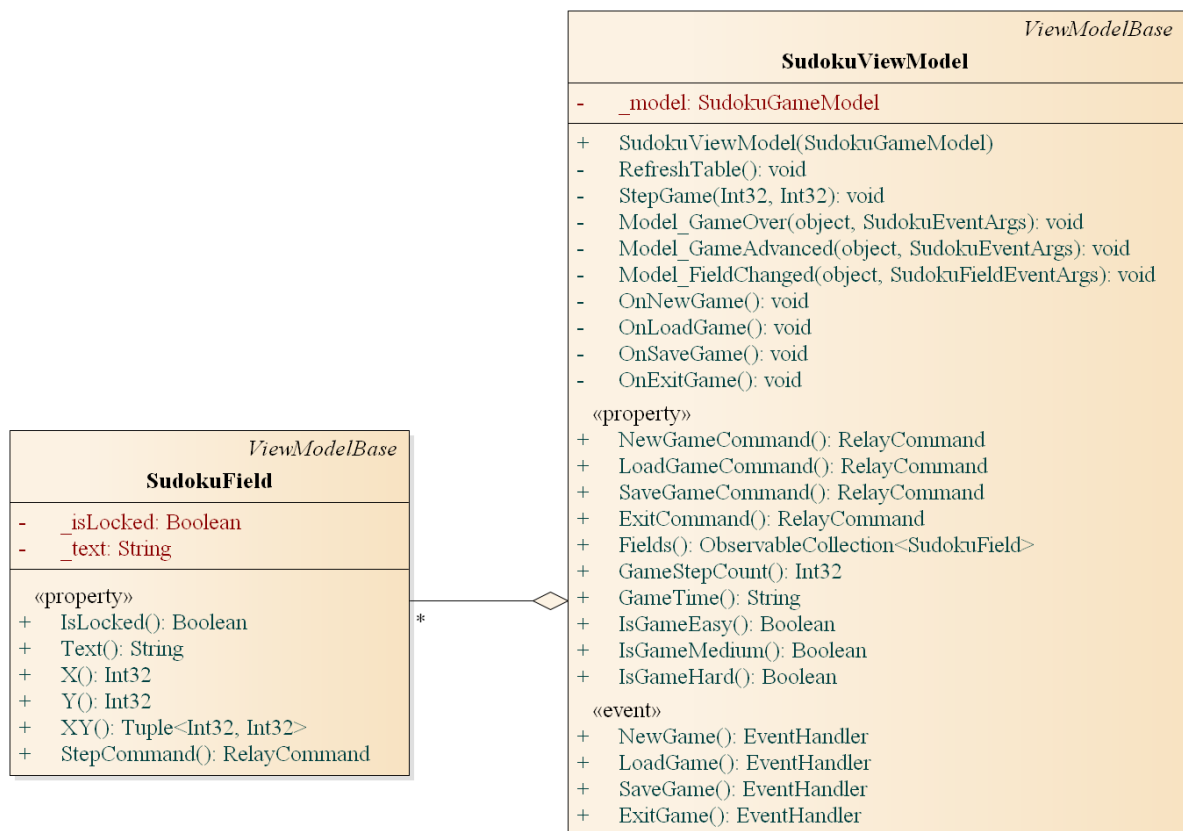
- Modell (4. ábra):
 - A modell lényegi részét a **SudokuGameModel** osztály valósítja meg, amely szabályozza a tábla tevékenységeit, valamint a játék egyéb paramétereit, úgymint az idő (`_gameTime`) és a lépések (`_gameStepCount`). A típus lehetőséget ad új játék kezdésére (`NewGame`), valamint lépésre (`StepGame`). Új játéknál megadható a kiinduló játéktábla is, különben automatikusan generálódnak kezdő mezők.
 - A játék időbeli kezelését egy időzítő végzi (`_timer`), amelyet inaktíválunk majd (`PauseGame`), amennyiben bizonyos menüfunkciók futnak, majd újraindítjuk (`ResumeGame`).

- A mezők állapotváltozásáról a **FieldChanged** esemény tájékoztat. Az esemény argumentuma (**SudokuFieldEventArgs**) tárolja a megváltozott mező pozícióját.
- A játékállapot változásáról a **GameAdvanced** esemény, míg a játék végéről a **GameOver** esemény tájékoztat. Az események argumentuma (**SudokuEventArgs**) tárolja a győzelem állapotát, a lépések számát, valamint a játékidőt.
- A modell példányosításkor megkapja az adatkezelés felületét, amelynek segítségével lehetőséget ad betöltésre (**LoadGame**) és mentésre (**SaveGame**)
- A játék nehézségét a **GameDifficulty** felsorolási típuson át kezeljük, és a **SudokuGame** osztályban konstansok segítségével tároljuk az egyes nehézségek paramétereit.



