2. Beadandó feladat dokumentáció

Készítette:
Hallgató Harald
E-mail: haha@inf.elte.hu

Feladat:

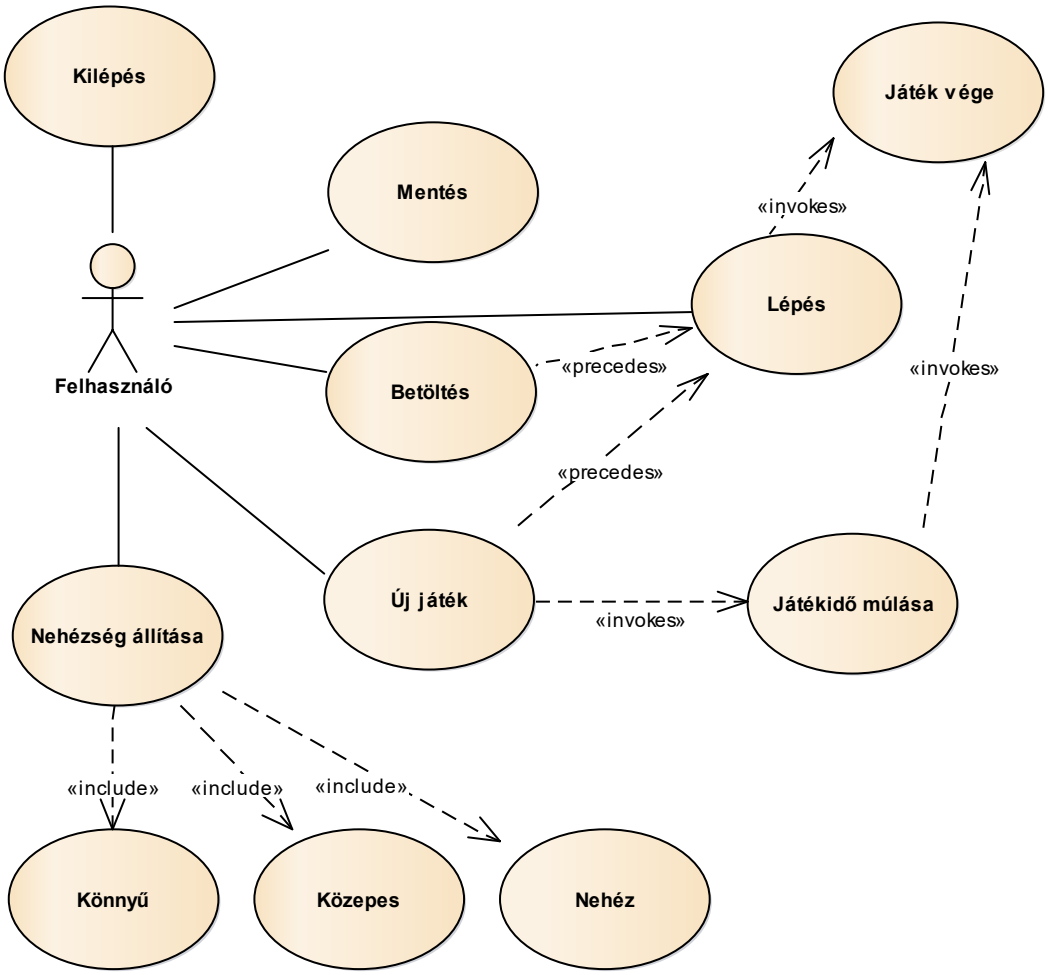
Készítsünk egy Sudoku játékprogramot. A Sudoku egy olyan 9×9-es táblázat, amelyet úgy kell a 0-9 számjegyekkel kitölteni, hogy minden sorában, minden oszlopában és minden házában egy számjegy pontosan egyszer szerepeljen. (Házaknak a 9×9-es táblázatot lefedő, de egymásba át nem érő kilenc darab 3×3 résztáblázatot nevezzük.) 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. Új játék kezdésekor legyen véletlenszerűen kitöltve pár mező, amelyeket utólag nem lehet módosítani. Játék mentésekor jegyezzük meg az így zárolt mezőket és betöltéskor ennek megfelelően állítsuk vissza a játéktáblát. 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 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 feladatot egyablakos asztali alkalmazásként Windows Presentation Foundation grafikus felülettel valósítjuk meg.
- Az ablakban elhelyezünk egy menüt a következő menüpontokkal: File (Új játék, Játék betöltése, Játék mentése, Kilépés), Beállítások (Könnyű játék, Közepes játék, Nehéz játék). Az ablak alján megjelenítünk egy státuszsort, amely a lépések számát, illetve a hátralévő időt jelzi.
- A játéktáblát egy 9×9 nyomógombokból álló rács reprezentálja. A nyomógomb egérkattintás hatására megváltoztatja a megjelenített számot. A számot változtatáskor egyesével növeljük, és csak azokat az értékeket engedjük, amelyek még nem szerepelnek az adott sorban, oszlopban és házban. A táblán az előre generált mezőket nem engedjük megváltoztatni, ezeket sárgával jelöljük.

