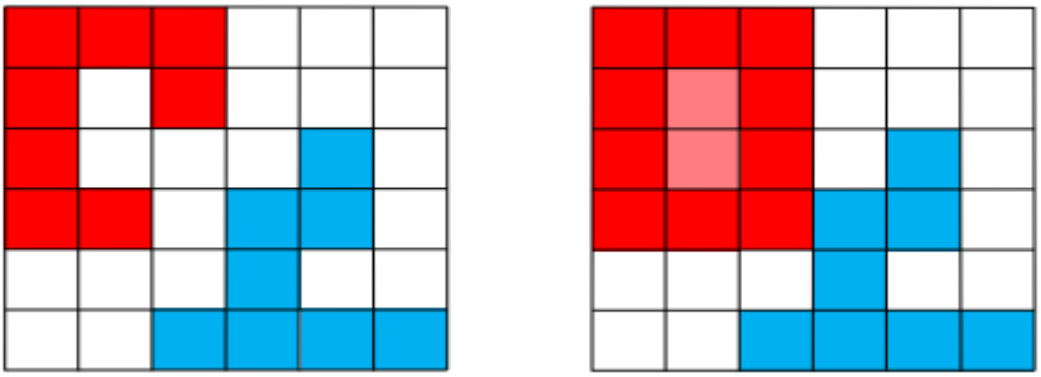
3. Beadandó feladat dokumentáció

Készítette:
Jakab Olivér
CAGJCQ
jakaboliver98@gmail.com

Feladat:

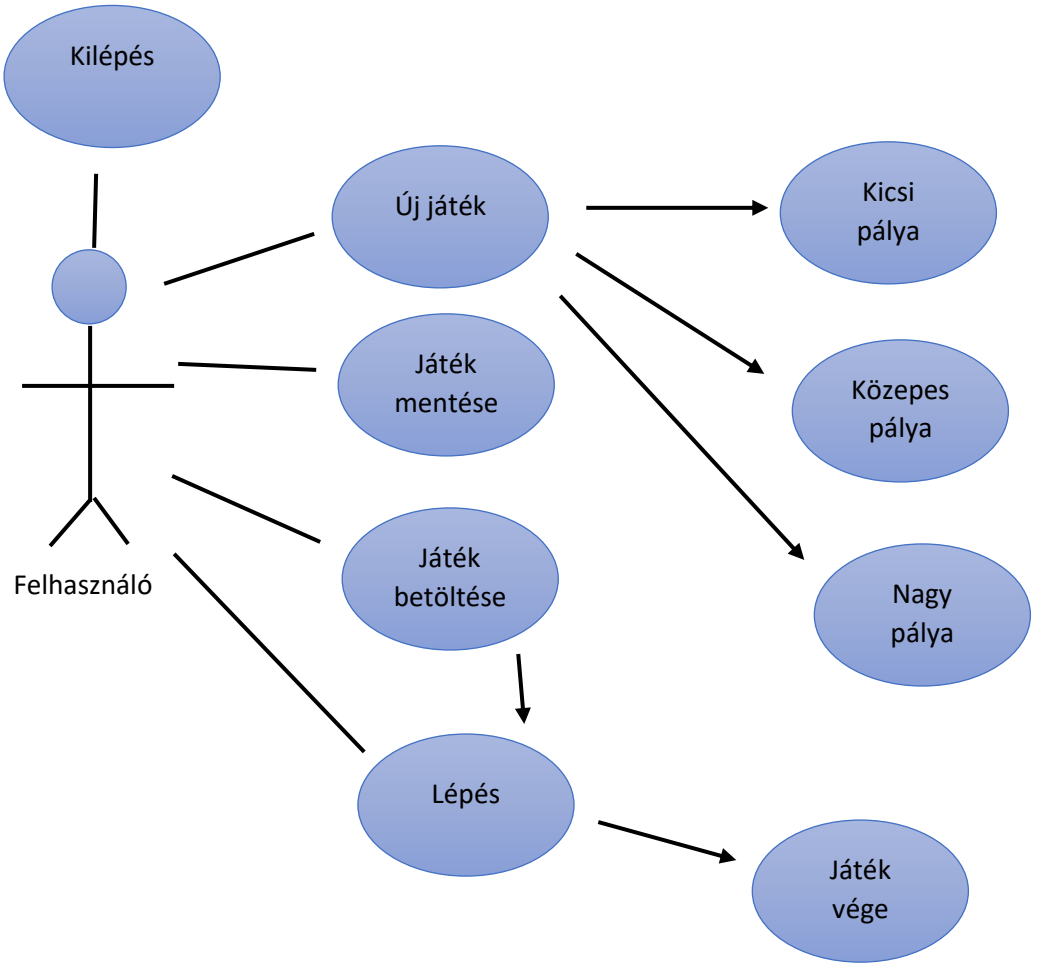
Készítsünk programot, amellyel a következő két személsyes játékot játszhatjuk. Adott egy $n \times n$ mezőből álló tábla, amelyre a játékosok 2 × 1-es méretű téglalapokat helyezhetnek el (vízszintesen, vagy függőlegesen).
A játékosok felváltva léphetnek. A játék célja, hogy a téglalapokkal elhatároljuk a tábla egy részét (teljesen körbevéve téglalapokkal), amelyben így minden mező a játékosé lesz (beleértve az ellenfél által korábban elfoglalt mezőket is). A program külön jelölje meg a lehelyezett téglalapokat, illetve az elfoglalt területeket, és játék közben folyamatosan jelenítse meg az elfoglalt terület méretét játékosonként. A program biztosítson lehetőséget új játék kezdésére a pályaméret megadásával, (6 × 6, 8 × 8, 10 × 10), valamint játék mentésére és betöltésére. Ismerje fel, ha vége a játéknak, és jelenítse meg, melyik játékos győzött.



