

3. Beadandó feladat dokumentáció

Készítette:

Bárkányi Péter

h4u2hj@inf.elte.hu**Feladat:**

Készítsünk programot a közismert Tetris játékra.

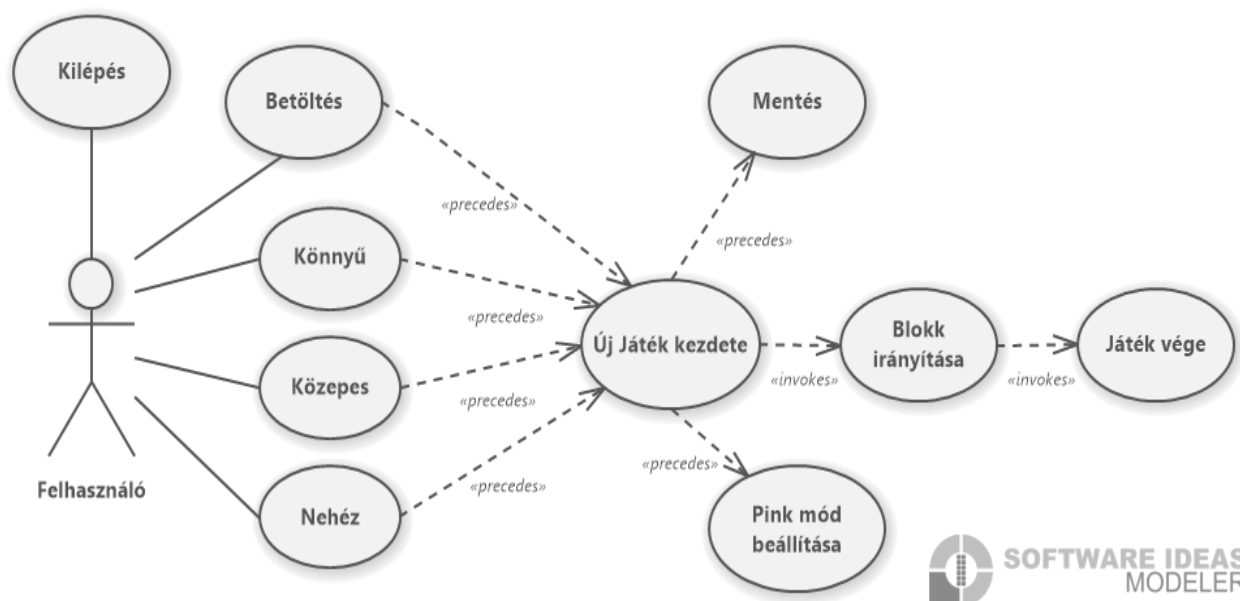
Adott egy $n \times m$ pontból álló tábla, amely kezdetben üres. A tábla tetejéről egymás után új, 4 kockából álló építőelemek hullanak, amelyek különböző formájúak lehetnek (kocka, egyenes, L alak, tető, rombusz). Az elemek rögzített sebességgel esnek lefelé, és az első, nem telített helyen megállnak. Amennyiben egy sor teljesen megtelik, az eltűnik a játékmezőről, és minden felette lévő kocka eggyel lejjebb esik.

A játékosnak lehetősége van az alakzatokat balra, jobbra mozgatni, valamint forgatni óramutató járásával megegyező irányba, így befolyásolhatja azok mozgását. A játék addig tart, amíg a kockák nem érik el a tábla tetejét.

A program biztosítson lehetőséget új játék kezdésére a pályaméret megadásával (4×16 , 8×16 , 12×16), valamint játék szüneteltetésére (ekkor nem telik az idő, és nem mozognak az elemek). Ismerje fel, ha vége a játéknak, és jelenítse meg, mennyi volt a játékidő. Ezen felül szüneteltetés alatt legyen lehetőség a játék elmentésére, valamint betöltésére.

