**3. Beadandó feladat dokumentáció**

**Feladat:**

Készítsünk programot, amellyel a következő két személyes játékot játszhatjuk.

Adott egy n × 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 életerő számukat.

**Elemzés:**

* A feladatot három különböző ablakból álló .NET MAUI alkamazásként valósítjuk meg. Főként a Windows és Android platformokon valósítjuk meg. Az alkalmazás támogatja a portré tájolást.
* Az első ablakban elhelyezünk:
  + Egy menüt az ablak tetején a következő menüpontokkal: File (Új játék, Játék betöltése, Játék mentése, Kilépés).
  + Az ablak közepén találhatjuk, a választott nagyságnak megfelelő méretű játékasztalt.
  + Amikor elindítjuk a programot, a bal felső sarokban tudunk új játékot indítani, vagy egy korábbi expliciten lementett állapotot betölteni.
  + Egy új játék kezdésekor létrehozunk egy új játékasztalt, egy n x n-es Label-ekből álló rácsot. A malacok helyzetét és irányát a megfelelő Label-en szöveg segítségével reprezentáljuk. A két játékost, a malacok színei segítségével lehet megkülönböztetni.
  + Amikor a malacok valamilyen támadást hajtanak végre, a táblán az érintett mezők a malac színével felvilágítanak. Így egyértelműen látszik a felhasználóknak, hogy ki, mikor és miért sebződött.
  + Az ablak alján található egy a felhasználók által használt panel:
  + Felül található az egy-egy játékos hátralévő életereje.
  + Alatta található egy felirat, amely jelzi, hogy melyik játékostól várjuk, hogy a parancsait beüsse. Alatta található 5 darab menüpont ahol a parancsokat kilehet választani. amelyen ezt megteheti.
  + Alatta található „Rögzítés” feliratú gombbal tudják véglegesíteni a döntéseiket.
  + Ez a szekció csak akkor aktív, ha épp bemenetet várunk.
  + Alatta található egy „Kezdés” feliratú gomb. Ezzel a gombbal lehet elindítani a kört miután mindkét játékos érvényes parancsokat adott meg.
  + Minden egyes lépéskor fontos fenntartani a reprezentáció helyességét. Ebben az esetben arra kell figyelni, hogy a malacok ne tudják elhagyni a játékteret.
* A játék önműködően feldob egy dialógusablakot, amikor vége a játéknak, hogy melyik játékos nyert.
* Két különböző ablakkal kezeljük a mentést, illetve betöltést, a fájlneveket a felhasználó adja meg. A választott állomány típus: „.dat”