1.ábra: minta a körbevétele (a sarkok nem számítanak)

Elemzés:

- A játékot három különböző méretű pályán játszhatjuk: 6x6, 8x8 és 10x10. A játék elindításakor alapból egy kis pályát állít be nekünk a játék.
- A feladatot egyablakos asztali alkalmazásként Windows Presentation Foundation felülettel valósítjuk meg.
- Az ablakban elhelyezünk egy menü a következő menüpontokkal:
  - File
    - Új játék
      - Kicsi pálya
      - Közepes pálya
      - Nagy pálya
    - Játék mentése
    - Játék betöltése
    - Kilépés
- A játékban téglalapokat kell lehelyezni, amely téglalap 2 pontot ér (az elfoglalt mezők száma)
- A játék véget ér ha a tábla teljesen betelt illetve, ha az egyik játékos a tábla felét vagy több mint felét birtokolja. Ilyenkor egy üzenetet jelenítetünk meg, hogy melyik játékos nyerte a játékot.
- A játék állapota adatbázisba menthető az Entity Framework ORM keretrendszer használatával. A betöltést és a mentést egyedi dialógusablakokkal végezzük, a mentések egyedi nevét a felhasználó adja meg.
- A felhasználói esetek a következő ábrán, azaz az 2.ábran láthatóak:



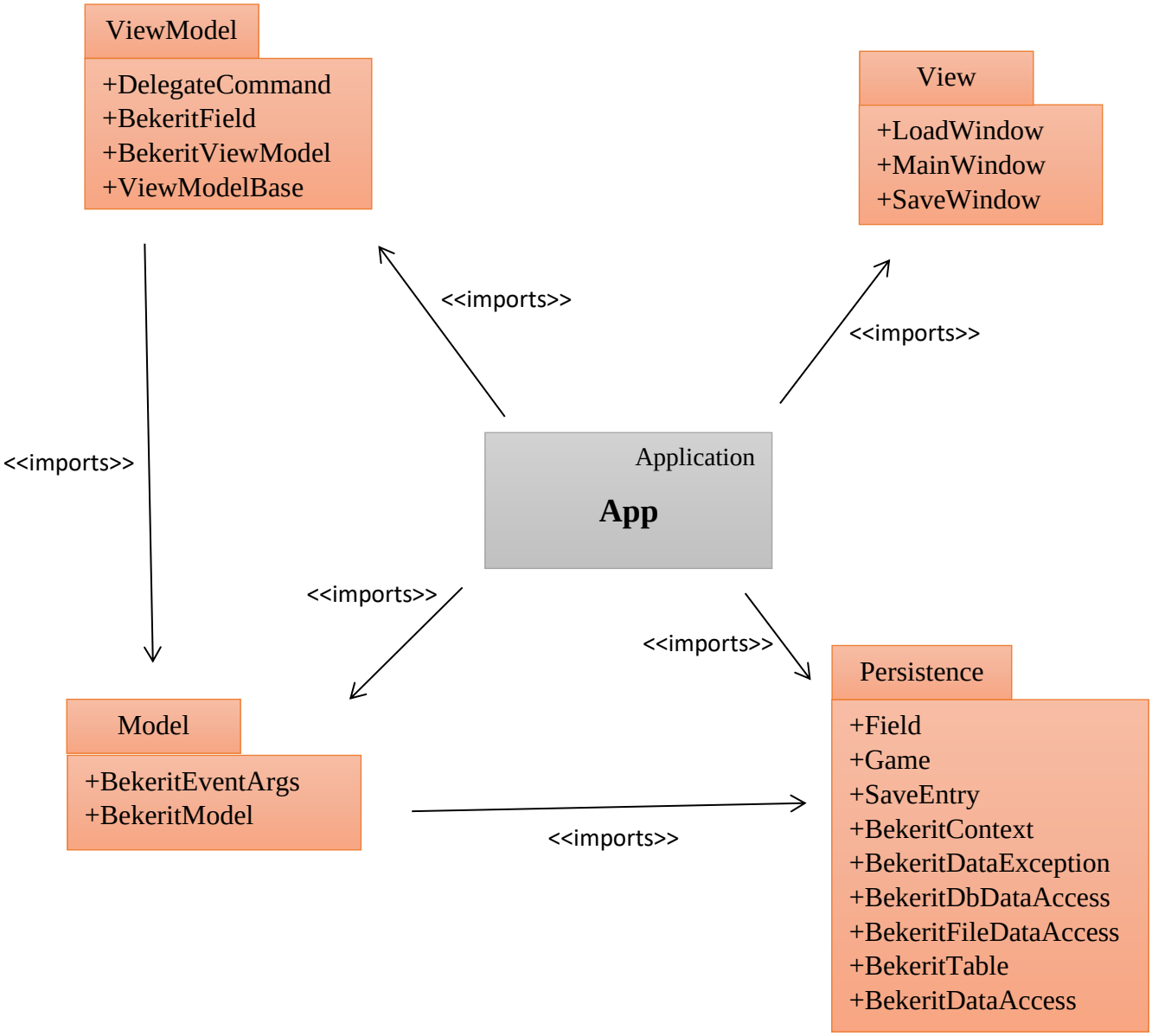
2.

2. ábra: a felhasználó esetek diagramja

Tervezés:

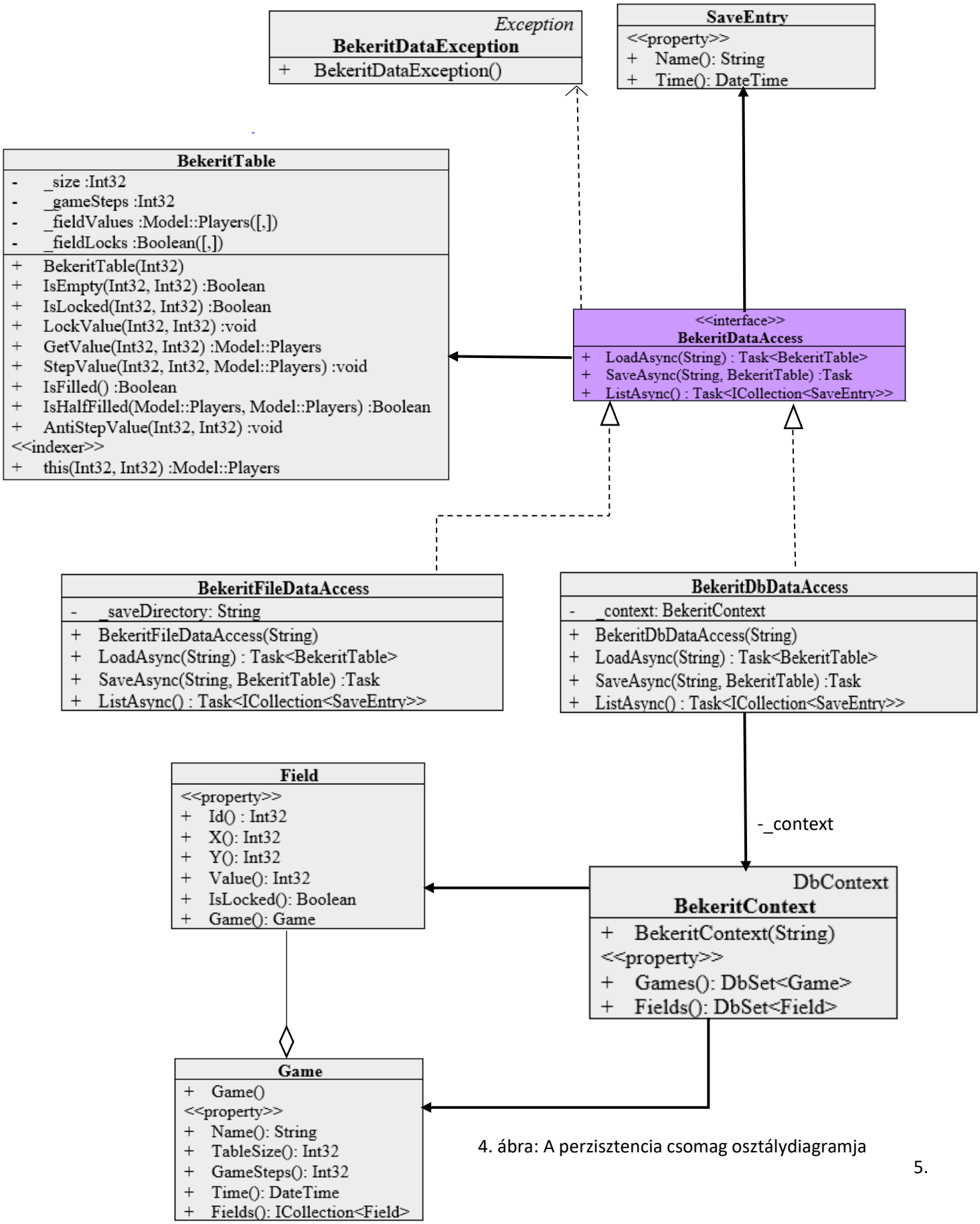
- Programszerkezet:

  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 3. ábrán látható.



3. ábra: Az alkalmazás csomagdiagramja

- Perzisztencia:
  - Az adatkezelés feladata a Bekerit táblával kapcsolatos információk tárolása, valamint mentés/betöltés biztosítása.
  - A **BekeritTable** oszály egy táblát biztosít, ahol minden mezőre ismert az értéke: **(_fieldValues)**, illetve zártsága **(_fieldLocks)**.
  - A hosszú távú adattárolás lehetőségeit a **BekeritDataAccess** interfész adja meg, amely lehetőséget ad a tábla mentésére (SaveAsync) valamint mentésére (LoadAsync). A műveletet hatékonysági okokból aszinkron módon valósítjuk meg.
  - Az interfész szöveges fájl alapú adatkezelésére a **BekeritFileDataAccess** osztály valósítja meg. A fájlkezelés során fellépő hibákat a **BekeritDataException** kivétel jelzi.
  - Az interfészt adatbázis alapú adatkezelésre a **BekeritDbDataAccess** osztály valósítja meg. Az adatbáziskezelés az Entity Framework használatával a **Field** és **Game** entitás típusokkal és a **BekeritContext** adatbázis kontextussal történik. Az adatbáziskezelés során fellépő hibákat a **BekeritDataException** kivétel jelzi.
  - A program az adatokat tehát tudja szöveges fájlként tárolni (melyek az **stl** kiterjesztést kapják) vagy adatbázisban is (az **App.config** konigurációs állományban megadott connection string által leírt módon). Ezeket az adatokat a programban bármikor be lehet tölteni, illetve ki lehet menteni az aktuális állást.