Elemzés:

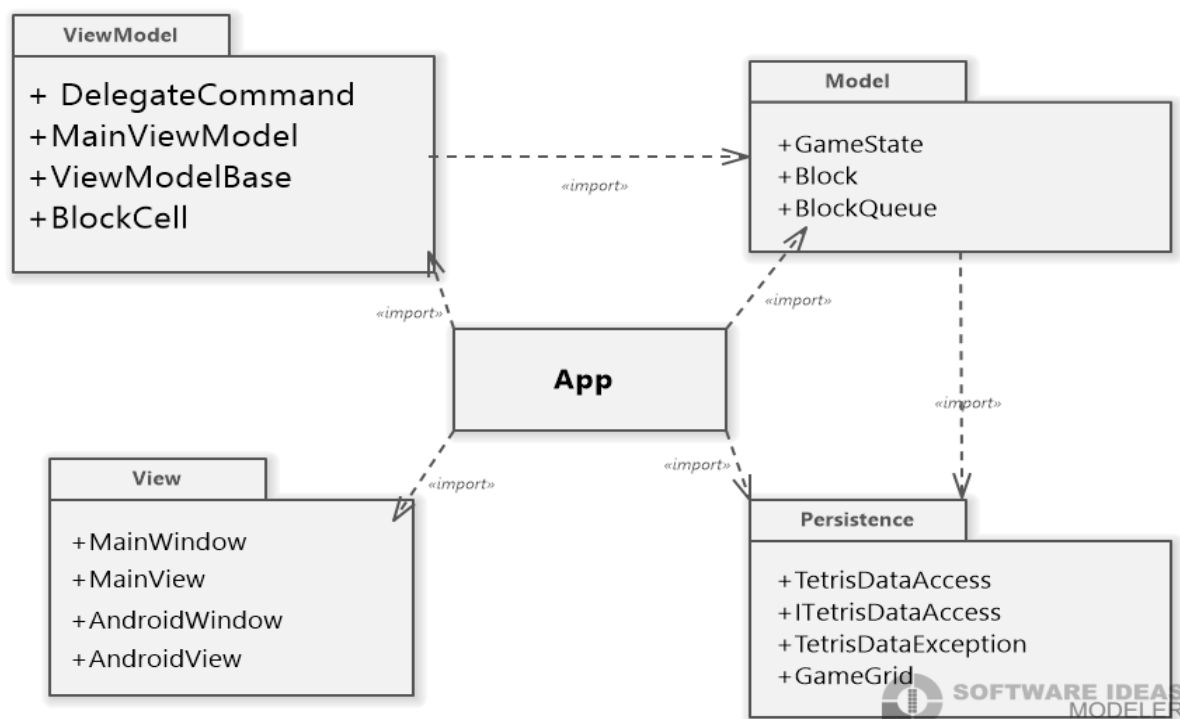
- A játékot három nehézségi szinttel játszhatjuk: könnyű (4 oszlop szélességgel), közepes (8 oszlop szélességgel), nehéz (12 oszlop szélességgel). A program indításakor ki kell választani a nehézséget, ekkor elindul a játék.
- A feladatot egyablakos asztali alkalmazásként Windows Presentation Foundation grafikus felülettel valósítjuk meg.
- Az ablak bal oldalán elhelyezünk egy Load és egy Save gombot. Ezekkel új játékot tudunk betölteni, illetve elmenteni. A jobb oldalon tudjuk gombokkal kiválasztani a nehézséget, majd megjelenik egy Pause és egy New Game gomb melyekkel szüneteltethetjük és új játékot kezdhetünk. Felettük megjelenik a pontszámunk, valamint a játék végén az eltelt idő. A jobb oldalon található még a Pink Mode gomb mellyel pink színűre állíthatjuk a játékot.
- A játéktáblát egy $4/8/12 \times 16$ képekből álló rács reprezentálja. A képek forrásának megváltoztatásával jelenik meg a pályán egy blokk vagy üres cella. A blokkokat lehet forgatni mindkét irányba és lehet őket leejteni alulra.
- A játék megjeleníti a végső pontszámunkat és eltelt időt amikor vége a játéknak (a rács tetejére értünk és nincs már hely blokkoknak). A játékban dialógusablakokkal végezzük el a mentést, illetve betöltést, a fájln neveket a felhasználó adja meg.
- A felhasználói 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 nézetmodellt és a nézetet, biztosítja a kommunikációt, valamint felügyeli az adatkezelést. Az alkalmazás környezeti réteg feladata a platformfüggő alkalmazás életciklus helyénvaló kezelése is.
- A program csomagdiagramja a 2. ábrán látható.

