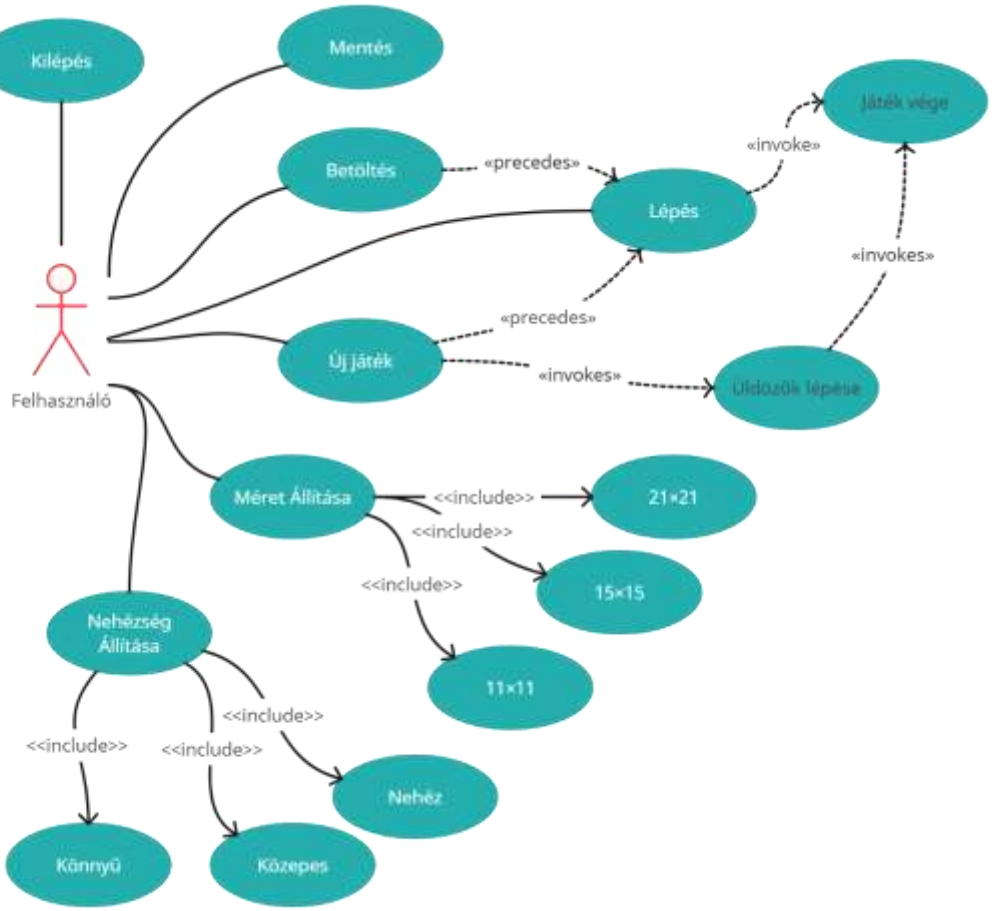
2. Beadandó feladat dokumentációja

Készítette:
Gyorgyevics Gabriella, ebh97v
gyorgyevics.gaga@gmail.com

Feladat:
Készítsünk programot, amellyel a következő játékot játszhatjuk.
Adott egy $n \times n$ elemből álló játékpálya, ahol a játékos két üldöző elől próbál menekülni, illetve próbálja őket aknára csalni.
Kezdetben a játékos játékpálya felső sorának közepén helyezkedik el, a két üldöző pedig az alsó két sarokban.
Az ellenfelek adott időközönként lépnek egy mezőt a játékos felé haladva úgy, hogy ha a függőleges távolság a nagyobb, akkor függőlegesen, ellenkező esetben vízszintesen mozognak a játékos felé.
A pályán véletlenszerű pozíciókban aknák is elhelyezkednek, amelyekbe az ellenfelek könnyen beleléphetnek, ekkor eltűnnek (az akna megmarad).
A játékos vízszintesen, illetve függőlegesen mozoghat (egyesével) a pályán, és célja, hogy az ellenfeleket aknára csalja, miközben ő nem lép aknára. Ha sikerül minden üldözőt aknára csalnia, akkor győzött, ha valamely ellenfél elkapja (egy pozíciót foglal el vele), vagy aknára lép, akkor veszített.
A program biztosítson lehetőséget új játék kezdésére a pályaméret megadásával (11×11, 15×15, 21×21), valamint játék szüneteltetésére (ekkor nem telik az idő, és nem léphet senki). Ismerje fel, ha vége a játéknak, és jelenítse meg, hogy győzött, vagy veszített-e a játékos. Ezen felül szüneteltetés alatt legyen lehetőség a játék elmentésére, valamint betöltésére. A program játék közben folyamatosan jelezze ki a játékidőt.

