3. Beadandó feladat dokumentáció

Készítette:

Trembickij Máté
Neptun: miszuu
E-mail: miszuu@inf.elte.hu

Feladat:

Készítsünk programot a közismert Tetris játékra.

Adott egy × 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.

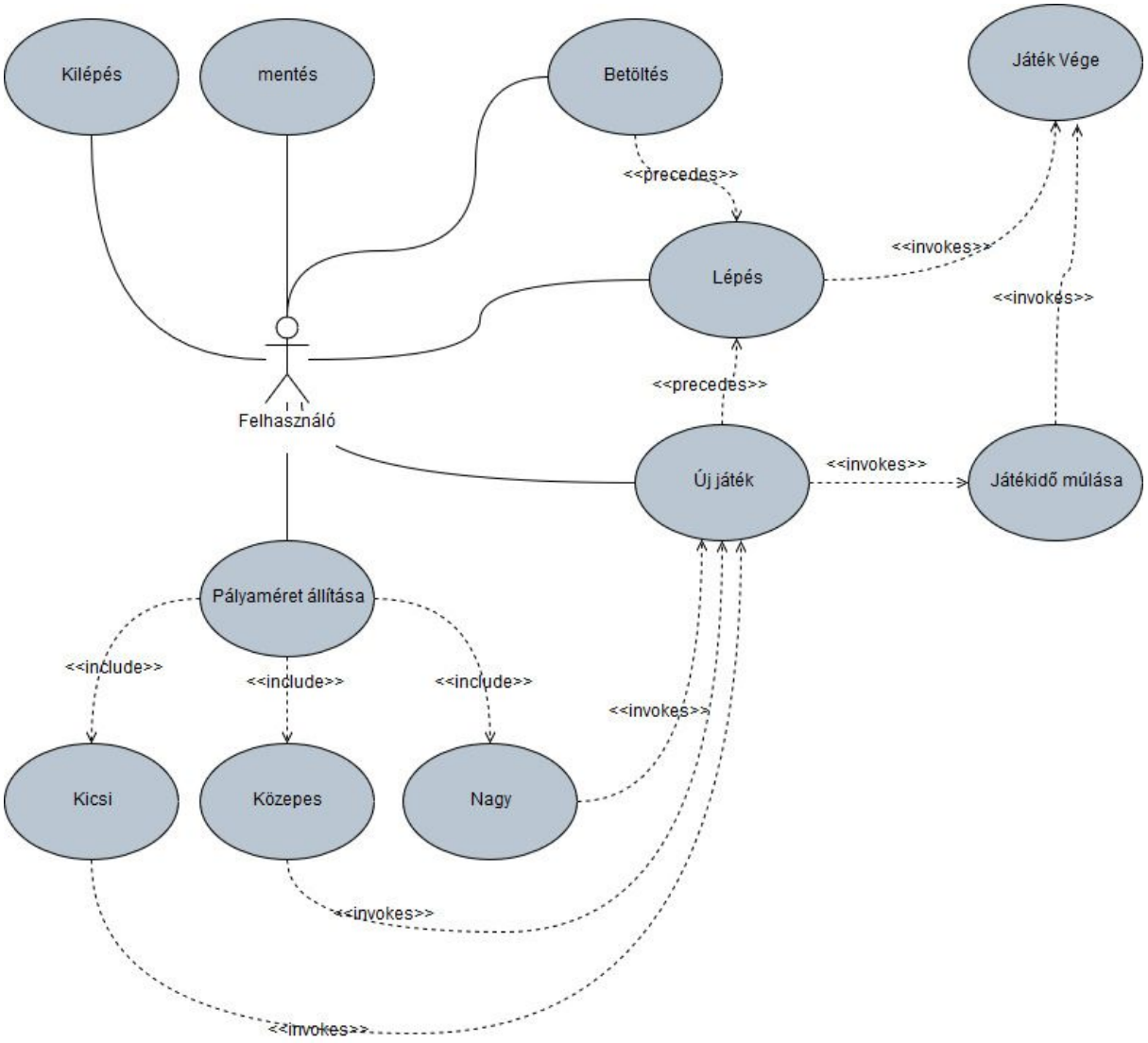
A program biztosítson lehetőséget új játék kezdésére a pályaméret megadásával (4 × 16, 8 × 16, 12 × 16), valamint játék szüneteltetésére (ekkor nem telik az idő, és nem mozognak az elemek). Ismerje fel, ha vége a játéknak, és jelenítse meg, mennyi volt a játékidő. Ezen felül legyen szüneteltetés alatt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