- A játék automatikusan feldob egy dialógusablakot, amikor vége a játéknak (kiraktuk a táblát, vagy letelt az idő). Szintén dialógusablakokkal végezzük el a mentést, illetve betöltést, a fájlneveket a felhasználó adja meg.
- A felhasználói esetek az 1. ábrán láthatóak.

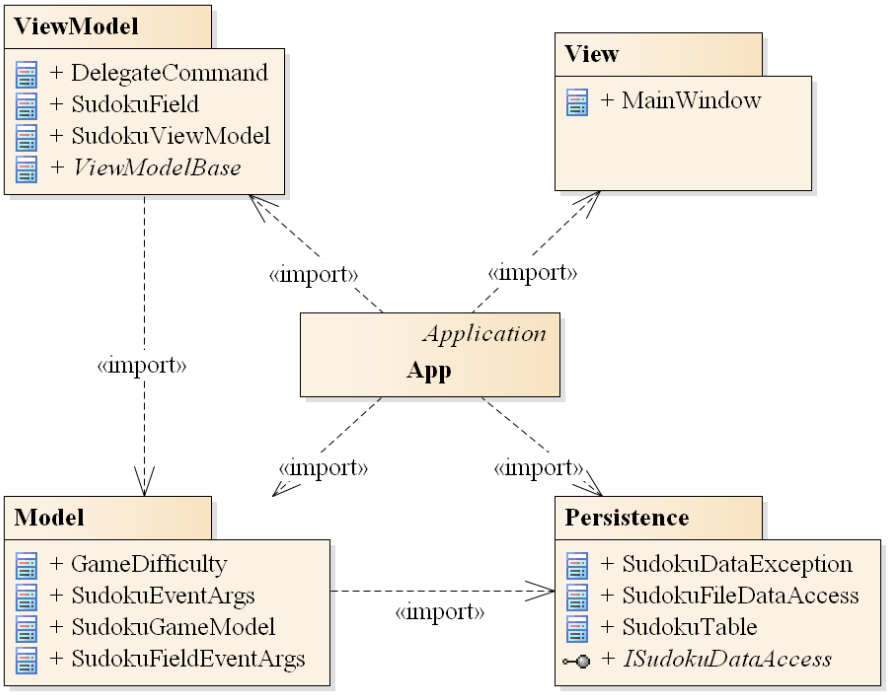


1. ábra: Felhasználói esetek diagramja

Tervezés:

- Programszerkezet:
  - A programot MVVM architektúrában valósítjuk meg, ennek megfelelően **`View`**, **`Model`**, **`ViewModel`** és **`Persistence`** névtereket valósítunk meg az alkalmazáson belül. A program környezetét az alkalmazás osztály (**`App`**) végzi, amely példányosítja a modellt, a nézetmodell és a nézetet, biztosítja a kommunikációt, valamint felügyeli az adatkezelést. A program csomagszerkezete a 2. ábrán látható.
  - A program szerkezetét két projektre osztjuk implementációs megfontolásból: a **`Persistence`** és **`Model`** csomagok a program