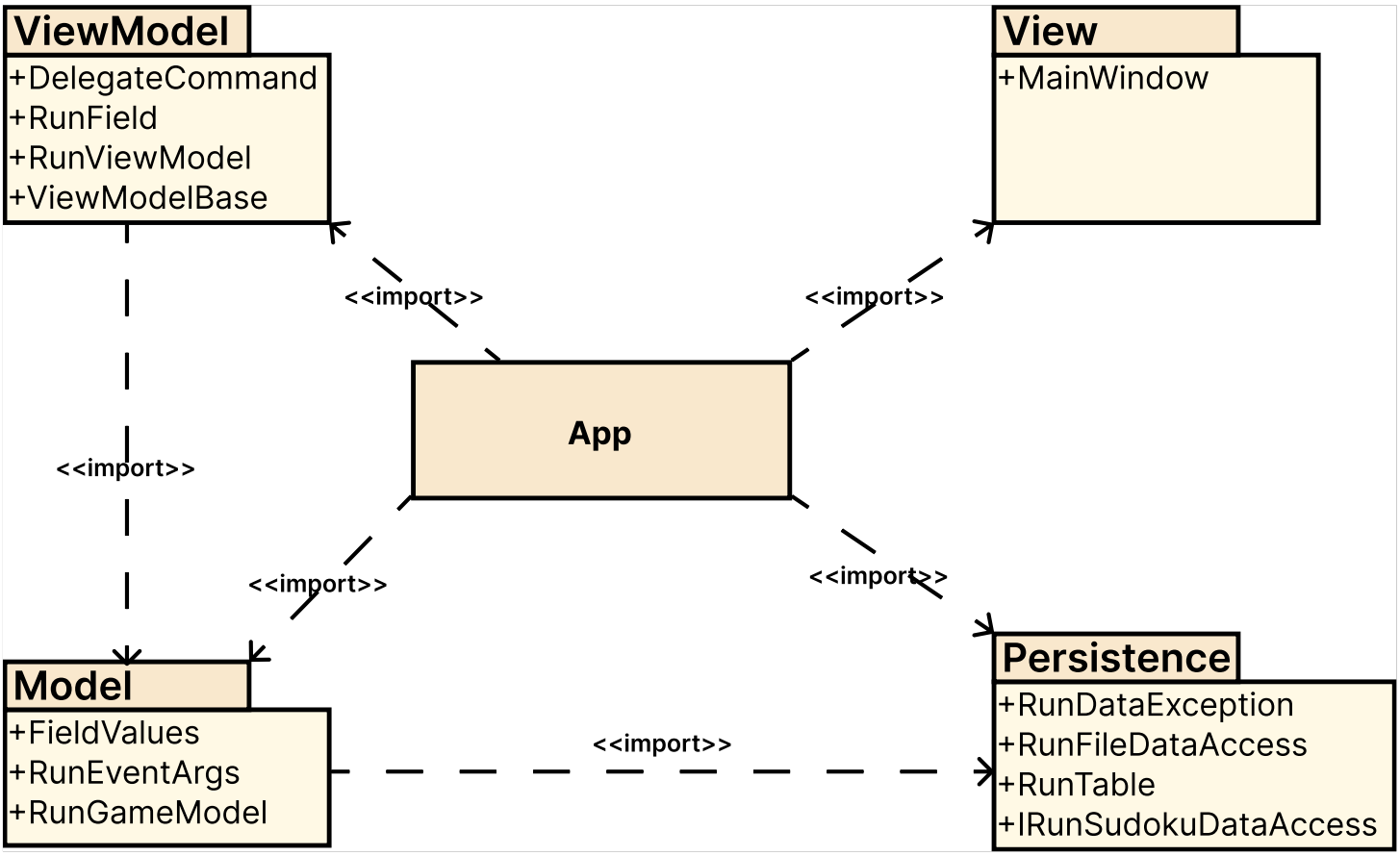
Elemzés:

- A játékot három különböző mérettel játszhatjuk: 11x11, 15x15, 21x21. Indításkor a program a közepes mérettet állítja be, és új játékot indít.
- A feladat nem határozza meg az aknák számát. Erre a *Minesweeper* játék képletét használjuk: $Aknák = Nehézség * Pályaméret * 0{,}15625$. Innen könnyen implementálható nehézségi szint is, így választhatunk Könnyű (3 másodpercenként mozgó üldözők, sok akna), Közepes (2 másodpercenként mozgó üldözők, kevesebb akna) és Nehéz (1 másodpercenként mozgó üldözők, nagyon kevés akna). A játék indításkor közepes beállításon van.
- A feladatot egyablakos asztali alkalmazásként *Windows Presentation Foundation (WPF)* grafikus felülettel valósítjuk meg.
- Az ablakban elhelyezünk egy menüt a következő menüpontokkal: File (Új játék, Játék betöltése, Játék mentése, Kilépés), Beállítások (Játékméret (11x11, 15x15, 21x21), Nehézség (Könnyű, Közepes, Nehéz)). Az ablak alján megjelenítünk egy státuszsort, amely a lépések számát, illetve az elmúlt időt jelzi.
- A játéktáblát egy nyomógombokból álló rács reprezentálja. A nyomógombok száma megegyezik a fent feltüntetett méreteknek megfelelő szorzattal. A nyomógomb színekkel jelzi, hogy akna (piros), üldöző (sárga) vagy játékos (zöld) van rajta.
    - Az aknát tartalmazó mezők mindig pirosok maradnak.
    - Az üldözőt tartalmazó aknák periodikusan mozognak (nehézségtől függő periodikussággal). Ekkor a mező, ahol eddig volt átváltozik fehérre, ahova lép pedig sárgává színeződik.
    - A játékos a W, A, S, D billentyűk lenyomásakor mozog rendre fel, balra, le és jobbra. Ennek színezése hasonló az üldöző mozgásához.
- 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.

