1. Beadandó feladat dokumentáció

Készítette:

Restye János Barnabás

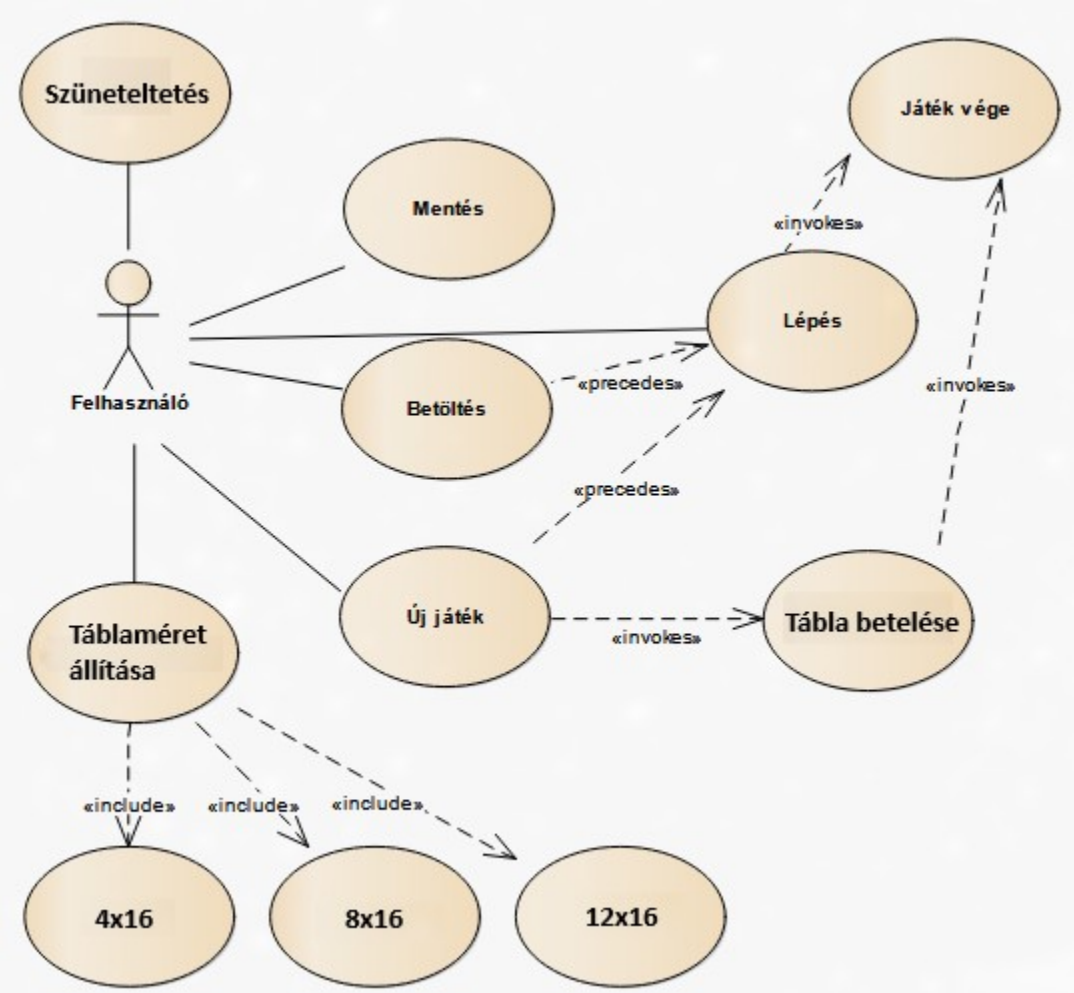
f8u9i2@inf.elte.hu

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:

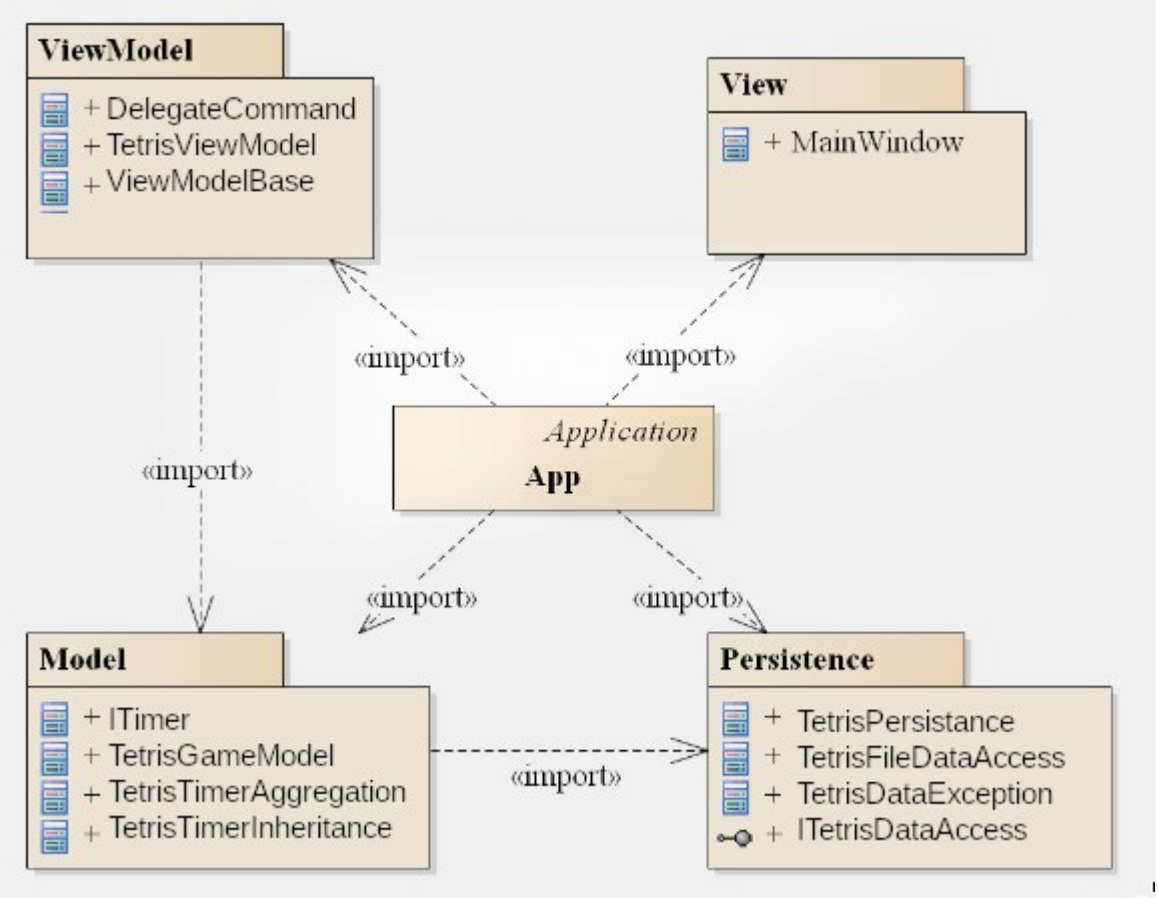
- A játékot három táblamérettel játszhatjuk: 4x16, 8x16 és 12x16. A program indításakor a játékos játék megkezdése előtt lehetlőséget kap ezek közül választani.
- A feladatot egyablakos asztali alkalmazásként Windows Presentation Foundation grafikus felülettel valósítjuk meg.
- Az ablak bal oldalán lesz a tábla, a jobb oldalán a menü a következő menüpontokkal: Új játék, Szünet, Mentés, Betöltés. A menüpontok alatt lesz a időszámláló.
- A játéktábla a játékos által választott méretnek fog megfelelni. Új játék menüpont választása után a játék elindul egy tetrominó megjelenésével. A játékos a tetrominó mozgását a WASD gombokkal tudja irányítani.
- Ha a játékos veszít, egy dialógusablak értesíti a játék végéről. Dialógusablak értesíti még a játékost a játék sikeres mentése és betöltése után.
- A felhasználói esetek az 1. ábrán láthatóak.



