Eseményvezérelt Alkalmazások

3. Beadandó feladat Dokumentáció

Készítette:

Péter Bálint

Neptun kód: gs2xc6

E-mail: gs2xc6@inf.elte.hu

Feladat:

Készítsünk programot, amellyel a következő két személyes játékot játszhatjuk. Adott egy $n \times n$ elemből álló játékpálya, ahol két harcos robotmalac helyezkedik el, kezdetben a két ellentétes oldalon, a középvonaltól eggyel jobbra, és mindkettő előre néz. A malacok lézerágyúval és egy támadóököllel vannak felszerelve. A játék körökből áll, minden körben a játékosok egy programot futtathatnak a malacokon, amely öt utasításból állhat (csak ennyi fér a malac memóriájába). A két játékos először leírja a programot (úgy, hogy azt a másik játékos ne lássa), majd egyszerre futtatják le őket, azaz a robotok szimultán teszik meg a programjuk által előírt 5 lépést. A program az alábbi utasításokat tartalmazhatja:

• előre, hátra, balra, jobbra: egy mezőnyi lépés a megadott irányba, közben a robot iránya nem változik.

• fordulás balra, jobbra: a robot nem vált mezőt, de a megadott irányba fordul.

• tűz: támadás előre a lézerágyúval.

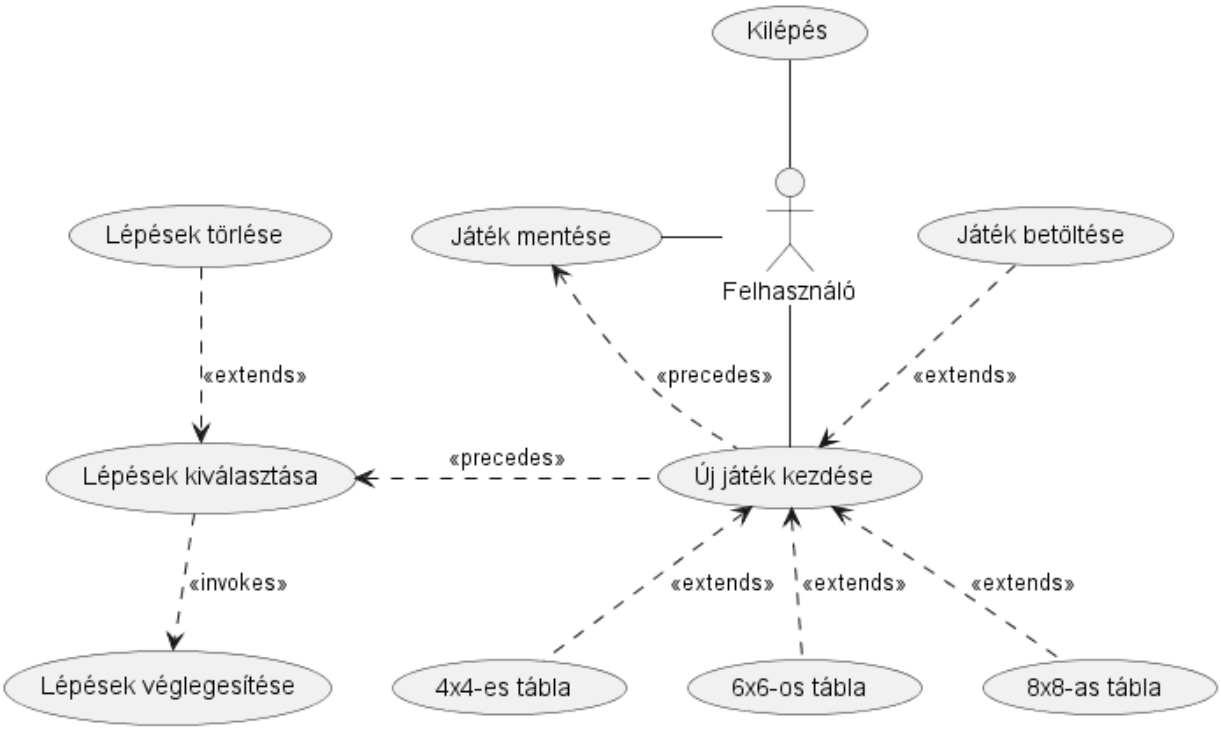
• ütés: támadás a támadóököllel.