4. ábra: A perzisztencia csomag osztálydiagramja

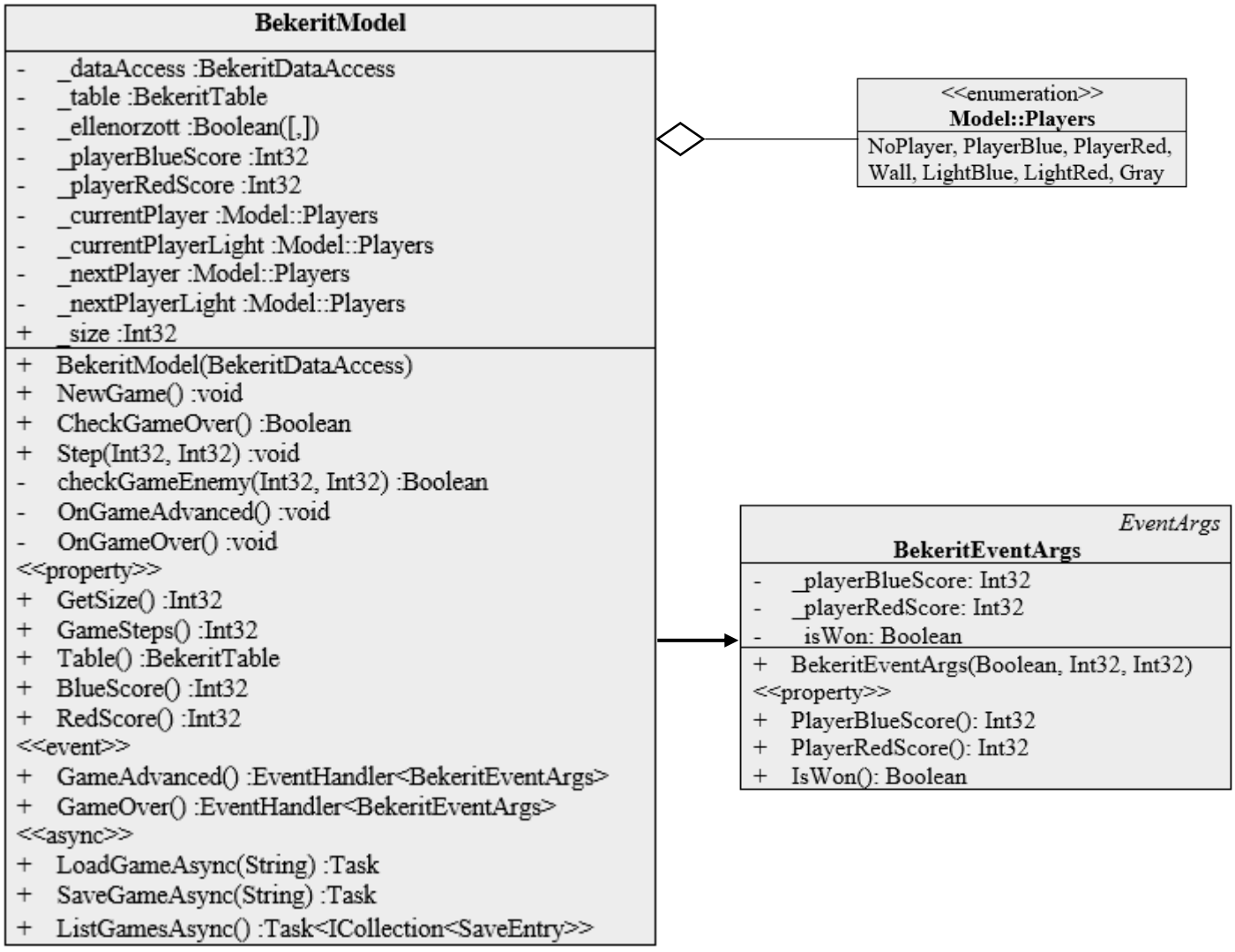
Modell:

- A modell lényegi részét a **BekeritModel** osztály valósítja meg, amely szabályozza a tábla tevékenységeit, valamit a játék egyéb paramétereit, úgy mint a méret **(_size)** a játékosok **(_currentPlayer, _nextPlayer)**. A típus lehetőséget ad új játék kezdésére **(NewGame)**, valamint a lépésre **(StepGame)**.