1. ábra felhasználói esetek diagramja

Tervezés:

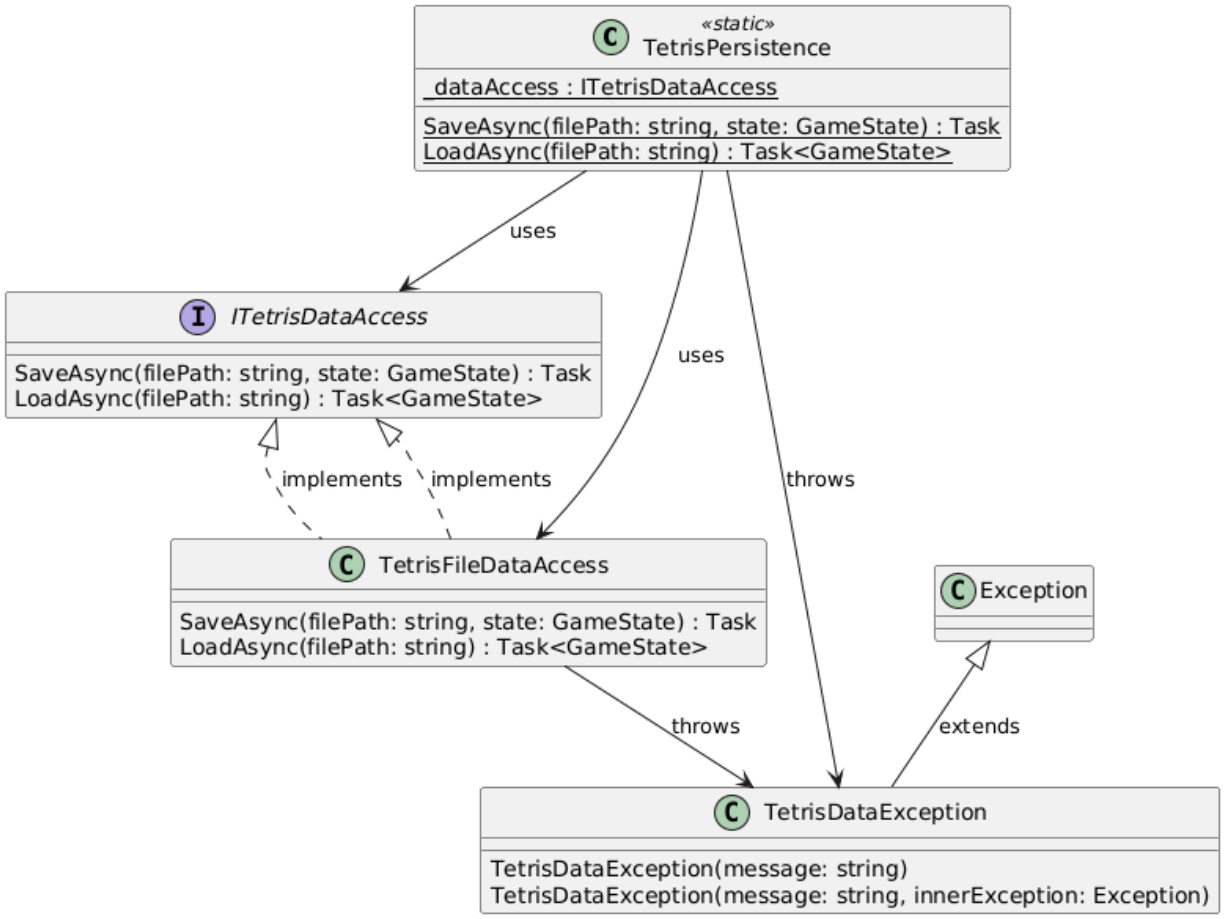
- A programszerkezet:
  - A programot MVVM architektúrában valósítjuk meg, ennek megfelelően View, Model, ViewModel és Persistence névtereket valósítunk meg az alkalmazáson belül. A program környezetét az alkalmazás osztály (App) végzi, amely példányosítja a modellt, a nézetmodell és a nézetet, biztosítja 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:

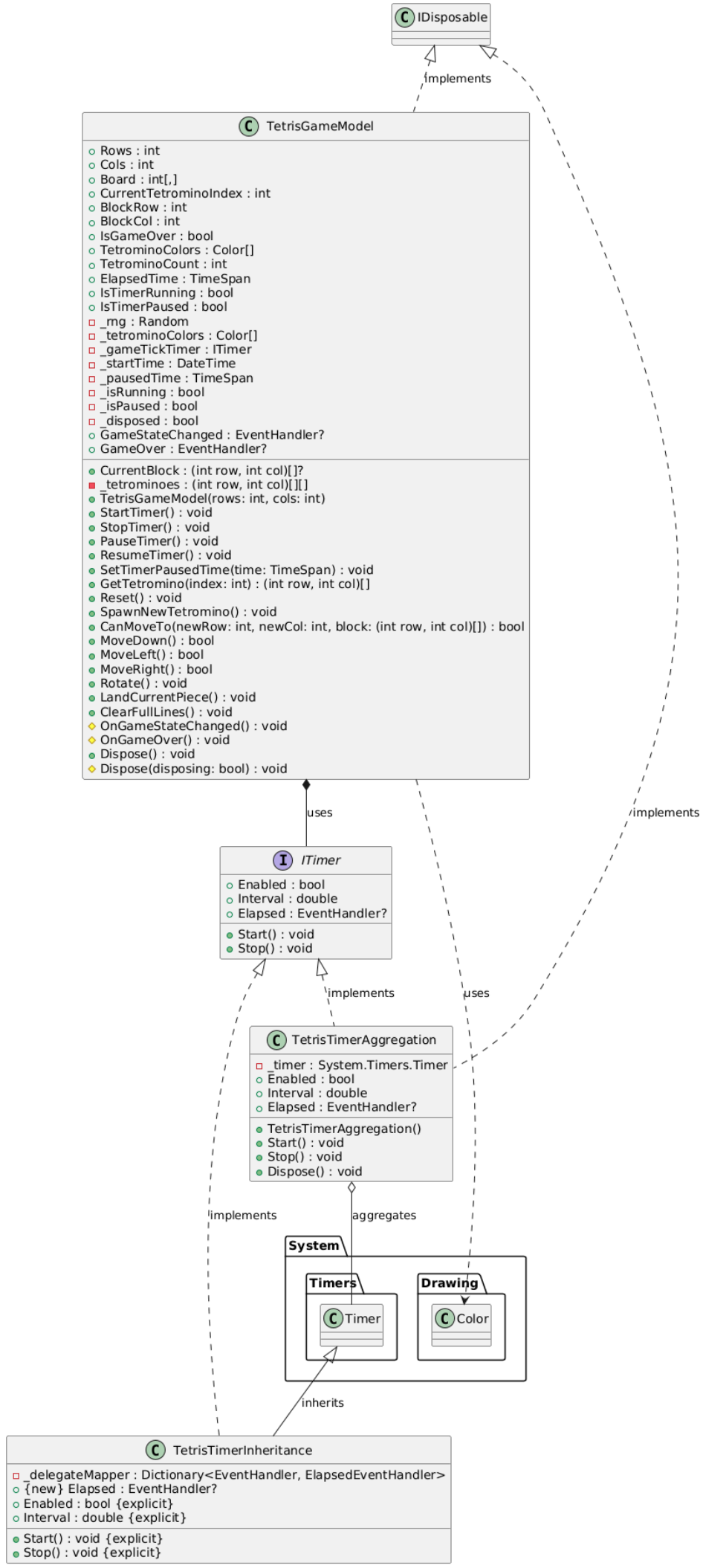
- Az adatkezelés feladata a Tetris táblával kapcsolatos információk tárolása, valamint a betöltés/mentés biztosítása. A TetrisPersistence osztály egy érvényes Tetris táblát biztosít (azaz mindig ellenőrzi a beállított értékeket). A tábla tartalmazza a méretet (Rows, Cols), a tábla tartalmát (Board), a jelenleg aktív tetrominó indexét és dimenzióit (CurrentTetrominoIndex, CurrentBlock), stb.
- A hosszú távú adattárolás lehetőségeit az ITetrisDataAccess 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 TetrisFileDataAccess osztály valósítja meg. A fájlkezelés során fellépő hibákat a TetrisDataException kivétel jelzi.