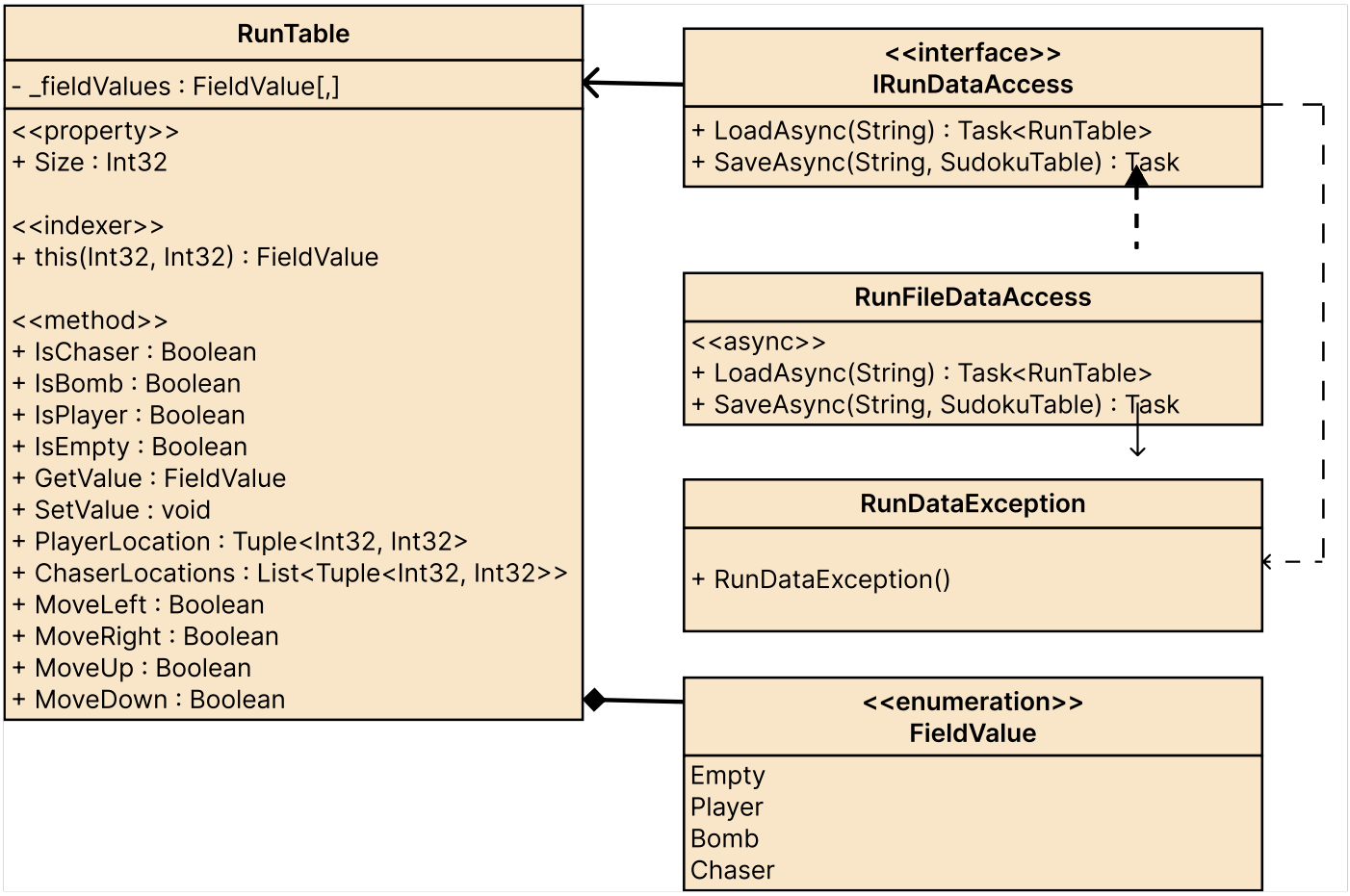


Tervezés:

- 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 programcsomagszerkezete 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-től függő projektjében kap helyet.