**Felhasználói esetek:**

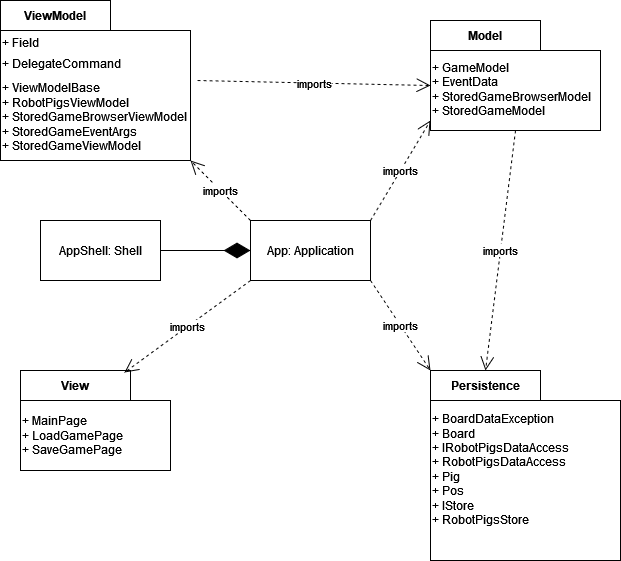
****

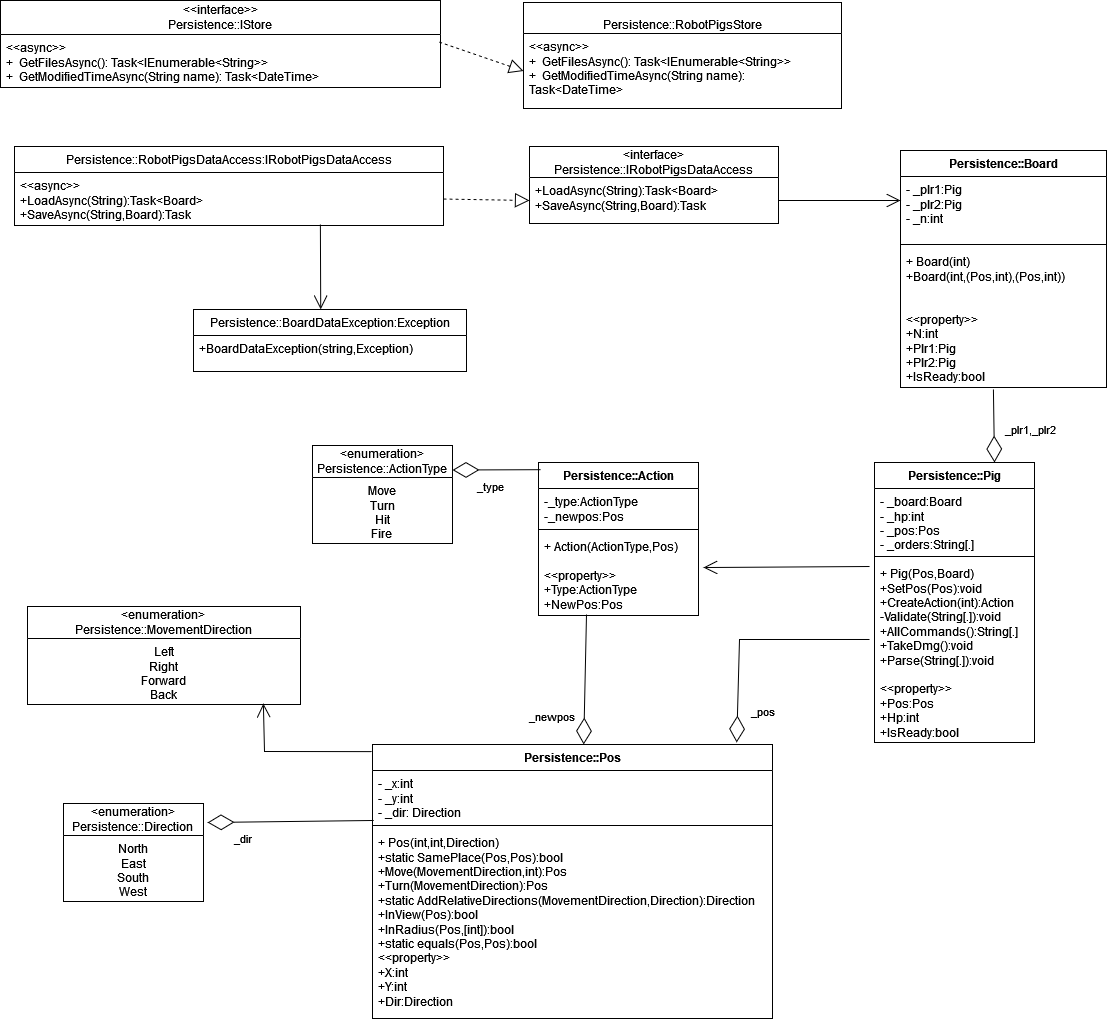
(userdiag.png)

Először az egyes, majd a kettes játékos rögzítheti a parancsait, a kezdés csak ezután válik elérhetővé.

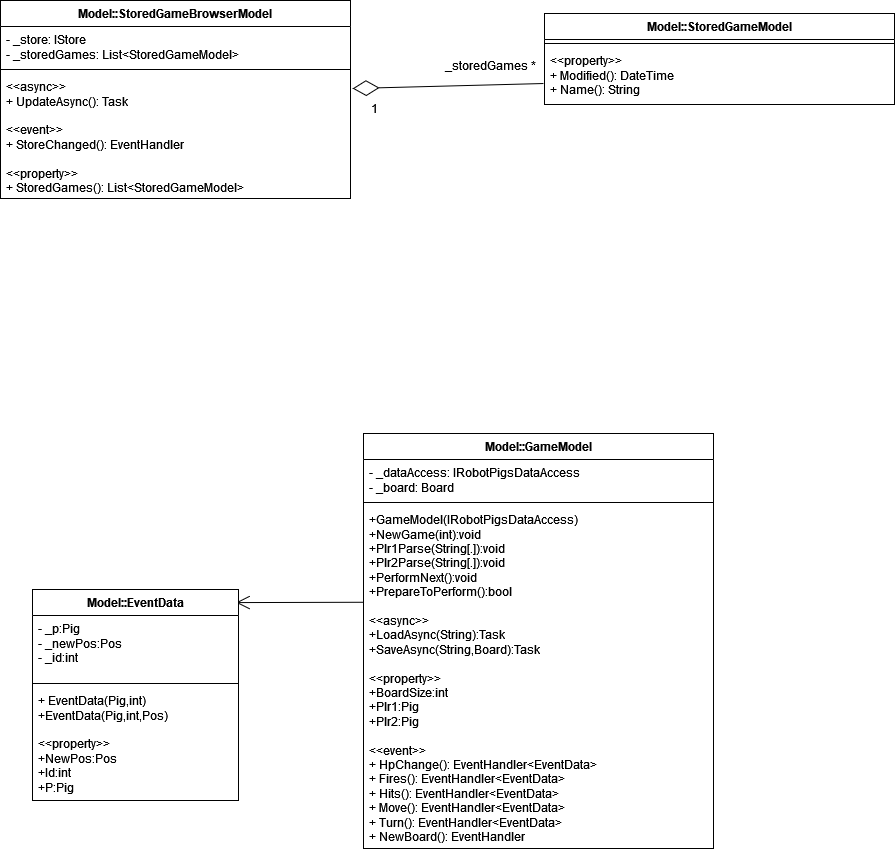
A kezdés gomb megnyomása után a parancsok beütése illetve a játék mentése elérhetetlenné válik amíg a kör be nem fejeződik, a „Következő” gomb 5 darab megnyomása által.