Elemzés:

- A játékot három pályamérettel játszhatjuk: létezik a kicsi (4 sor, 16 oszlop), közepes (8 sor, 16 oszlop), nagy (12 sor, 16 oszlop). A program indításakor közepes méretű pályamérettel, és

el is indul egy új játék. A játék méretét a felül lévő menüben lehet állítani (size-on belül small/medium/large). A gombok megnyomásával azonnal új játék indul.

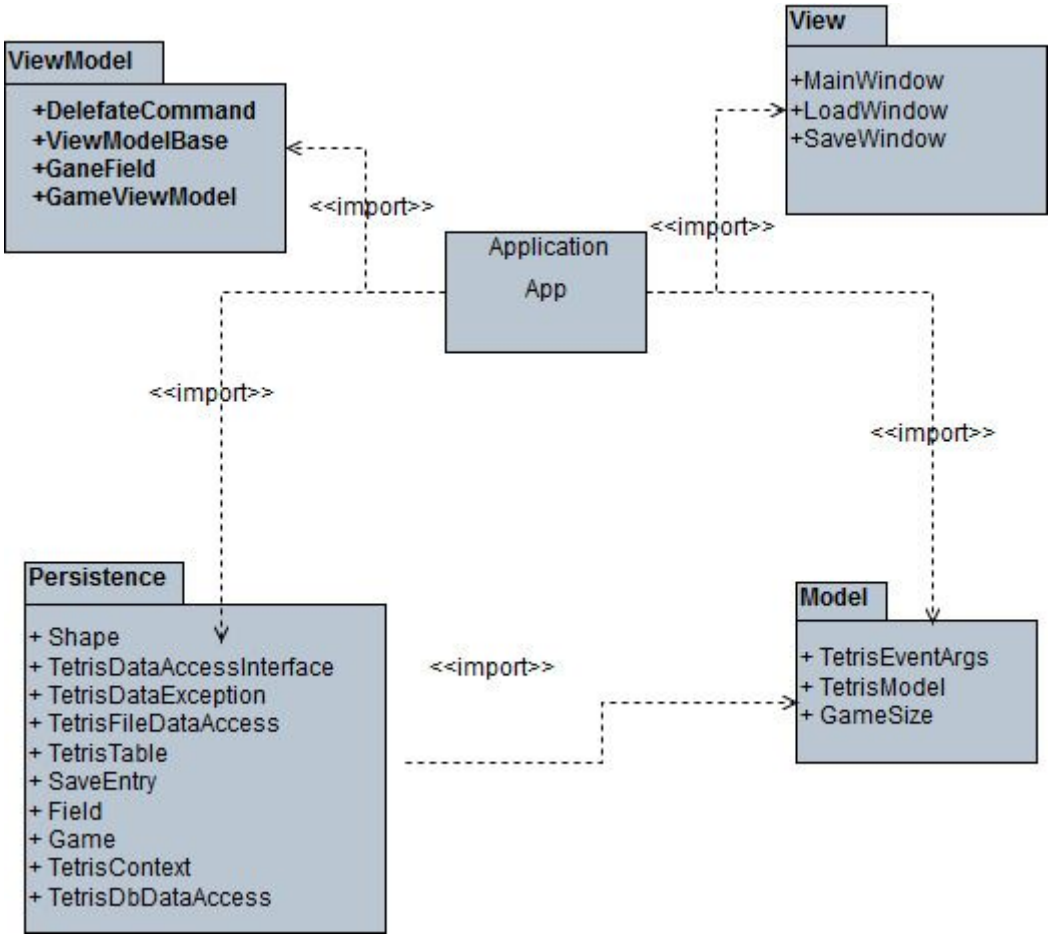
- A feladatot egyablakos asztali alkalmazásként Windows Presentation Foundation grafikus felülettel valósítjuk meg.
- Az ablakban felül található egy menü a következő menüpontokkal: File (New Game, Size, Save, Load, Exit), Pause gomb amivel a játékot szüneteltethetjük. Az ablak alján megjelenítünk egy státuszsort, amely az eltelt időt jelzi.
- A játéktáblát egy (4x16, , 8 x 16, 12 x 16) gombokból álló rács reprezentálja. A nyomógombok, nem funkcionálnak másra, csak a játék megjelenítésére. A játékot a billentyűzettel irányítja a játékos azon belül a nyilakkal. A bal, le illetve jobb nyíl a vele megegyező irányba mozgatja az alakzatot, míg a felfelé nyíl forgatja.
- Az alakzatok a leérkezéskor (amikor nem tudnak már tovább leesni), megszilárdulnak, s beleolvadnak a játéktáblába. Ezzel együtt megidéznek egy újabb alakzatot a kezdőpontban, ha nem sikerül ez a cselekedet, a játék befejeződik.
- A játék automatikusan feldob egy dialógus ablakot, amikor vége van a játéknak. A dialógusablak megjeleníti az időt. Szintén dialógusablak az ami kezeli a játék mentését illetve a betöltését. A fájlnevet a felhasználó adhatja meg.
- A felhasználói esetek az 1. ábrán láthatóak.