Amennyiben a robot olyan mezőre akar lépni, ahol a másik robot helyezkedik, akkor nem léphet (átugorja az utasítást), amennyiben a két robot ugyanoda akar lépni, akkor egyikük se lép (mindkettő átugorja az utasítást). A két malac a lézerrel és az ököllel támadhatja egymást. A lézer előre lő, és függetlenül a távolságtól eltalálja a másikat. Az ütés pedig valamennyi szomszédos mezőn (azaz egy 3 × 3-as négyzetben) eltalálja a másikat. A csatának akkor van vége, ha egy robotot háromszor eltaláltak.

A program biztosítson lehetőséget új játék kezdésére a pályaméret megadásával (4 × 4, 6 × 6, 8 × 8), valamint játék mentésére és betöltésére. Ismerje fel, ha vége a játéknak, és jelenítse meg, melyik játékos győzött. Játék közben folyamatosan jelenítse meg a játékosok aktuális sérülésszámait.

Elemzés:

- A feladatot multiplatformos .NET Avalonia felülettel valósítjuk meg, asztali valamint Android alkalmazásként
- Indításkor alapértelmezetten egy 4x4-es táblaméretű játék kezdőállapota fogad, új játék a fenti menüsorból indítható 3 mérettel (4x4,6x6,8x8), vagy betölthető fájlból (.rf kiterjesztés)
- A játéktábla bal középen jelenik meg nxn-es képekből álló rácsként (játékos karakter képe van egy mezőn vagy üres), alsó sarkokban a játékosok életereje. A rendelkezésre álló akciók közüli választást, valamint az utasítások törlését és kör befejezését a játéktábla melletti gombok teszik lehetővé. A tábla alatt szerepel a jelenlegi játékos száma a hátrelévő akcióoik számával együtt.
- Elmenteni a játék állapotát vagy újra betölteni egyet szintén a fenti menüsoron keresztül lehet
- A játék kétszemélyes, miután két darab, öt utasítás hosszú sorozat sikeresen be lett táplálva szimultán (mindkét játékos első akciója, aztán mindkettőjük másodikja, és így tovább) lejátszódnak, ha a végére mindkét játékos életereje nagyobb mint nulla, a játék folytatódik az első játékos körével
- A játék kétféleképpen érhet véget: mindkét játékos életereje elfogy, ez esetben a játék döntetlent hirdet ki; vagy az egyik játékos életben marad míg a másik nem, ebben az esetben természetesen a megfelelő játékos nyert. Mindkét esetben dialógusablak jelenik meg az eredményről, közben új játék indul a jelenlegi táblamérettel