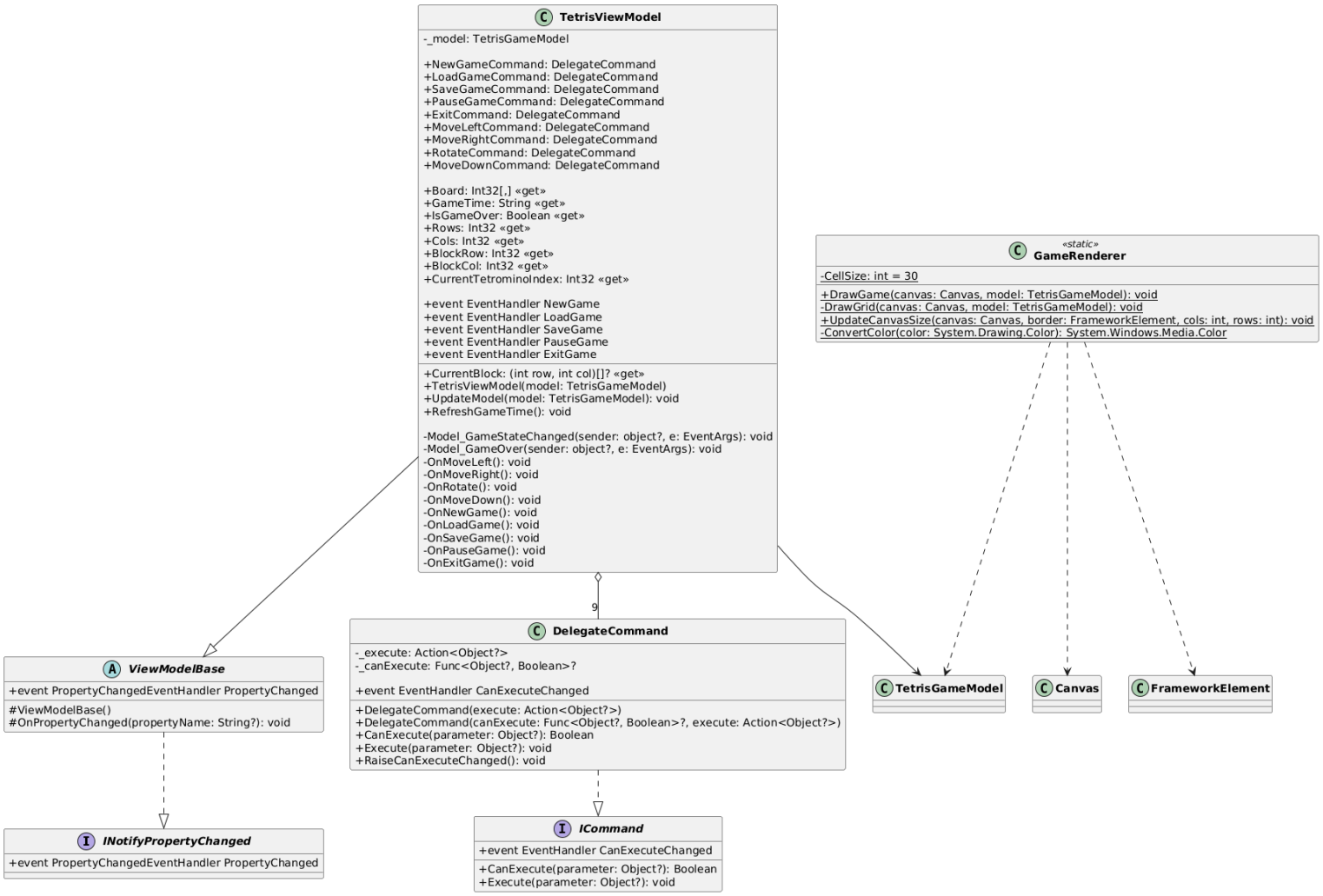
3. ábra: Az Persistance csomag osztálydiagramja

- Modell (4. ábra):
  - A modell lényegi részét a TetrisGameModel osztály valósítja meg, amely szabályozza a tábla tevékenységeit, valamint a játék egyéb paramétereit. Új játéknál megadható a kiinduló játéktábla is.