1. ábra: Felhasználói esetek diagramja

Tervezés:

- Programszerkezet:
  - A programot háromrétegű architektúrában valósítjuk meg. A megjelenítés a **View**, a modell a **Model**, míg a perzisztencia a **Persistence** névtérben helyezkedik el.

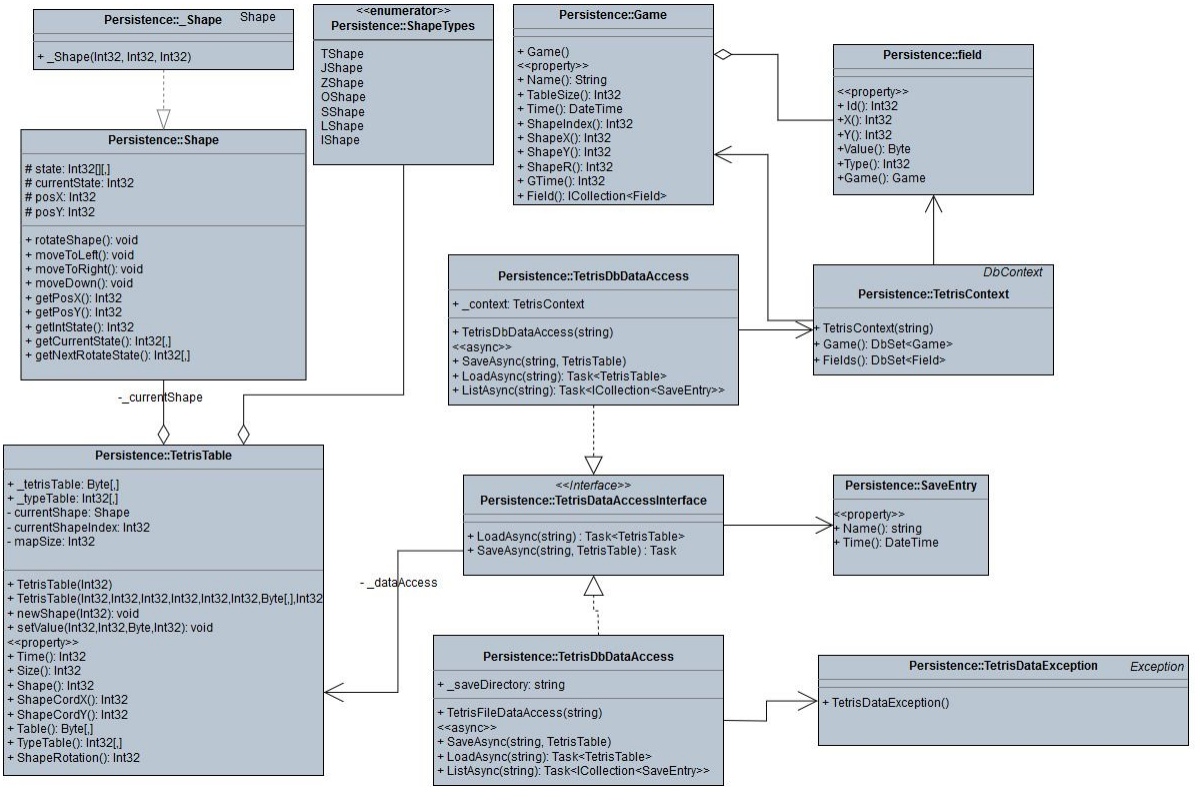


Az alkalmazás csomagdiagrammja

- Perzisztencia:
  - Az adatkezelés feladata a Tetris táblával kapcsolatos információk tárolása, valamint a betöltés/mentés biztosítása.
  - A TetrisTable osztály egy érvényes Tetris táblát biztosít, ahol minden mező értéke ismert az értéke (**_tetrisTable**) (tehát hogy van ott valami, fixen van ott valami, vagy pedig ideiglenesen van-e ott valami), és ismert a típuskódja is (**_typeTable**). Mindkettő a játék kezdetekkor generált, kezdőértékekkel. A tábla alapértelmezetten (8+1) x 16-os, ez a konstruktorban változtatható. A tábla lehetőséget ad különböző állapotok elérésére (**Table,TypeTable, ShapeRotation, ShapeCordY, ShapeCordX, Shape, Size, Time, CurrentShape, CurrentShapeIndex**).
  - A Shape osztály egy alakzatot reprezentál, ami definiálja a lehetséges állapotait (**state**), jelenlegi állapotindexét (**currentState**), és a koordinátáját (**posX,posY**). Ennek vannak különböző függvényei amik változtatják a különböző tulajdonságait (**moveToLeft, moveToRight, rotateShape, moveDown**), meg különböző

függvények, amikkel lekérdezhető az állapotaik (**getPosX, getPosY, getIntState, getCurrentState, getNextRotateState)**.