- A játékosok pontjainak változásáról a **GameAdvanced** esemény, míg a játék végéről a **GameOver** esemény tájékoztat. Az események argumentuma a **(BekeritEventArgs)** tárolja a győzelem állapotát, valamint a játékosok pontjait.

- A modell példányosításkor megkapja az adatkezelés felületét, amelynek segítségével lehetőséget ad betöltésre **(LoadGameAsync)** és mentésre **(SaveGameAsync)** valamint a létező mentések lekérdezésére a **(ListGamesAsync)**.

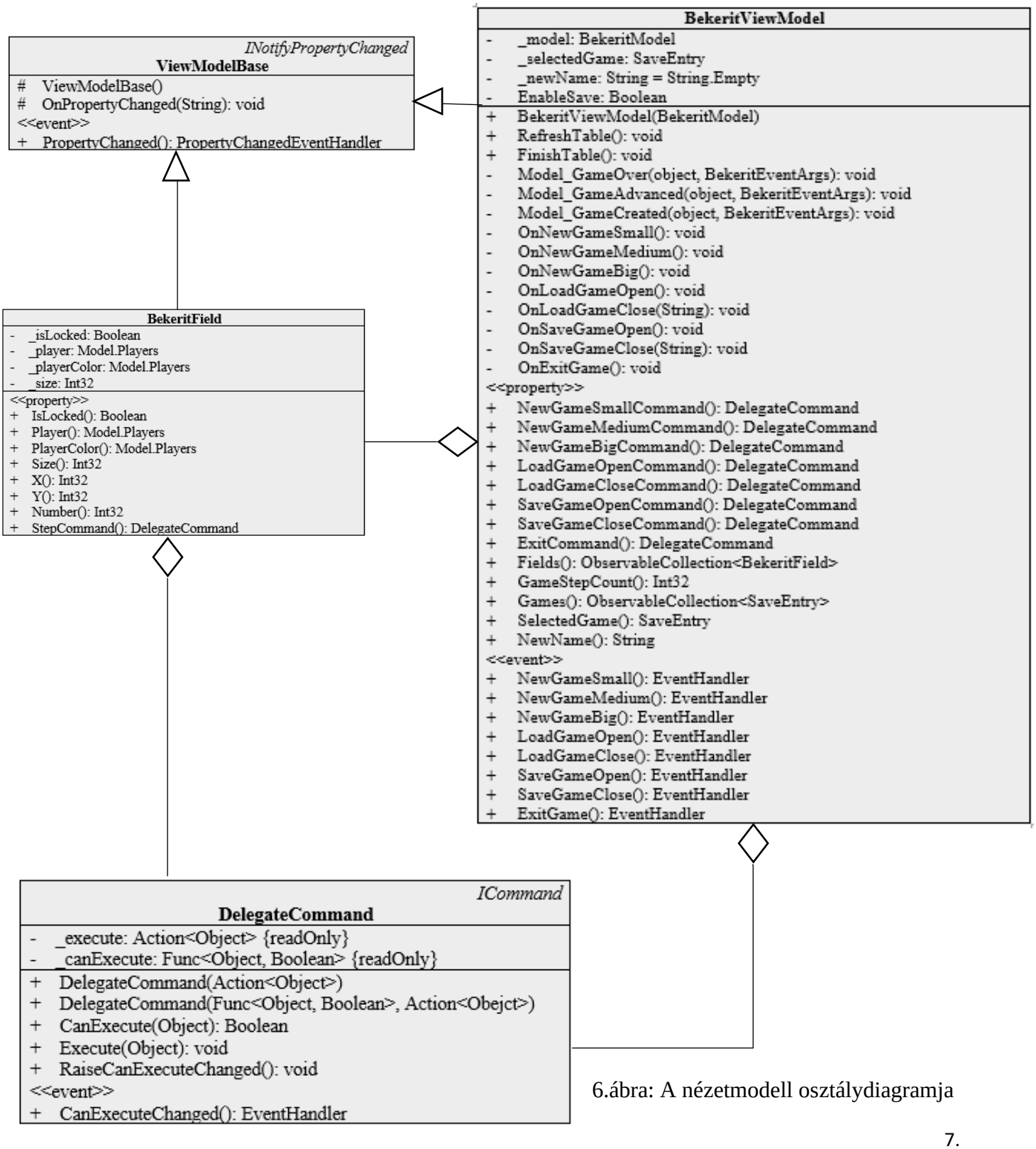
- A játékosokat a **Players** felsorolási típuson át kezeljük.