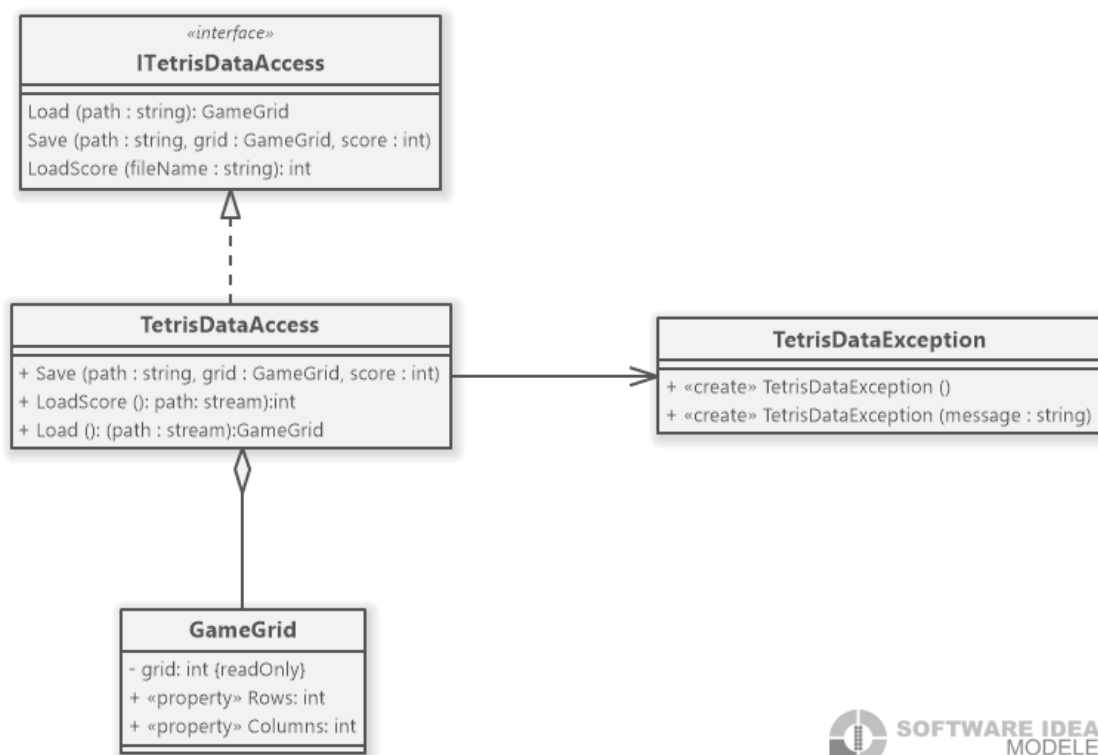


2. ábra: Az alkalmazás csomagdiagramja

- **Perzisztencia (3. ábra):**

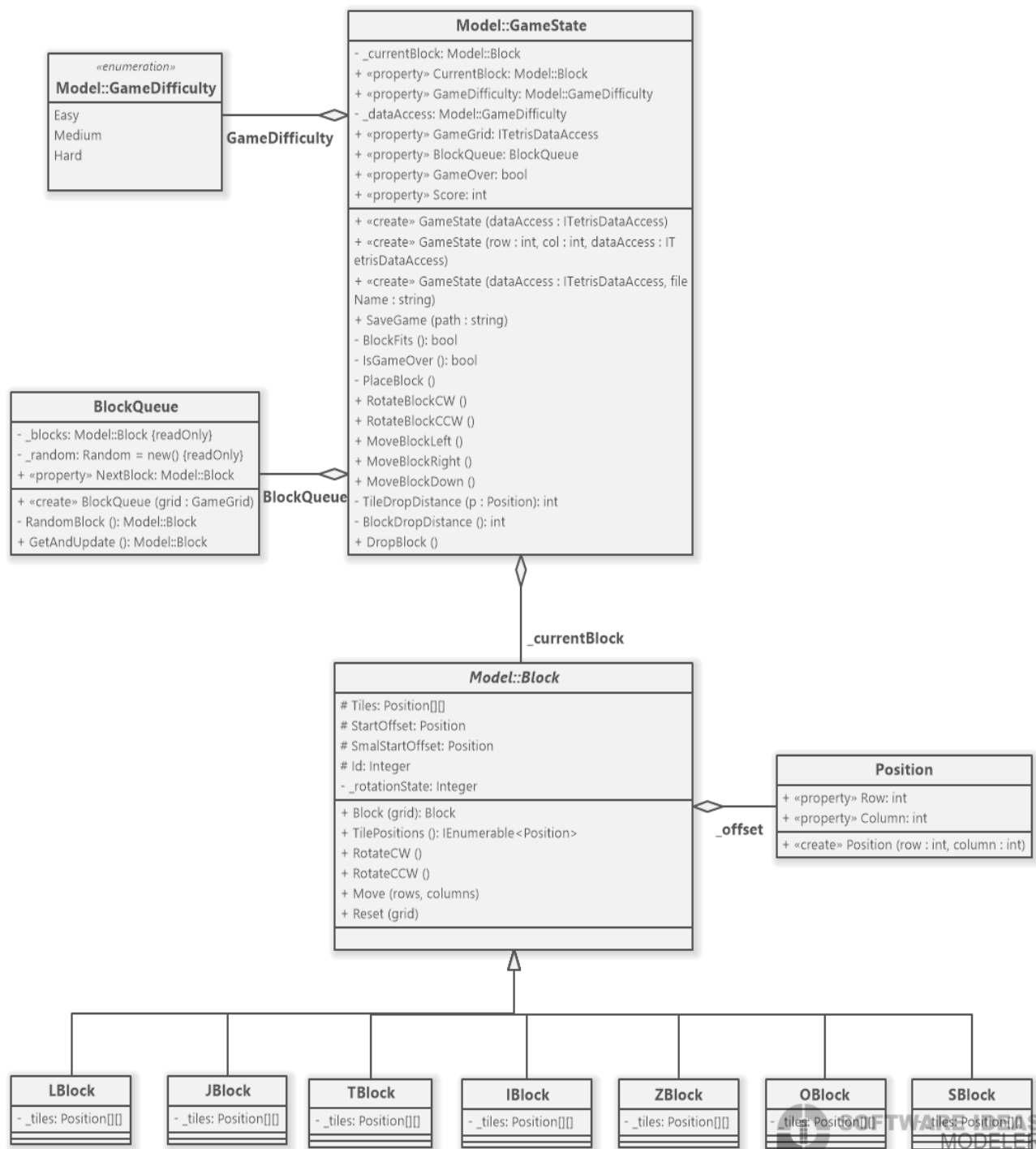
- Az adatkezelés feladata a Tetris rácsával kapcsolatos információk tárolása, valamint a betöltés/mentés biztosítása.
- A **GameGrid** osztály egy érvényes Tetris játékhoz szükséges rácsot biztosít kezdetben mindenhol üreset. **Rows** illetve **Columns** tulajdonságaival amiket a konstruktor paraméterben kap, tároljuk a rács sor és oszlopainak a számát. (**IsEmpty**, **IsInside**, **IsRowFull**, **IsSmall**,) metódusokkal tudunk adatokat lekérni a rácsról. (**ClearFullRows**, **MoveRowDown**, **ClearRow**) műveletekkel pedig a rács sorait tudjuk módosítani.
- A hosszú távú adattárolás lehetőségeit az **ITetrisDataAccess** interfész adja meg, amely lehetőséget ad a rács betöltésére (Load), valamint mentésére (Save). Illetve a pontszám betöltésére (LoadScore)
- Az interfészt szöveges fájl alapú adatkezelésre a **TetrisDataAccess** osztály valósítja meg. A fájlkezelés során fellépő hibákat a **TetrisDataException** kivétel jelzi.
- A program az adatokat szöveges fájlként tudja eltárolni, melyek az **txt** kiterjesztést kapják. Ezeket az adatokat a program első indításakor lehet betölteni, illetve a játék szüneteltetésekor ki lehet menteni az aktuális állapotot.