- A játék időbeli kezelését egy időzítő végzi (_timer), amelyet inaktiválunk majd (PauseGame), amennyiben bizonyos menüfunkciók futnak, majd újraindítjuk (ResumeGame).
- A játékállapot megváltozásáról a GameStateChange esemény, míg a játék végéről a GameOver esemény tájékoztat. Az események argumentuma (TetrisGameEventArgs) tárolja a győzelem állapotát, a lépések számát, valamint a játékidőt.
- A modell példányosításkor megkapja az adatkezelés felületét, amelynek segítségével lehetőséget ad betöltésre (LoadGameAsync) és mentésre (SaveGameAsync)


IDisposable
implements
TetrisGameModel
+ Rows : int
+ Cols : int
+ Board : int[,]
+ CurrentTetrominoIndex : int
+ BlockRow : int
+ BlockCol : int
+ IsGameOver : bool
+ TetrominoColors : Color[]
+ TetrominoCount : int
+ ElapsedTime : TimeSpan
+ IsTimerRunning : bool
+ IsTimerPaused : bool
- _rng : Random
- _tetrominoColors : Color[]
- _gameTickTimer : ITimer
- _startTime : DateTime
- _pausedTime : TimeSpan
- _isRunning : bool
- _isPaused : bool
- _disposed : bool
+ GameStateChanged : EventHandler?
+ GameOver : EventHandler?
+ CurrentBlock : (int row, int col)[]?
- _tetrominoes : (int row, int col)[][]
+ TetrisGameModel(rows: int, cols: int)
+ StartTimer() : void
+ StopTimer() : void
+ PauseTimer() : void
+ ResumeTimer() : void
+ SetTimerPausedTime(time: TimeSpan) : void
+ GetTetromino(index: int) : (int row, int col)[]
+ Reset() : void
+ SpawnNewTetromino() : void
+ CanMoveTo(newRow: int, newCol: int, block: (int row, int col)[]) : bool
+ MoveDown() : bool
+ MoveLeft() : bool
+ MoveRight() : bool
+ Rotate() : void
+ LandCurrentPiece() : void
+ ClearFullLines() : void
# OnGameStateChanged() : void
# OnGameOver() : void
+ Dispose() : void
# Dispose(disposing: bool) : void
uses
implements
ITimer
+ Enabled : bool
+ Interval : double
+ Elapsed : EventHandler?
+ Start() : void
+ Stop() : void
implements
uses
TetrisTimerAggregation
- _timer : System.Timers.Timer
+ Enabled : bool
+ Interval : double
+ Elapsed : EventHandler?
+ TetrisTimerAggregation()
+ Start() : void
+ Stop() : void
+ Dispose() : void
implements
aggregates
System
Timers
Timer
Drawing
Color
inherits
TetrisTimerInheritance
- _delegateMapper : Dictionary<EventHandler, ElapsedEventHandler>
+ {new} Elapsed : EventHandler?
+ Enabled : bool {explicit}
+ Interval : double {explicit}
+ Start() : void {explicit}
+ Stop() : void {explicit}

- 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.



- 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.

- 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 TetrisGameModelTest osztályban.
  - A modell időzítőjét egy ITimer interfészből származtattuk le, így azt a teszt projektben egy MockTimer megvalósítással mockolhatjuk.
  - Az alábbi tesztesetek kerültek megvalósításra:
    - InitializeTest,
    - ResetTest,
    - MoveDownTest,
    - MoveRightTest,
    - MoveDownBlockedTest,
    - LandPieceTest,
    - SpawnTetrominoTest,
    - ClearLineTest,
    - DetectWallsTest,
    - RotatesBlockCorrectly