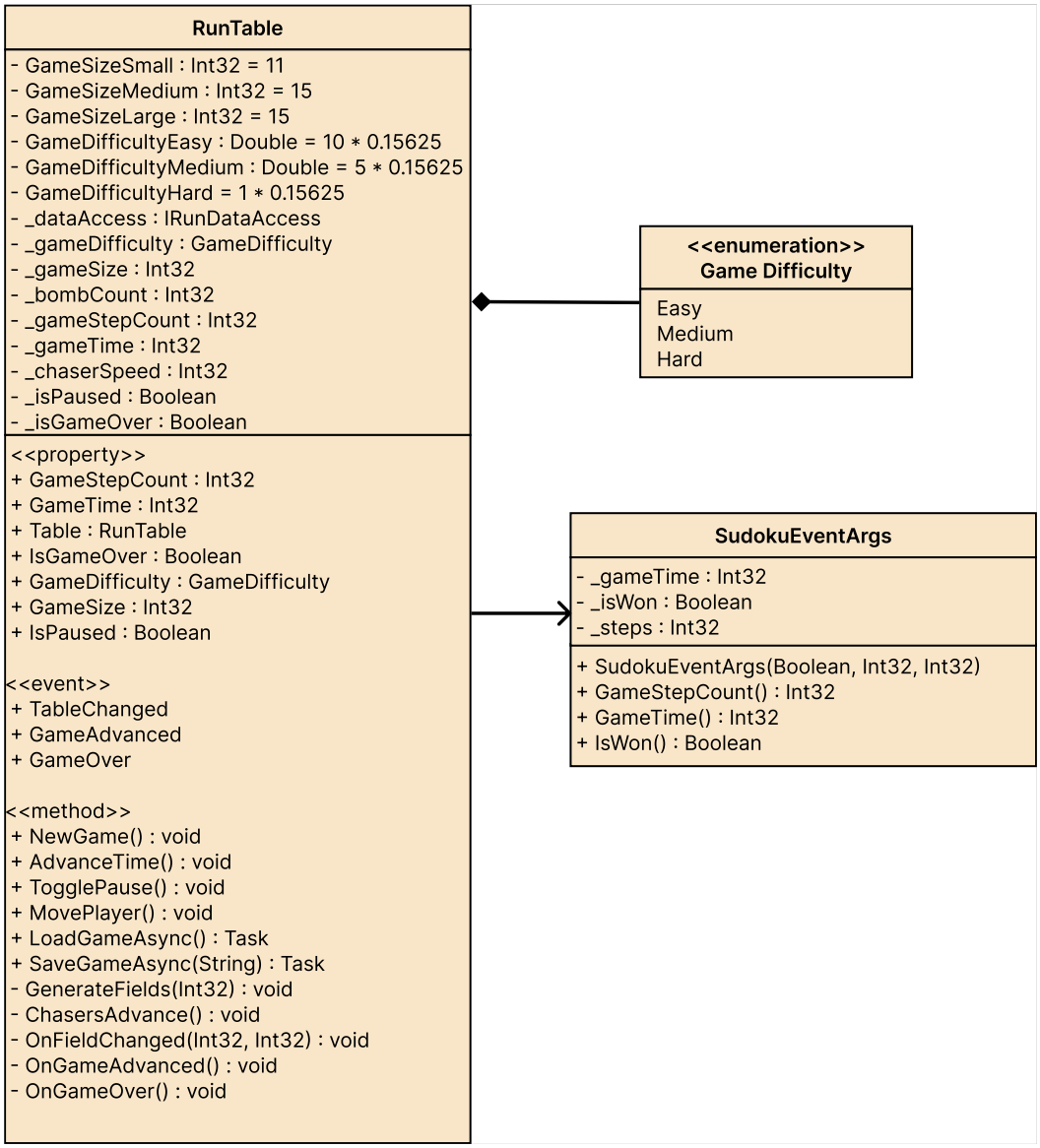


- Perzisztencia:
  - Az adatkezelés feladata a *Menekülj!* táblával kapcsolatos információk tárolása, valamint a betöltés/mentés biztosítása.
  - A *RunTable* osztály egy érvényes *Menekülj!* táblát biztosít (azaz mindig ellenőrzi a beállított értékeket), ahol minden mezőre ismert az értéke *(_fieldValues*). Utóbbit a játék kezdetekor generált, illetve értékekre alkalmazzuk. A tábla lehetőséget ad az állapotok lekérdezésére (*IsEmpty*, *IsBomb*, *IsPlayer*, *IsChaser*, *GetValue*), valamint szabályos mozgatásra (*MoveUp*, *MoveDown*, *MoveLeft*, *MoveRight*), illetve direkt beállítás (*SetValue*) elvégzésére. Ezen felül le tudjuk kérni a játékos és az üldözők pillanatnyi helyét is (*PlayerLocation*, *ChaserLocations*).
  - A hosszú távú adattárolás lehetőségeit az *IRunDataAccess* interfész adja meg, amely lehetőséget ad a tábla betöltésére (*Load*), valamint mentésére (*Save*).
  - Az interfészt szöveges fájl alapú adatkezelésre a *RunFileDataAccess* osztály valósítja meg. A fájlkezelés során fellépő hibákat a *RunDataException* kivétel jelzi.
  - A program az adatokat szöveges fájlként tudja eltárolni, melyek a *txt* kiterjesztést kapják. Ezeket az adatokat a programban szüneteltetés közben be lehet tölteni, illetve ki lehet menteni az aktuális állást.
  - A fájl első sora megadja a tábla méretét (legyen n), megtett lépések számát, elmúlt időt, és a pálya nehézségi szintjét. A fájl többi része izomorf leképezése a játéktáblának, azaz összesen n sor következik, és minden sor n karaktert tartalmaz szóközökkel választva. A karakterek megfelelnek a lehetséges értékek kezdőbetűivel:
    - *e - empty* – üres mező
    - *p - player* – a játékos pozíciója
    - *c - chaser* – üldöző pozíciója
  - b – bomb – akna