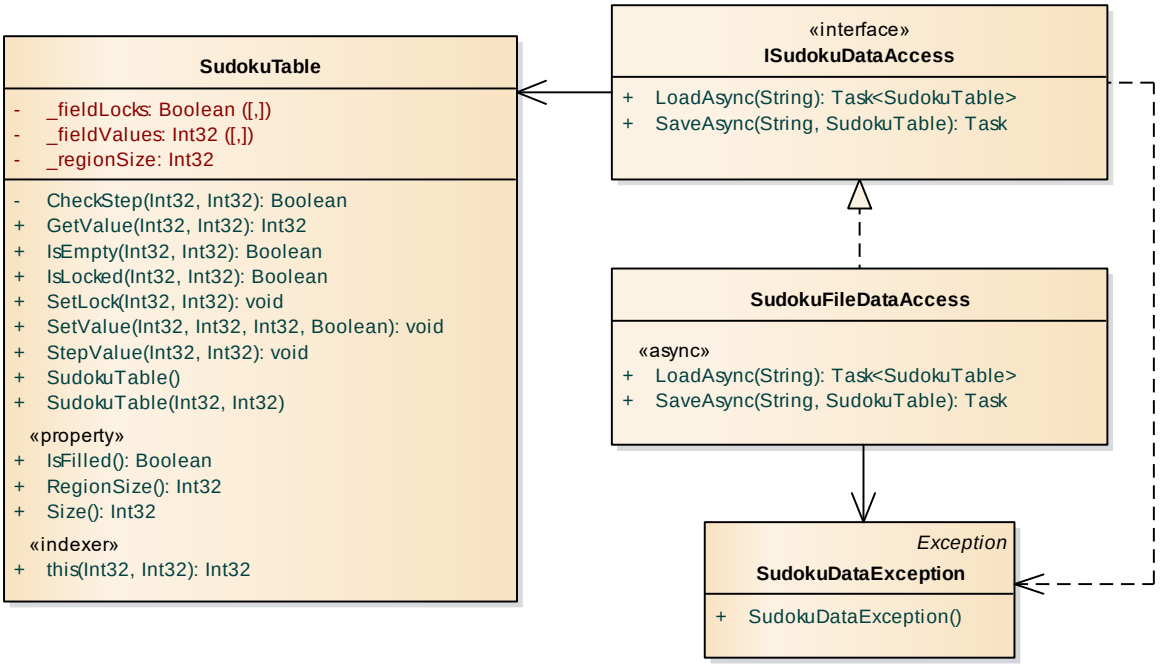
felületfüggetlen projektjében, míg a **`ViewModel`** és **`View`** csomagok a WPF függő projektjében kapnak helyet.



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.
  - Az interfészt szöveges fájl alapú adatkezelésre a **`SudokuFileDataAccess`** osztály valósítja meg. A fájlkezelés során fellépő hibákat a **`SudokuDataException`** kivétel jelzi.

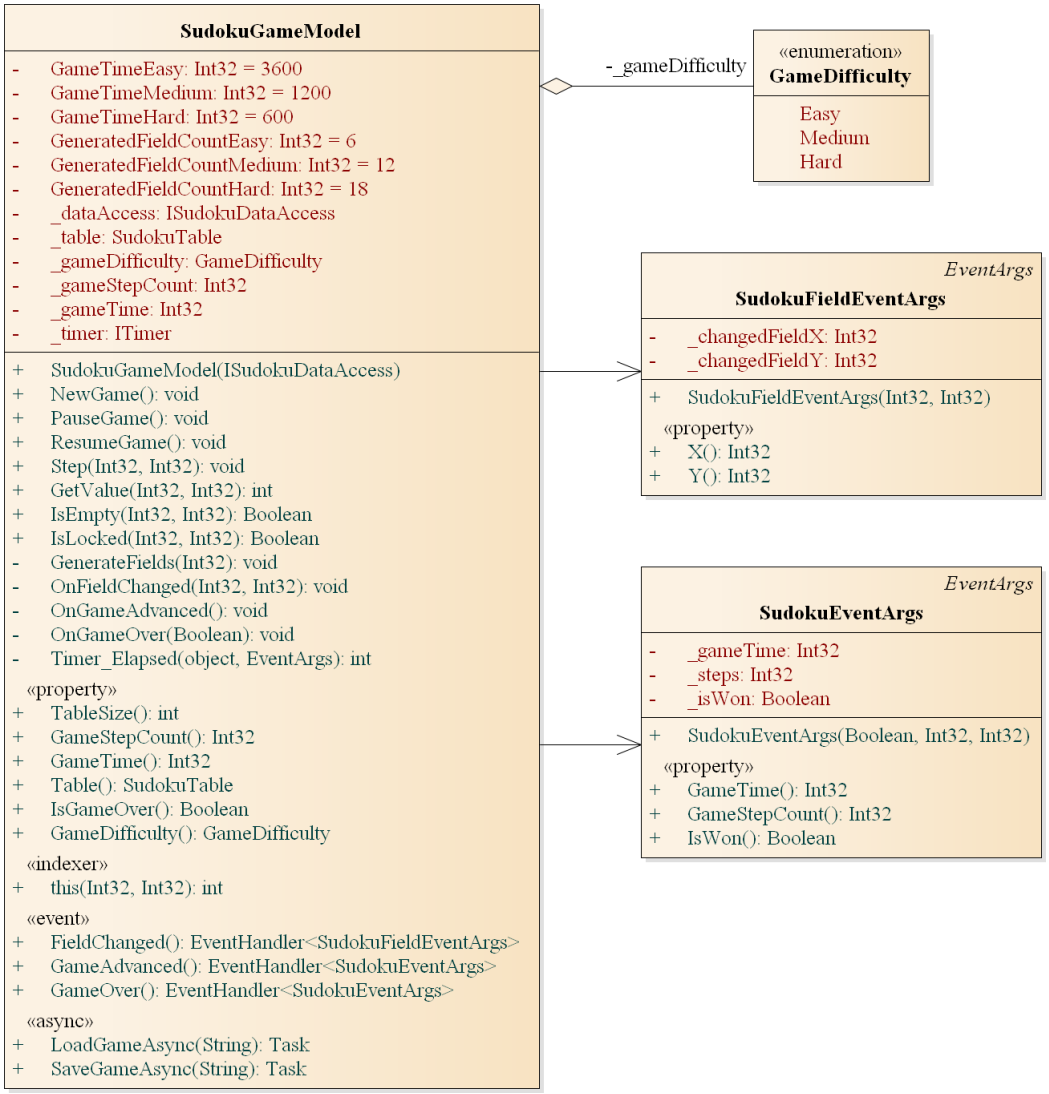
- A program az adatokat szöveges fájlként tudja eltárolni, melyek az **`stl`** kiterjesztést kapják. Ezeket az adatokat a programban bármikor be lehet tölteni, illetve ki lehet menteni az aktuális állást.
- A fájl első sora megadja a tábla méretét, valamin a házak méretét (ami alapértelmezés szerint 9 és 3). A fájl többi része izomorf leképezése a játéktáblának, azaz összesen 9 sor következik, és minden sor 9 számot tartalmaz szóközökkel 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 inaktiválunk 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.