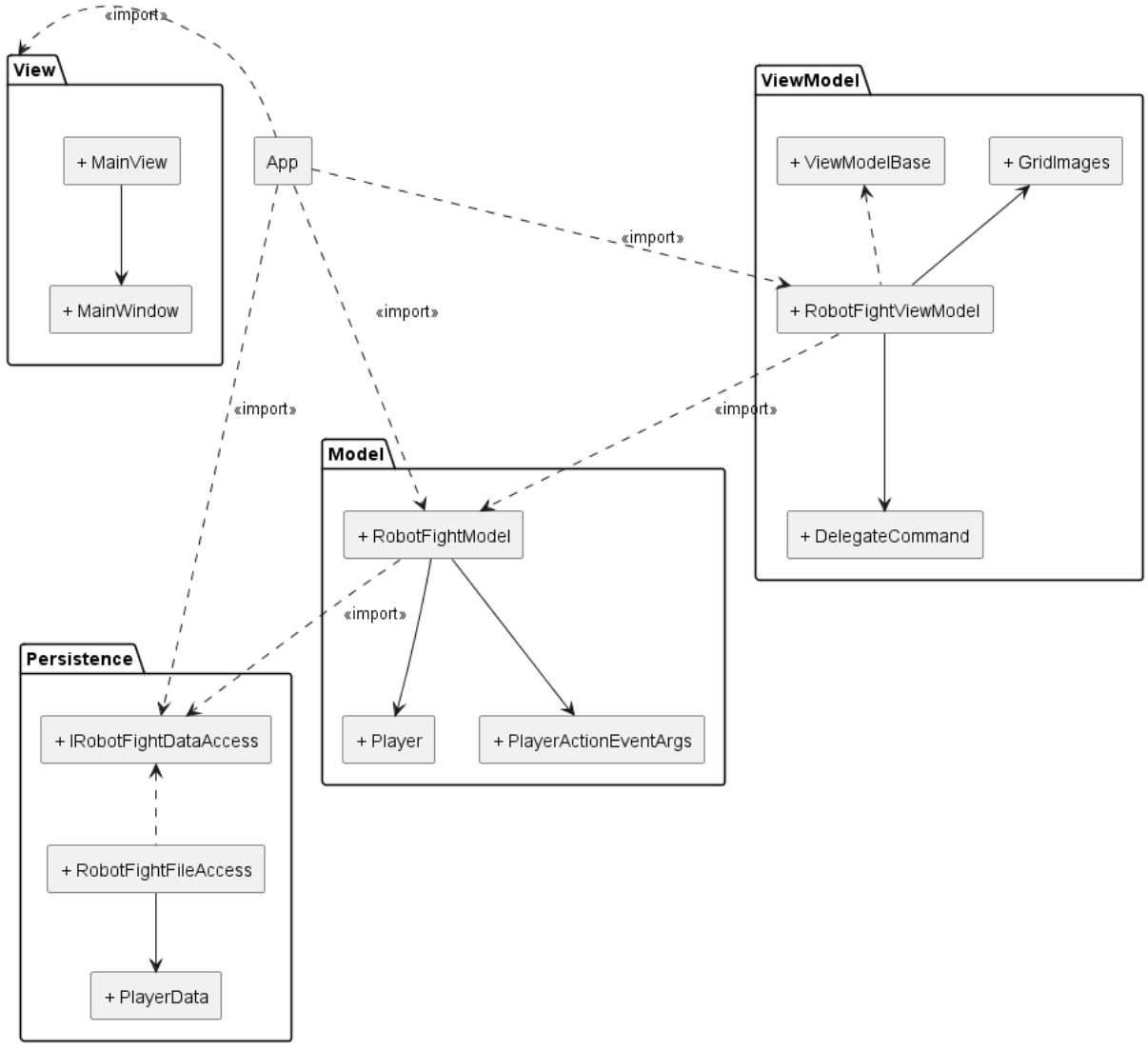


Felhasználói eset diagram

Tervezés:

Programszerkezet

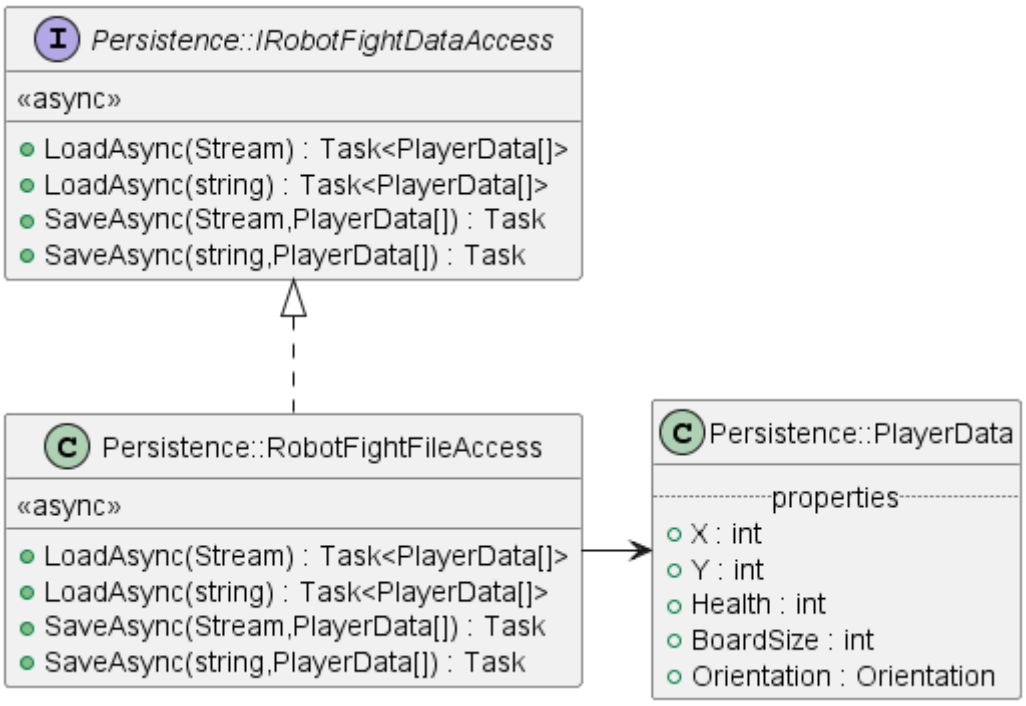
- A programot MVVM architektúrában valósítjuk meg, ennek megfelelően **View**, **Model**, **ViewModel** és **Persistence** névtereket valósítunk meg az alkalmazáson belül. A program környezetét az alkalmazás osztály (**App**) végzi, amely példányosítja a modellt, a nézetmodell és a nézetet, biztosítja a kommunikációt a felhasználó felé, valamint felügyeli az adatkezelést (mentés és betöltés)
- A **Model** és **Persistence** réteg egy osztály könyvtár projektben szerepelnek (**RobotFight**), míg a **View** és **ViewModel** egy **Avalonia** projektben (**RobotFight_Avalonia**), amihez tartozik asztali (**RobotFight_Avalonia.Desktop**) és Androidos (**RobotFight_Avalonia.Android**) alkalmazás projekt



Perzisztencia

- A perzisztencia feladata a játékállapot hosszú távú elmentésének valamint elmentett játékállapot betöltésének biztosítása, a felhasználó által megadott módon vagy automatikusan kilépéskor/belépéskor
- A hosszú távú adattárolás lehetőségeit az **IRobotFightDataAccess** interfész adja meg, amely lehetőséget ad a tábla betöltésére (**LoadAsync**), valamint mentésére (**SaveAsync**), aszinkron módon mind elérési útvonal (**Path**), mind adatfolyam (**Stream**) által
- **PlayerData** osztály: tartalmazza egy játékosra vonatkozóan az x és y koordinátáit a táblán (**int,int**), életerő pontjainak számát (**int**) , pályájának méretét (**int**) valamint irányát (**RobotFight.Model.Orientation**)
- Az említett interfészt megvalósítja a **RobotFightFileAccess** osztály szöveges fájlba való mentésre és betöltésre. A szöveges fájlok .rf kiterjesztéssel rendelkeznek, és a **PlayerData** osztály mezőihez tartozó értékeket tartalmazzák, a következő módon:
  - 1.sor:”x y életerő pályaméret”,
  - 2.sor:”orientáció”