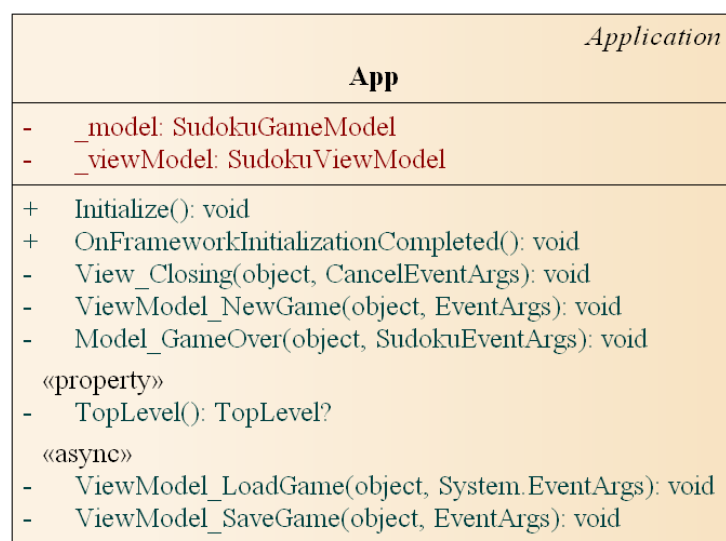
4. ábra: A Model csomag osztálydiagramja

- Nézetmodell (5. ábra):
 - A nézetmodell megvalósításához felhasználjuk az *MVVM Toolkit* csomagból elérhető általános utasítás (**RelayCommand**), valamint egy ős változásjelző (**ObservableObject**) osztályt.
 - A nézetmodell feladatait a **SudokuViewModel** osztály látja el, amely parancsokat biztosít az új játék kezdéséhez, játék betöltéséhez, mentéséhez, valamint a kilépéshez. A parancsokhoz eseményeket kötünk, amelyek a parancs lefutását jelzik a vezérlőnek. A nézetmodell tárolja a modell egy hivatkozását (**_model**), de csupán információkat kér le tőle, illetve a játéknehézséget szabályozza. Direkt nem avatkozik a játék futtatásába.
 - A játékmező számára egy külön mezőt biztosítunk (**SudokuField**), amely eltárolja a pozíciót, szöveget, engedélyezettséget, valamint a lépés parancsát (**StepCommand**). A mezőket egy felügyelt gyűjteménybe helyezzük a nézetmodellbe (**Fields**).



5. ábra: A nézetmodell osztálydiagramja

- **Nézet:**
 - A nézetet ablakok, egyedi vezérlők és dialógusablakok használatával valósítjuk meg.
 - A **MainView** osztály, mint **UserControl** leszármazott tartalmazza a játéktáblát, amelyet egy **UniformGrid** segítségével valósítunk meg, amelyben **Button** elemeket helyezünk el.
 - A **MainWindow** ablakba egyszerűen a **MainView** vezérlőt ágyazzuk be. Ilyen módon a felület asztali alkalmazásokban ablakos alkalmazásként, mobil platformon pedig lapként is megjeleníthető.
 - Betöltéshez és mentéshez a **StorageProvider** osztály által az **OpenFilePickerAsync** és **SaveFilePickerAsync** metódusokon keresztül biztosított, operációs rendszer specifikus dialógus ablakokat (lapokat) használjuk.
 - Felugró üzenetek megjelenítéséhez a **MessageBox.Avalonia** NuGet csomagot használjuk.
- **Vezérlés (6. ábra):**
 - Az **App** osztály feladata az alkalmazás vezérlése, a rétegek példányosítása és az események feldolgozása.
 - A **OnFrameworkInitializationCompleted** metódus felüldefiniálásával kezeljük a nézet platform specifikus megjelenítését, továbbá az alkalmazás életciklusát a megfelelő eseményekre történő feliratkozással. Így különösen mobil célplatform esetén az alkalmazás felfüggesztéskor (**Deactivated**) elmentjük az aktuális játékállást (**SuspendedGame**), míg folytatáskor vagy újraindításkor (**Activated**) pedig folytatjuk, amennyiben történt mentés.



6. ábra: A vezérlés osztálydiagramja

Tesztelés:

- A modell funkcionalitása egységtesztek segítségével lett ellenőrizve a **SudokuGameModelTest** osztályban.
- A modell időzítőjét egy **ITimer** interfészből származtattuk le, így azt a teszt projektben egy **MockTimer** megvalósítással *mockolhatjuk*, és az időzítőt explicit is triggerelhetjük (**RaiseElapsed**).
- Az alábbi tesztesetek kerültek megvalósításra:
 - **SudokuGameModelNewGameEasyTest**,
SudokuGameModelNewGameMediumTest,
SudokuGameModelNewGameHardTest: Új játék indítása, a mezők kitöltése, valamint a lépésszám és nehézség ellenőrzése a nehézségi fokozat függvényében.
 - **SudokuGameModelStepTest**: Játékbeli lépés hatásainak ellenőrzése, játék megkezdése előtt, valamint után. Több lépés végrehajtása azonos játékmezőn, esemény kiváltásának ellenőrzése.
 - **SudokuGameModelAdvanceTimeTest**: A játékbeli idő kezelésének ellenőrzése, beleértve a játék végét az idő lejártával.