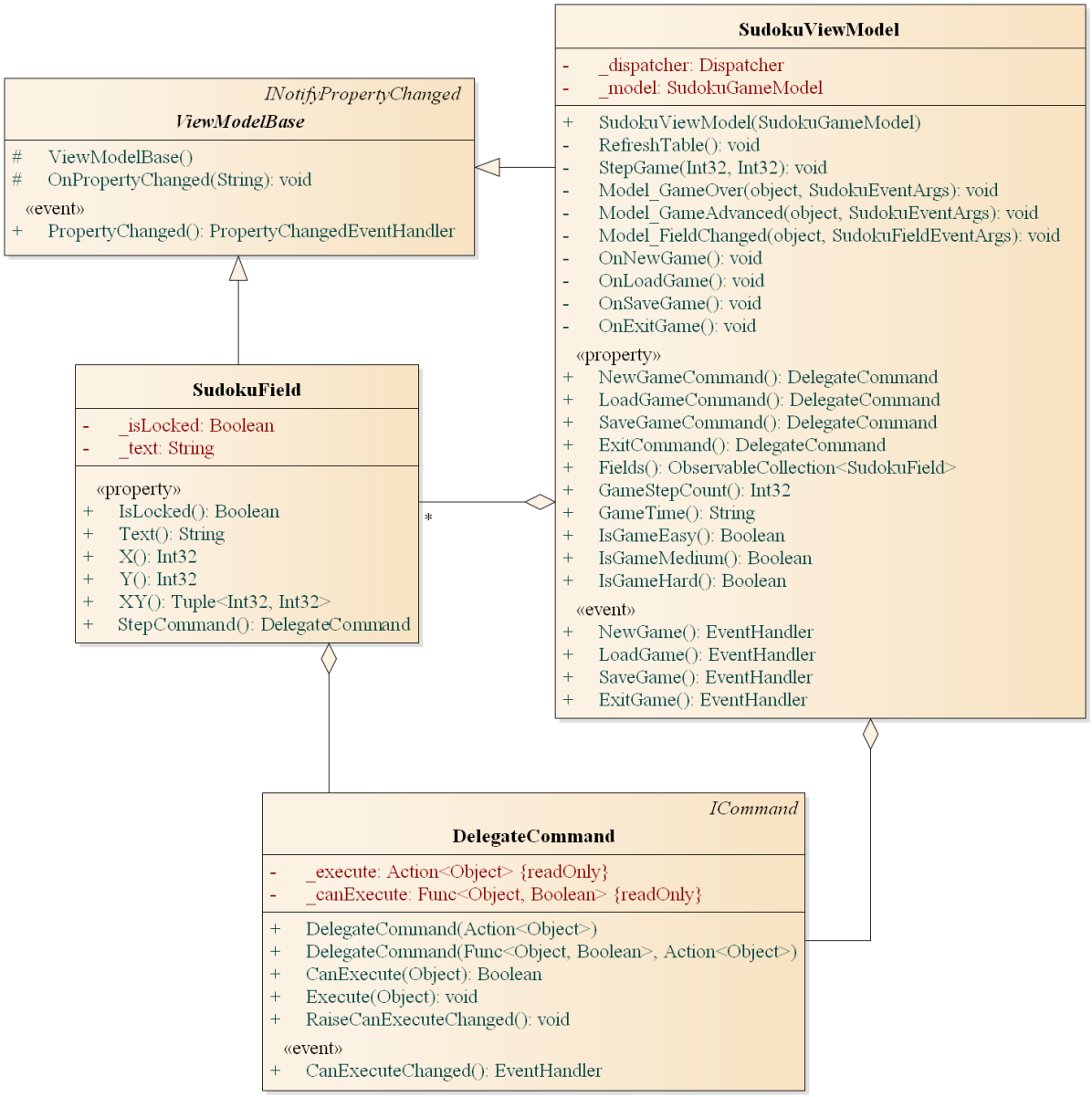
- A játékállapot változásáról a **`GameAdvanced`** esemény, míg a játék végéről a **`GameOver`** esemény tájékoztat. Az események argumentuma (**`SudokuEventArgs`**) tárolja a győzelem állapotát, a lépések számát, valamint a játékidőt.
- A modell példányosításkor megkapja az adatkezelés felületét, amelynek 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 felhasználunk egy általános utasítás (**`DelegateCommand`**), valamint egy ős változásjelző (**`ViewModelBase`**) osztályt.