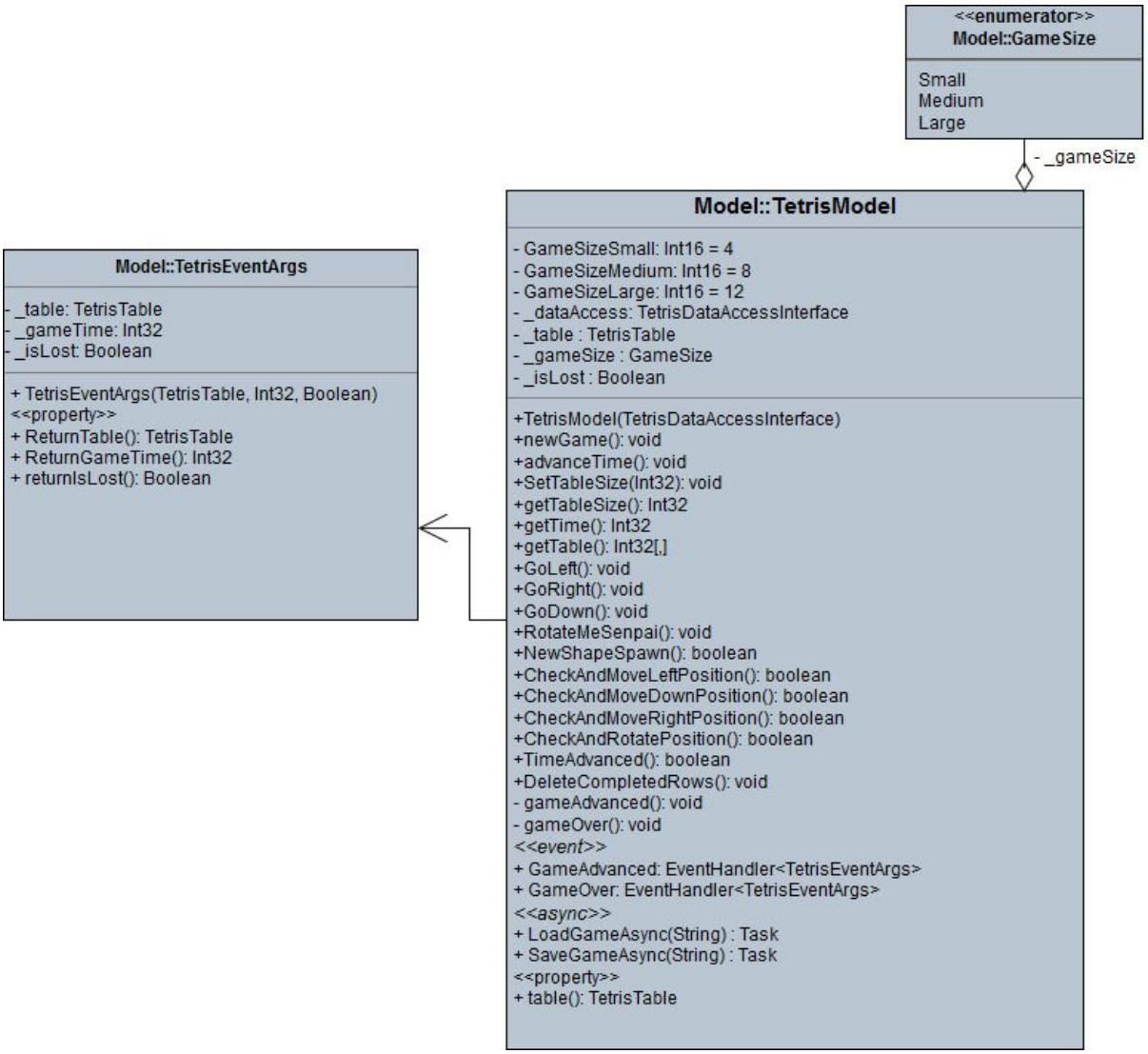
- A hosszú távú adattárolást a **TetrisDataAccessInterface** szolgáltatja, amely lehetővé teszi egy elmentett játékállás betöltését (**LoadAsync**) avagy elmentését (**SaveAsync**), és kilistázását (**ListAsync**). A műveleteket hatékonysági okokból aszinkron módon valósítjuk meg.
- Az interfészt szöveges alapú adatkezelésre a **TetrisFileDataAccess** osztály valósítja meg, a fájlkezelés során történő hibákat pedig a **TetrisDataException** kivétel jelzi.
- Az interfészt adatbázis alapú adatkezelésre a **TetrisDbDataAccess** osztály valósítja meg. Az adatbáziskezelés az Entity Framework használatával a **Field** és **Game** entitás típusokkal és a **TetrisContext** adatbázis kontextussal történik. Az Adatkezelés során fellépő hibákat a **TetrisDataException** kivétel jelzi.
- A program az adatokat szöveges fájlként is tudja tárolni tárolja, amelyek a **tt** 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 (4/8/12), valamint az időt. A következő sorban található az aktuális alakzatról való fontos információk, azaz az alakzat típusa, alakzat x koordinátája, y koordinátája és az alakzat forgásindexe. A fájl többi része izomorf leképezése a játéktáblának, összesen méret+1 sor és 16 oszlop található szóközökkel elválasztva. Ez a mátrix kétszer található meg, mivel a másodikban a táblán lévőknek a színkódja található (avagy az alakzat típusa).