**Tervezés:**

* Programszerkezet:
  + A programot két projektben valósítjuk meg:
    - Egy .NET Class Library, ami a modellt és a perszisztenciát tartalmazza
    - Egy .NET MAUI többplatformos projekt.
  + A programot MVVM architektúrában valósítjuk meg, architektúrában, 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 megvalósításból külön építjük fel a játék, illetve a betöltés és mentés funkciót, valamennyi rétegben. Utóbbi funkcionalitást újrahasznosítjuk egy korábbi projektből, így nem igényel újabb megvalósítást
  + A program csomagszerkezete itt látható:
  + 
* Perszistencia:
  + A „Board” mindig egy érvényes táblát ábrázol, azaz az osztály illetve adattagjai, mindig ellenőrzik a beállított értékeket.
  + A tábla illetve adattagjai lehetőséget adnak az értékeik lekérdezésére, property-k által, illetve egyes értékek beállítására settereken keresztül.
  + A lemezre való mentés lehetőségét az „IRobotPigsDataAccess” interfész adja. A töltéshez „LoadAsync”, a mentéshez pedig „SaveAsync” műveletet lehet használni.
  + Ezt az interfészt a „RobotPigsDataAccess” osztály valósítja meg, a fellépő esetleges hibákat a „BoardDataException” kivétel jelzi.
  + A program az adatokat szöveges fájlként tudja eltárolni, amelyeket egy megadott könyvtárban (\_directory) helyez el. Ez majd az alkalmazás platformfüggő saját adatkönyvtára lesz. Ezt a „Store” nevű osztály kezeli.
  + A program az adatokat szöveges fájlban menti el, ahol az első sorban a tábla mérete található, a következő két sorban pedig a malacok tulajdonságait írja le számokkal reprezentálva: X Y koordináta, majd az irány és végül az életereje.



* Model:
  + A „GameModel” osztály valósítja meg, a nézet ezen keresztül tudja módosítani a táblát, illetve ezen keresztül tudja elérni tábla adatait is.
  + Lehetőséget ad egy új játék kezdésére, avagy annak mentésére, betöltésére.
  + A játék állapotának változásáról hat darab különböző cselekvéshez kapcsolt eseménnyel értesíti a nézetet. Egy eseten kívül mindig egy „EventData” osztályban küldi el az új, illetve az esemény szempontjából hasznos adatokat.



* Nézetmodell:
  + A nézetmodell megvalósításához felhasználunk egy általános utasítás

(DelegateCommand), valamint egy ős változásjelző (ViewModelBase)

osztályt.