5. ábra: Model csomag osztálydiagramja

Nézetmodell (5. ábra):

- A nézetmodell megvalósításához felhasználunk egy általános utasítás (**DelegateCommand**), valamint egy ős változásjelző (**ViewModelBase**) osztályt.

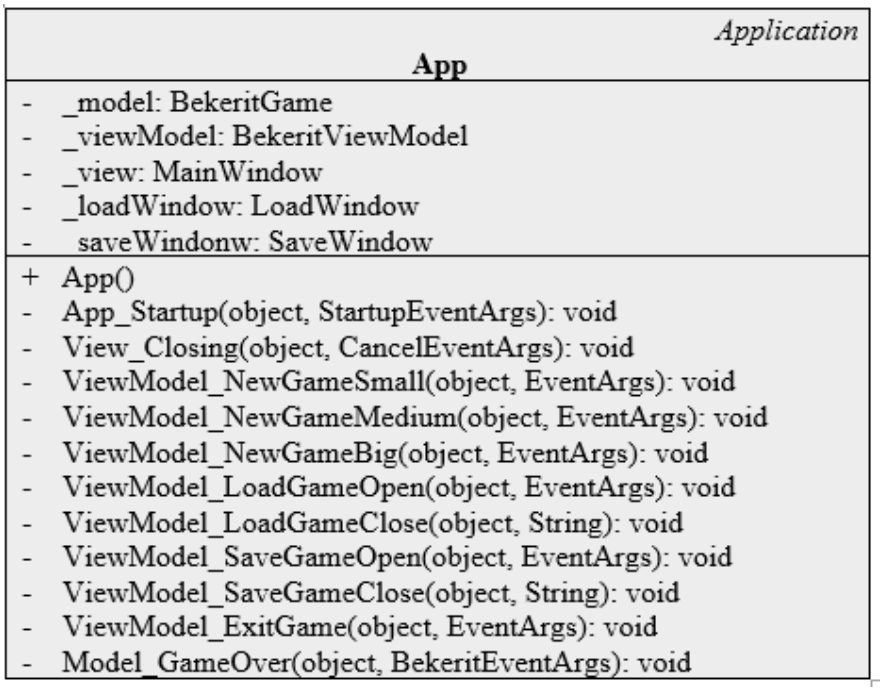


6.ábra: A nézetmodell osztálydiagramja

- A nézetmodell feladatait a **BekeritViewModel** 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 (**BekeritField**), amely eltárolja a pozíciót, szöveget, engedélyezettséget, valamint a lépés parancsát (**StepCommand**). A mezőket egy felügyelt gyűjteménybe helyezzük a nézetmodellbe (**Fields**).

- Nézet:
  - A nézet fő képernyőjét a **MainWindow** osztály tartalmazza 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 betöltendő játékállapot bekérésért a **LoadWindow** osztály felel, amely dialógusablakként került megjelenítésre. A nézet 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 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 **BekeritModelTest** osztályba.
  - Az alábbi tesztesetek kerültek megvalósításra.
  - **NewGameModelSmallMapTest**
  - **NewGameModelMediumMapTest**
  - **NewGameModelBigMapTest**: Új játék indítása mind a 3 méretű pályán. Lépések, pontok és mezők száma ellenőrzése.
  - **BekeritModelStepGameTest**: A játékos lépését ellenőrzi. Figyeli a mező helyes változását illetve a mező helyes zárolását.
  - **BekeritModelGameAdvanceTest:** A játékosok pontjainak ellenőrzése a **GameAdvance** event alapján.
  - **BekeritModelGameOverTest**: Teszteli, hogy a modell észreveszi-e, ha a játéknak vége van.