Az alkalmazás Perzisztenciája

- Modell:
    - A modell lényegi részét a **TetrisModel** osztály valósítja meg, amely szabályozza a tábla tevékenységeit, valamint a játék egyéb paramétereit mint az, hogy veszített e az állás (**_isLost**) és a játék méretét (**_gameSize)**. Az osztály lehetőséget ad új játék kezdésére (**newGame**), valamint a játék léptetésére (**advanceTime)**.
    - A játékállapot változásáról a **gameAdvanced** esemény, míg a játék végéről a **gameOver** esemény ad tájékoztatást. Az események argumentuma (**TetrisEventArgs)** tárolja a győzelem állapotát, a játék idejét, illetve hogy vége van-e játéknak vagy sem.
    - A modell példányosításkor megkapja az adatkezelés felületét, amelynek segítségével lehetőséget ad beöltésre (**LoadGameAsync**) és mentésére (**SaveGameAsync**).
    - A játék méretét a **_gameSize** felsorolási típuson át kezeljük, és a **TetrisModel** osztályban constansként tároljuk az egyes értékeket.
    - A játékban lévő **Shape** mozgatását és annak különböző ellenőrzéseit a következő metódusok segítségével hajtja végre (**NewShapeSpawn, CheckAndMoveLeftPosition, CheckAndMoveDownPosition, CheckAndMOveRightPostion, CheckAndRotatePosition)**, s azon kívül ellenőrzi

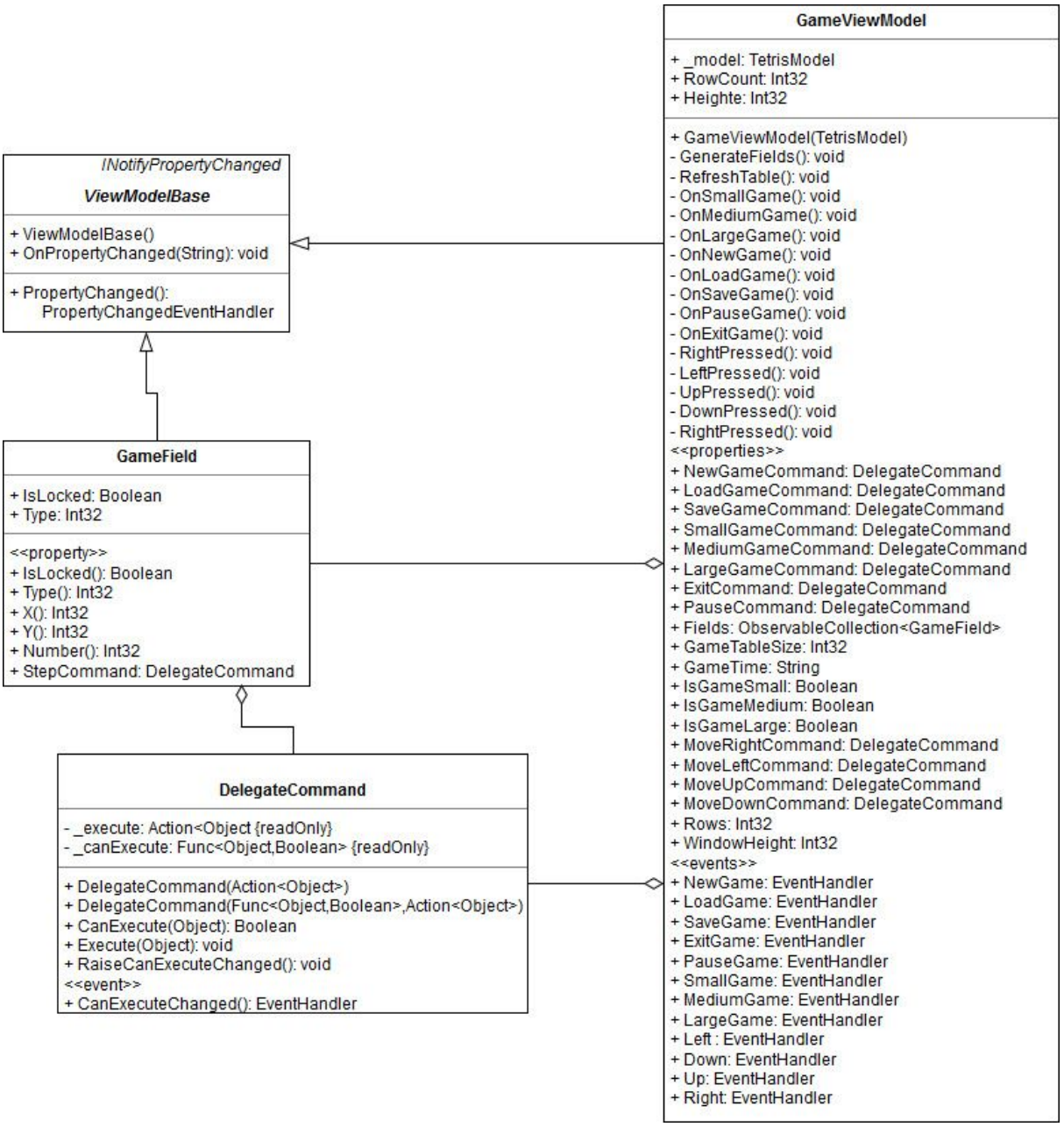
ha betelik egy sor, s elvégzi a megfelelő mozgatásokat ha ez megtörténik (**DeleteCompletedRows)**.