- A fájl első sora megadja az elmentett pontszámunkat. A következő sor a rács mentett állapotát adja meg, amely állhat 72, 144, 216 számból, az elmentett játék nehézségétől függően. A számok 0-7 közöttiek lehetnek, ahol 0 reprezentálja a még üres mezőt, a többi pedig az egyes blokkok típusát.
- A hosszú távú adattárolás lehetőségeit az **ITetrisDataAccess** interfész adja meg, amely lehetőséget ad a tábla betöltésére (**Load**), valamint mentésre (**Save**). A műveleteket hatékonysági okokból aszinkron módon valósítjuk meg. A mobilos platformok támogatása érdekében a műveleteket adatfolyamok (**Stream**) támogatásával is megvalósítjuk



3. ábra: A Persistence csomag osztálydiagramja

-
- **Modell (4. ábra):**
 - A modell lényegi részét a **GameState** osztály valósítja meg, amely szabályozza a játék tevékenységeit, valamint egyéb paramétereit, úgymint a pontszám (**Score**) és a játék végét (**GameOver**). Új játék kezdésére a konstruktor felel, valamint blokk megjelenésére a (**MoveBlockDown**). Új játéknál megadható a kiinduló rács mérete is, különben a legnagyobb rács generálódik.
 - A jelenlegi blokk mozgatásáról és manipulálásáról a **MoveBlockLeft**, **MoveBlockRight**, **RotateBlockCW**, **DropBlock** metódusok felelnek.
 - A modell példányosításkor megkapja az adatkezelés felületét, amelynek segítségével lehetőséget ad betöltésre (**Konstruktor**) és mentésre (**SaveGame**).
 - A játék nehézségét a **GameDifficulty** felsorolási típuson át kezeljük, és a **GameGrid** osztályban a rács méreteivel tároljuk az egyes nehézségeket.