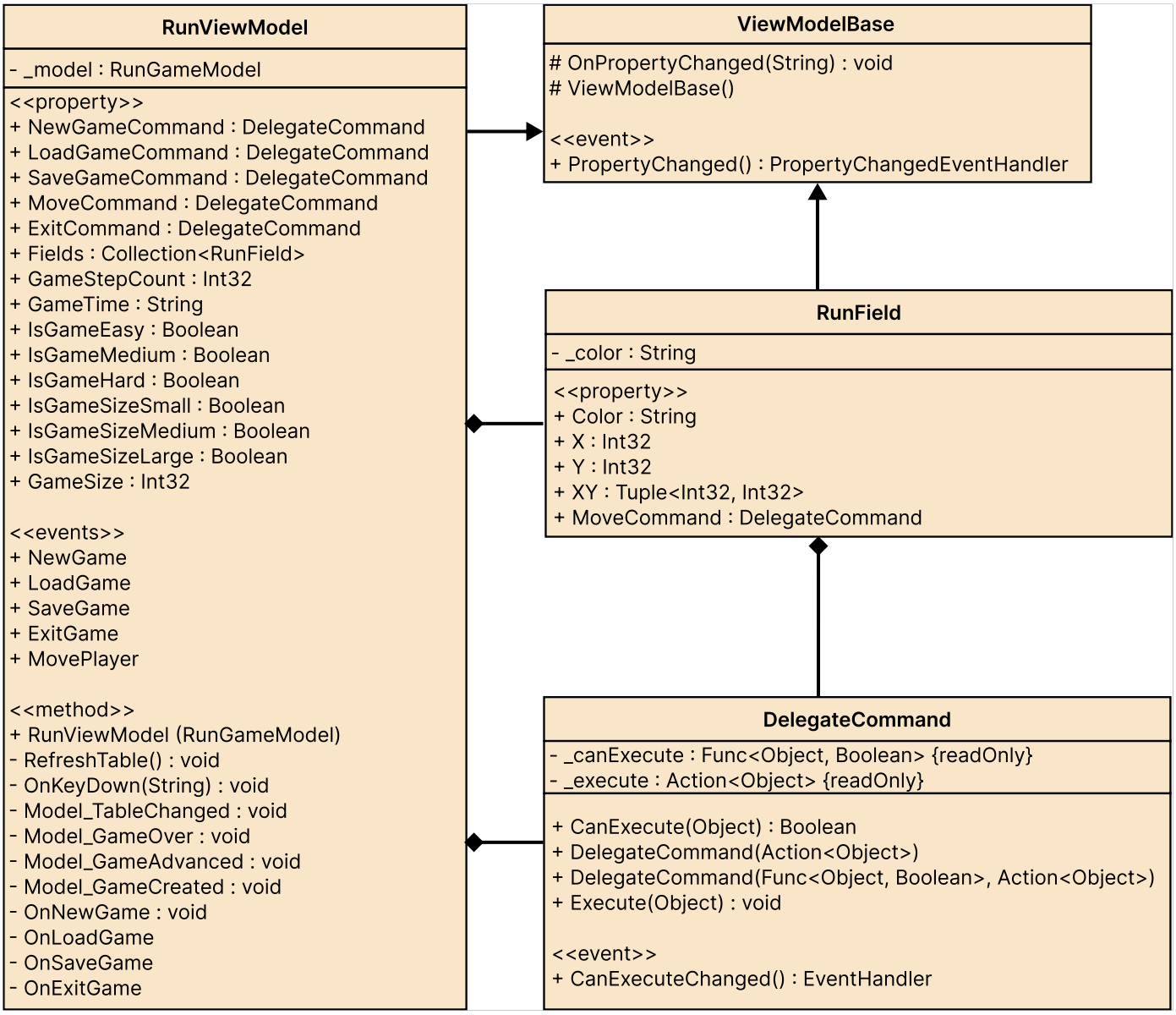


- Model:
  - A modell lényegi részét a *RunGameModel* osztály valósítja meg, amely szabályozza a tábla tevékenységeit, valamint a játék egyéb paramétereit, úgymint az
    - idő (*_gameTime*),
    - lépések (*_gameStepCount*),
    - nehézség (*_gameDifficulty*),
    - pálya mérete (*_gameSize*),
    - aknák száma (*_bombCount*) valamint
    - üldözők sebessége (*_chaserSpeed*).

    A típus lehetőséget ad új játék kezdésére (*NewGame*), valamint lépésre (*MovePlayer*, *ChasersAdvance*). Új játéknál a játékos és üldözők elhelyezése után automatikusan generálódnak az aknák. Az idő előre leptetését időbeli lépések végzésével (*AdvanceTime*) tehetjük meg.
  - A mezők állapotváltozásáról a *FieldChanged* esemény tájékoztat. Az esemény argumentuma (*RunEventArgs*) tárolja a megváltozott mező pozícióját.
  - A játékállapot megváltozásáról (lépések száma, hátra lévő idő) a *GameAdvanced* esemény, míg a játék végéről a *GameOver* esemény tájékoztat. Az események argumentuma (*RunEventArgs*) 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*)



- Nézetmodell (5. ábra):
    - A nézetmodell megvalósításához felhasználunk egy általános utasítás (*DelegateCommand*), valamint egy ős változásjelző (*ViewModelBase*) osztályt.
    - A nézetmodell feladatait a *RunViewModel* 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 (*RunField*), amely eltárolja a pozíciót, színt, valamint a lépés parancsát (*MoveCommand*). A mezőket egy felügyelt gyűjteménybe helyezzük a nézetmodellbe (*Fields*).



- Nézet:
    - A nézet csak egy képernyőt tartalmaz, a *MainWindow* osztályt. A nézet egyrácsban 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 és mennyiségét 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.
    - A játék léptetéséhez tárol egy időzítőt is (_timer), amelynek állítását is szabályozza az egyes funkciók hatására.

App
- _model : RunGameModel - _timer : DispatcherTimer - _view : MainWindow - _viewModel : SudokuViewModel
+ App() - App_Startup(object, StartupEventArgs) : void - Model_GameOver(object, SudokuEventArgs) : void - Timer_Tick(object, EventArgs) : void - ViewClosing(object, CancelEventArgs) : void - ViewModel_ExitGame(object, System.EventArgs):void - ViewModel_NewGame(object, EventArgs), void <<async>> - ViewModel_LoadGame(object, System.EventArgs):void - ViewModel_SaveGame(object, EventArgs) : void

Tesztelés:

A modell funkcionalitása egységtesztek segítségével lett ellenőrizve a SudokuGameModelTest osztályban.

Az alábbi tesztesetek kerültek megvalósításra:

- *RunGameModelNewGameSmallEasyTest*, *RunGameModelNewGameMediumMediumTest*, *RunGameModelNewGameLargeHardTest*: Új játék indítása, a mezők kitöltése, valamint a lépésszám és nehézség ellenőrzése a nehézségi fokozat függvényében.
- *RunGameModelPauseTest*: Szüneteltetés ellenőrzése: ilyenkor nem telhet az idő, a játékos pedig nem mozoghat. Amint vége a szüneteltetésnek, számoljuk a lépést.
- *RunGameModelMovementTest*: A játékos lépéseinek ellenőrzése: helyesen lép-e, nem megy-e ki a tábláról.
- *RunGameOverTest:* A játék végének kiváltása, ennek helyességének ellenőrzése.
- *RunGameModelLoadTest:* A játék modell betöltésének tesztelése mockolt perzisztencia réteggel.