Az alkalmazás modellje

- Nézetmodell:
  - A nézetmodell megvalósításához felhasználtunk egy általános utasítás {**DelegateCommand}**, valamint egy ős változásjelző {**ViewModelBase}** osztályt
  - A nézetmodell feladatait a **GameViewModel** 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 bele a játék futtatásába.

- A játékmező számára egy külön mezőt biztosítunk (**GameField**), amely eltárolja a pozíciót, szöveget, engedélyezettséget. A Mezőket egy felügyelt gyűjteménybe helyezzük gyűjteménybe helyezzük a nézetmoddelbe **(Fields)**



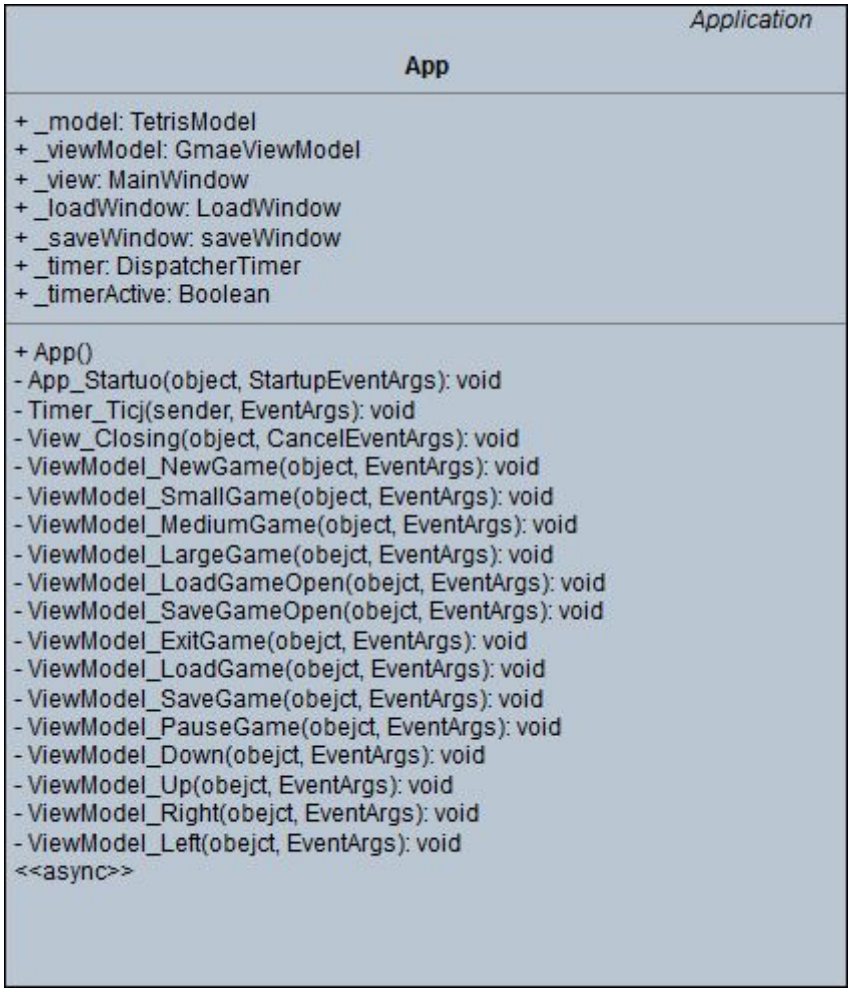
A nézetmodell osztálydiagrammja

- Nézet**:**
  - A nézet fő képernyőjét a **MainWindows** osztály tartalmazza a nézet egy rácsban tárolja a játékmezőt, a menüt és a státuszsort. A játékmező egy **ItemsControl** vezérlő, ahol dinamukusan felépítünk egy rácsot (**UniformGrid**), amely gombokból áll.

Minden adatot adatközéssel kapcsolunk a felülethez, továbbá azon keresztül szabályozzuk a gombok színét is.

- A betöltendő játékállapot bekérésért a **LoadWindow** osztály fele, amely dialogusablakként került megjelenítésre. A néze egy **ListBox** vezérlőben listázza ki az elérhető játékállapotokat.
- Új mentésének nevét a **SaveWindow** osztály által megjelenített felület kéri be. A nézeten egy szövegdobozban **(TextBox)** megadható az új mentés neve, valamint a **LoadWindow** ablakhoz hasonlóan megjeleníti a létező mentések nevét (felülírás céljából)