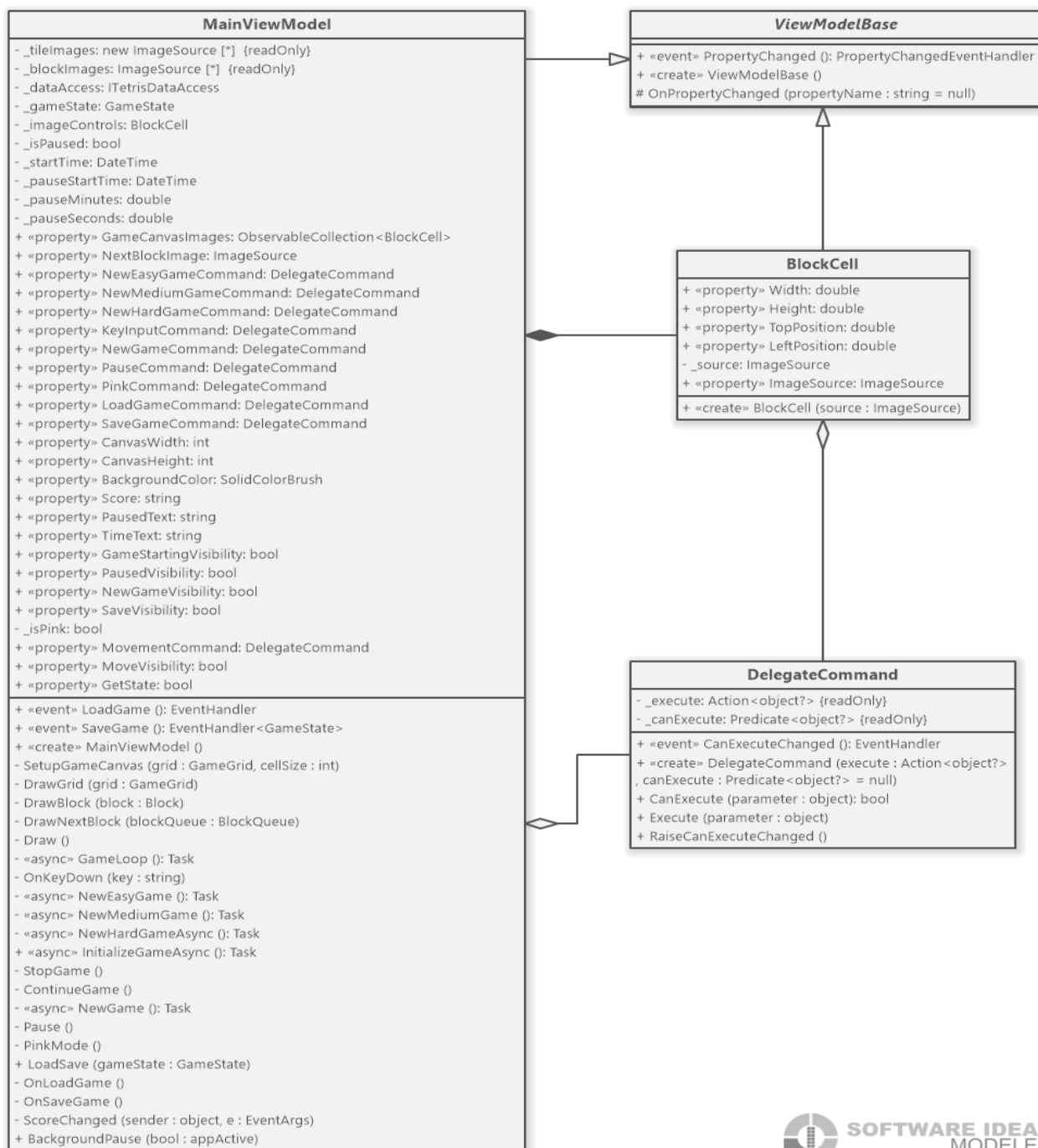


4. ábra: A Model csomag osztálydiagramja

- **Nézetmodell (5. ábra):**

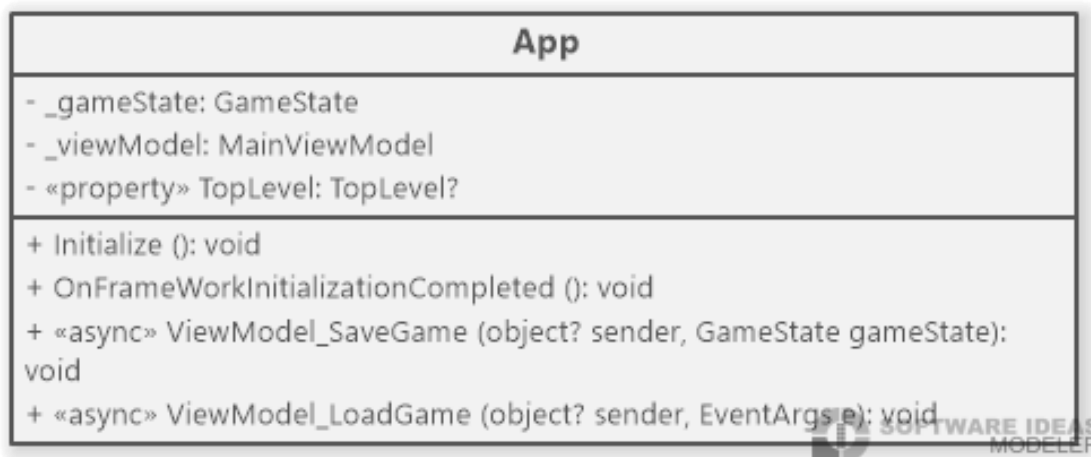
- A nézetmodell megvalósításához felhasználunk egy általános utasítás (**DelegateCommand**), valamint egy ős változásjelző (**ViewModelBase**) osztályt.
- A nézetmodell feladatait a **MainViewModel** osztály látja el, amely parancsokat biztosít az új játék kezdéséhez, játék betöltéséhez, mentéséhez, valamint a kilépéshez. A parancsokhoz eseményeket kötünk, amelyek a