  két példányban



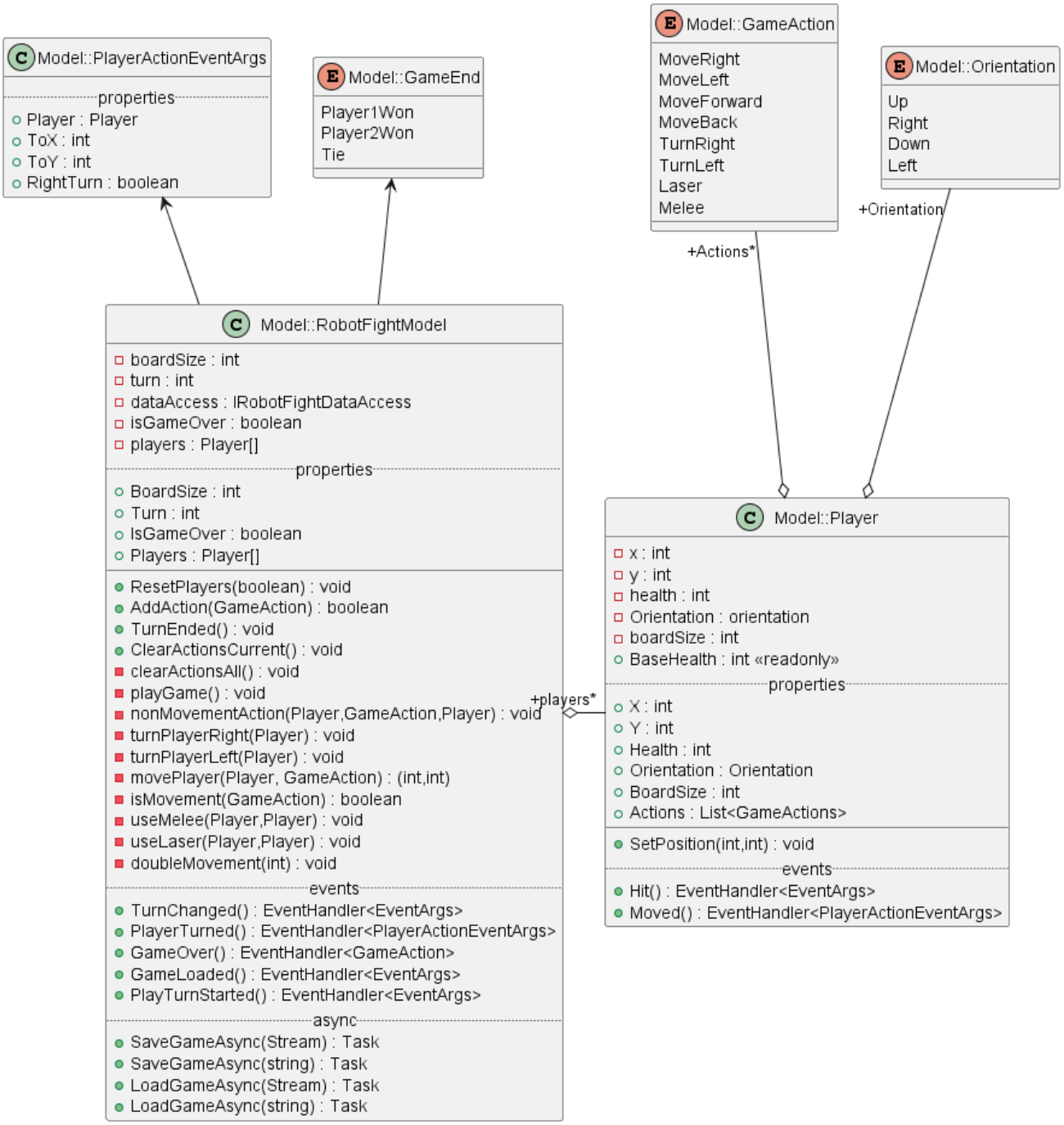
Perzisztencia osztálydiagramja

Modell:

- Tartalmazza a:
  - **RobotFightModel** osztály
  - **Player** osztály
  - **PlayerEventArgs** osztály
  - **GameAction** enum
  - **Orientation** enum
  - **GameEnd** enum

  tagokat (enum-ok nem saját fájlban)

- A modell megvalósításának fő komponense a **RobotFightModel** osztály, ami példányosít két játékost (**Player**), számon tartja a játéktábla méretét (**boardSize**), a játék állapotát (**turn**, **isGameOver**), változásakor kommunikál a nézet felé (**TurnChanged, PlayerTurnStarted**)
- A játék egy körének (különböző egy játékosnak a körétől) lejátszását a **playGame** metódus végzi, ez hívja meg a megfelelő metódusokat a játékosok akcióitól (**Player.Actions**) függően; ha mindkettő mozgás (**isMovement** segítségével) valamilyen irányba akkor doubleMovement-et, egyébként először a mozgást, aztán a másikat (**nonMovementAction**), vagy játékosok sorrendjében ha egyiksem mozgás, így tudja fenntartani a mozgások szimultánságat a mozgási szabályokra ügyelve
- **movePlayer** metódussal nyeri ki a célkoordinátákat, majd az előbb említett metódusok validálják a helyességét
- **AddAction**, **clearActionsAll** és **ClearActionsCurrent** segítségével menedzseli a játékosok beérkező akció inputjait
- **ResetPlayers** állítja a játékosokat alaphelyzetbe új játék indításakor, vagy más paraméterrel egy játék betöltésekor
- El lehet menteni a játék helyzetét (ami tulajdonképpen a játékosok bizonyos adattagjaiból áll) aszinkron módon a **SaveGameAsync** függvénnyel, betölteni egy másik játékállapotot **LoadGameAsync**-el (amihez a **GameLoaded** esemény is tartozik)
- Játéknak egy féleképpen lehet vége (leszámítva a programból való kilépést, de az nem a modellre tartozik): egy vagy mindkét játékos életereje nullára csökken, ezt a **GameOver** eseménnyel kommunikálja a nézetmodell és a környezet felé, felparaméterezve a nyertessel



Modell osztálydiagramja

Nézetmodell:

- A **RobotFightViewModel** egy **ViewModelBase**-ből származtatott osztály, így megvalósítja az **INotifyPropertyChanged** interfészt
- Tartalmazza a pálya reprezentációját **ObservableCollection<GridImage>**-ben (**GridImages**), a UI egyéb változói részeihez továbbítja a modell megfelelő adatait (pl. **CurrentActionsText**)
- Fogadja a nézet (**DelegateCommand**-ból leszármazott, tehát **ICommand** interfészt megvalósító) **Command**-jait, és feldolgozza azokat

(**NewGameCommand, AddActionCommand**, stb.), vagy továbbítja eseményekkel az App-nak (pl. **SaveCommand** és **OnSaveEvent**)

- Aggregálja a modellt (**_model**), felirakozik eseményeire, játékosainak eseményeire, összeköti a felülettel
- A **GridImage** osztály tárolja egy rácsnégyzet tartalmának repezentációját (**IsRobot1, IsRobot2**), és elforgatásának értékét (**Rotation**)

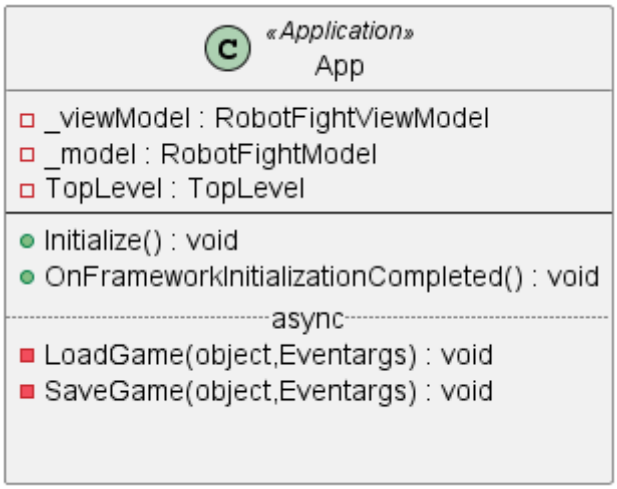
Nézetmodell osztálydiagrammja

Nézet:

- A nézet egy képernyőt tartalmaz, a **MainView** osztályt
- A **MainWindow** ablakba egyszerűen a MainView vezérlőt ágyazzuk be. Ilyen módon a felület asztali alkalmazásokban ablakos alkalmazásként, mobil platformon pedig lapként is megjeleníthető
- Egy rácsban tárolja a játékmezőt, a menüt és a státuszsort. A játékmező egy ItemsControl vezérlő, ahol dinamikusan felépítünk egy rácsot (**UniformGrid,** méretét a **BoardSize** határozza meg) , amely képekből áll, alatta a vezérlő gombok szintén **UniformGrid**. Minden adatot adatkötéssel kapcsolunk a felülethez, pl. képek forrását (**GridImages**), státuszsoron szereplő szövegeket. Státuszsoron jelennek meg a játékosok életerőpontjai, felette a kör helyzete az eddig megadott akciókkal
- Képek forrását a **GridImages** tulajdonságaira kötött **Style Class**-okkal határozzuk meg, elforgatását a **Rotation** tulajdonságával

Környezet:

- Az App osztály feladata az egyes rétegek példányosítása, összekötése, a NézetModell eseményeinek lekezelése (pl. **OnLoadEvent, OnSaveEvent**)
- Ezt az OnFrameworkInitializationCompleted metódus feluldefiniálásában végezzük el, ahogy a platformspecifikus fájlkezelést és a Nézet létrehozását is
- A fájlnév bekérését betöltéskor és mentéskor **StorageProvider** osztály segítségével végezzük
- Felugró üzenetek megjelenítéséhez a **MessageBox.Avalonia** NuGet csomagot használjuk
- Egyes Eseménykezelők (pl. **GameOver** eseménye) lambdafüggvényként lettek megvalósítva



Környezet osztálydiagrammja

Tesztelés:

A modellre vonatkozó egységtesztek a RobotFightModelTest osztályban lettek elhelyezve, Moq segítségével szimulált adateléréssel

A tesztesetek:

- **TestLoadGameAsync**: mock-olt betöltés után a modell **Player** adattagjaiban tárolt információk helyességének tesztje
- **TestResetPlayers**: helyes értékeket vesznek fel a re-inicializált **Player**-ek
- TestTurnChange: megadható-e helytelen érték a turn mezőnek (**Turn** propertyn keresztül), helyes értékadás kiváltja-e a megfelelő eseményt (**TurnChanged**)
- **TestSetPosition**: **Player** osztály **SetPosition** metódusának tesztelése a táblához képest helyes és helytelen értékekkel, esemény (**PlayerMoved**) megfelelő helyen történő kiváltása megtörténik-e
- **TestAddAction**: **AddAction** metódus tesztelése alaphelyzetben, és mikor már tartalmaz az aktuális **Player Action** listája 5 tagot
- **TestMoveToEachOther**: mozgás edgecase-ének a vizsgálata, amit a **SetPosition** önmagában nem ellenőriz, mert a táblán helyes: a játékosok egymásra akarnak lépni
- **TestTurnPlayers**: **turnPlayerLeft**, **turnPlayerRight** és eseményük (**PlayerTurned**) tesztelése
- **TestAttacks**: **useLaser** és **useMelee** tesztelése, a játékosok életerejével együtt
- **TestGameOver1Won**: **GameOver** esemény tesztelése az egyik játékos megölésén keresztül, esemény paraméterének ellenőrzése
- **TestGameOverTie**: **GameOver** esemény tesztelése játékos karakterek egyszerre történő halálakor