- A figyelmeztető és információs üzenetek megjelenését beépített dialógusablakok segítségével végezzük.

- Környezet:
  - Az **App** osztály feladata az egyes réteges példányosítása (**App_Startup),** összekötése, a nézetmodell, valamint a modell eseményeinek lekezelése, és ezáltal a játék, az adatkezelés, valamint a nézetek szabályozása.
  - A játék léptetéséhez tárol egy időzítőt is (**_timer)**, amelynek állítását is szabályozza az egyes funkciók hatása.



A vezérlés osztálydiagrammja

Tesztelés:

- A modell funkcionalitását egységtesztek segítségével lett ellenőrizve a
- Az alábbi tesztesetek kerültek megvalósításra:
    - **TetrisSmallGame, TetrisMediumGame, TetrisLargeGame:** Pályaméret beállítása, új játék indítása, időellenőrzés és játékméret ellenőrzése. Játéktábla méretének ellenőrzése.
    - **TetrisAdvanceTime**: Itt ellenőrizzük a játék idejének telését. És a játék léptetésének a hatását. Leellenőrizzük a bal, illetve a jobbra való mozgást is.
    - **TetrisLeftSideTest**: Itt leellenőrizzük a játéktábla jobb oldalát. Nem lehet kimenni az alakzattal.
    - **TetrisRightSideTest**: Itt leellenőrizzük a játéktálba jobb oldalát. Nem lehet kimenni az alakzattal, s megáll .
    - **TetrisDownMovement:** A játékbeli lefelé mozgást ellenőrzi, s a játék vége event is meghívódik nagy valószínüséggel.