- parancs lefutását jelzik a vezérőnek. A nézetmodell tárolja a modell egy hivatkozását (`_gameState`), de csupán információkat kér le tőle, illetve a játéknehézséget, pálya méretét szabályozza. Direkt nem avatkozik a játék futtatásába.
- Egy kocka számára egy külön mezőt biztosítunk (`BlockCell`), amely eltárolja a pozíciót, forrás képet, méretet. A mezőket egy felügyelt gyűjteménybe helyezzük a nézetmodellbe (`GameCanvasImages`).



5. ábra: A nézetmodell osztálydiagramja

- **Nézet:**
 - A nézetet ablakok, egyedi vezérlők és dialógusablakok használatával valósítjuk meg.
 - A **MainView** osztály, mint **UserControl** leszármazott tartalmazza az asztali számítógép felületét.
 - A **AndroidView** osztály, mint **UserControl** leszármazott tartalmazza a Android telefon felületét.
 - A nézet egy rácsban tárolja a játékmezőt, a bal és a jobb menü részt. A játékmező egy **ItemsControl** vezérlő, ahol dinamikusan felépítünk egy rácsot (**Canvas**), amely képekből áll. Minden adatot adatkötéssel kapcsolunk a felülethez, továbbá azon keresztül szabályozzuk a cellák hátterét is.
 - A fájlnev bekérését betöltéskor és mentéskor, valamint a figyelmeztető üzenetek megjelenését beépített dialógusablakok segítségével végezzük
 - A **MainWindow** ablakba egyszerűen a **MainView** vezérlőt ágyazzuk be. Ilyen módon a felület asztali alkalmazásokban ablakos alkalmazásként, mobil platformon pedig **AndroidView** jeleníti meg a **AndroidView** vezérlőt.
 - 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**) szüneteltetjük a játékot a Pause gomb háttérben lenyomásával míg folytatáskor vagy újraindításkor (**Activated**) pedig folytatjuk a játék menetet.



6. ábra: A vezérlés osztálydiagramja

Tesztelés:

- A modell funkcionalitása egységtesztek segítségével lett ellenőrizve a **TetrisUnitTest** osztályban.
- Az alábbi tesztesetek kerültek megvalósításra:
- **SmallGridTest, MediumGridTest, LargeGridTest**: A rácsok méretei és tartalma ellenőrzése.
- **NewSmallGameTest, NewMediumGameTest, NewHardGameTest**: Új játék kezdő tulajdonságainak tesztelése, pontszám, nehézség, rács állapota, betöltés.
- **CreateAndPlaceBlockTest**: Új blokk létrehozása és alulra helyezése. A rács új állapotának ellenőrzése.
- **PlaceBlockLeft, PlaceBlockRight**: Egy új blokk teljesen bal vagy jobb szélre helyezése és annak tesztelése, hogy jó helyen állt meg a blokk.
- **GameOverTest**: Blokkok egymásra helyezése, hogy elérjük a játék végét. Annak tesztelése, hogy a játék leáll-e.