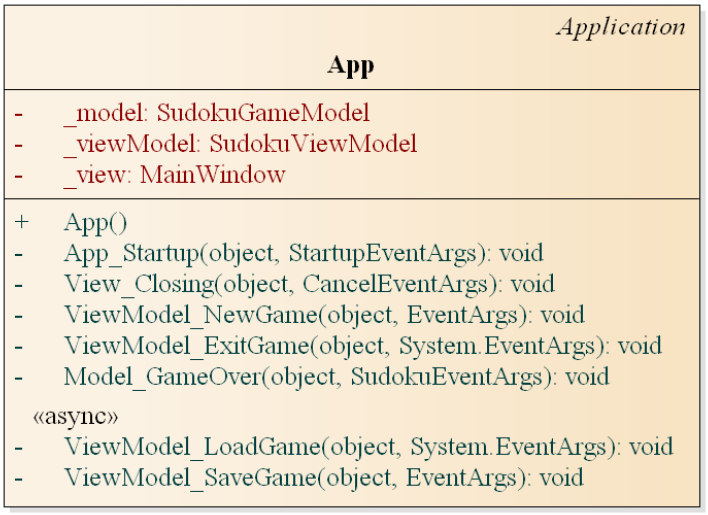
- A nézetmodell feladatait a **`SudokuViewModel`** osztály látja el, amely parancsokat biztosít az új játék kezdéséhez, játék betöltéséhez, mentéséhez, valamint a kilépéshez. A parancsokhoz eseményeket kötünk, amelyek a parancs lefutását jelzik a vezérlőnek. A nézetmodell tárolja a modell egy hivatkozását (**`_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 nézet csak egy képernyőt tartalmaz, a **`MainWindow`** osztályt. A nézet egy rácsban tárolja a játékmezőt, a menüt és a státuszsort. A játékmező egy **`ItemsControl`** vezérlő, ahol dinamikusan felépítünk egy rácsot (**`UniformGrid`**), amely gombokból áll. Minden adatot adatkötéssel kapcsolunk a felülethez, továbbá azon keresztül szabályozzuk a gombok színét is.
  - A fájlnév bekérését betöltéskor és mentéskor, valamint a figyelmeztető üzenetek megjelenését beépített dialógusablakok segítségével végezzük.

- Környezet (6. ábra):
  - Az **`App`** osztály feladata az egyes rétegek példányosítása (**`App_Startup`**), összekötése, a nézetmodell, valamint a modell eseményeinek lekezelése, és ezáltal a játék, az adatkezelés, valamint a nézetek szabályozása.



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.
- **`SudokuGameModelAdcanceTimeTest`**: A játékbeli idő kezelésének ellenőrzése, beleértve a játék végét az idő lejártával.
- **`SudokuGameModelLoadTest`**: A játék modell betöltésének tesztelése mockolt perzisztencia réteggel.