* + A nézetmodell feladatait a ViewModel osztály látja el, amely

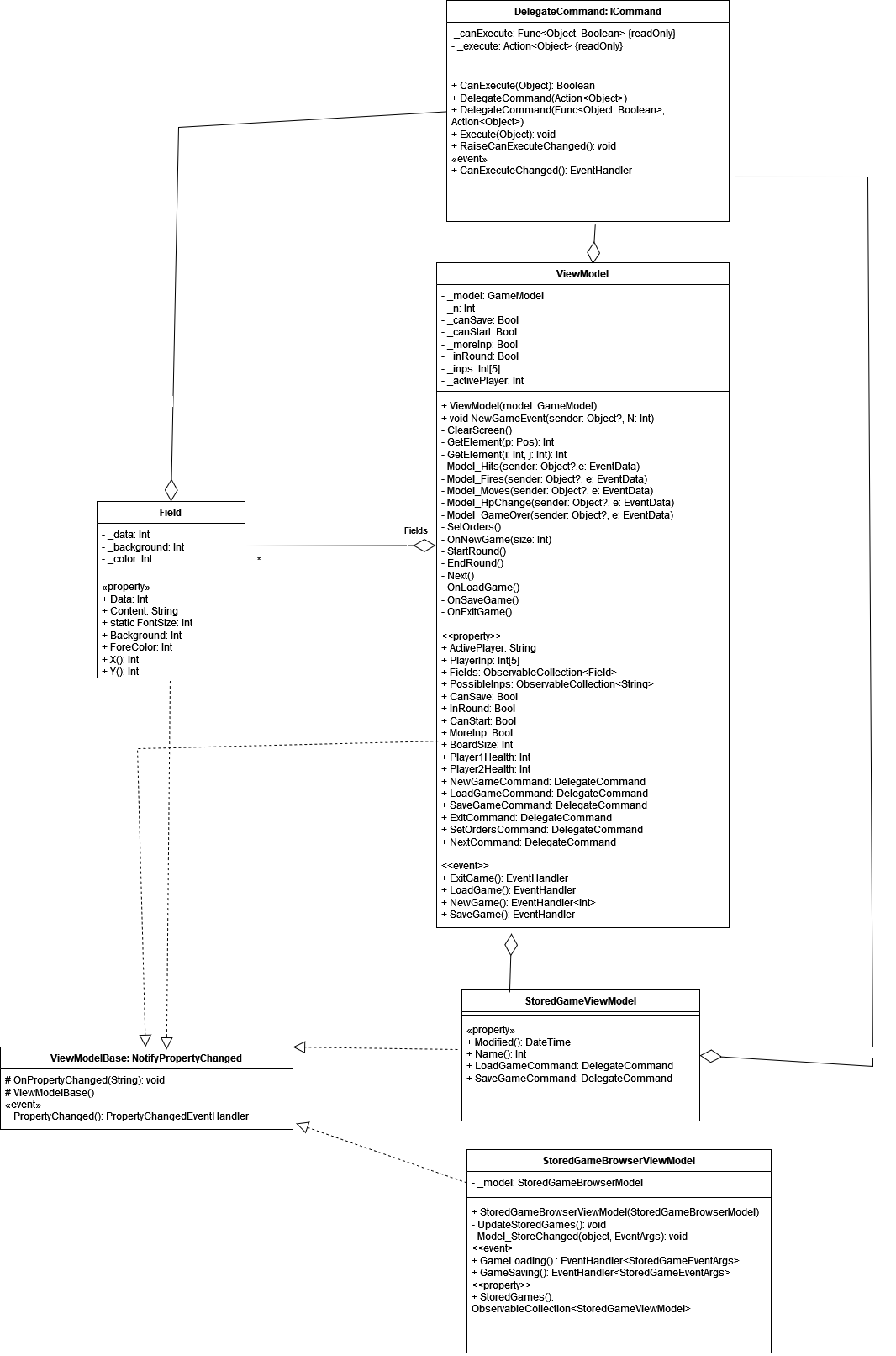
parancsokat biztosít az új játék kezdéséhez, játék betöltéséhez,

mentéséhez, valamint a kilépéshez. A parancsokhoz eseményeket kötünk,

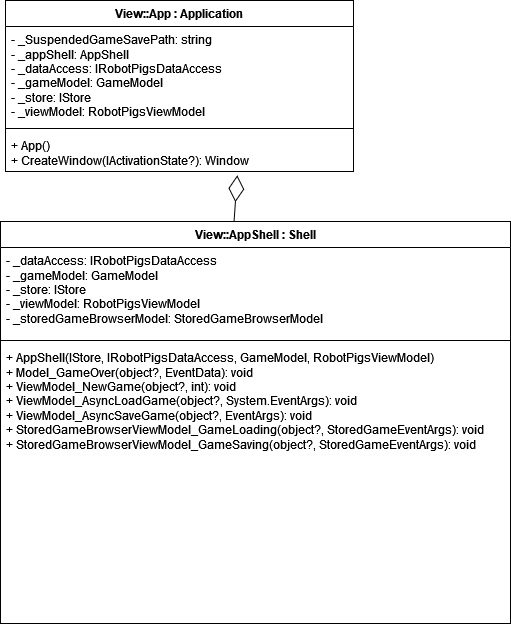
amelyek a parancs lefutását jelzik a vezérlőnek. A nézetmodell tárolja a

modell egy hivatkozását (\_model), de csupán információkat kér le tőle vagy a felhasználó által szolgáltatott bemeneteket továbbítja. Direkt nem avatkozik a játék futtatásába.

* + A játékmező számára egy külön mezőt biztosítunk (Field), amely eltárolja a pozíciót, a rajta lévő karakter állását, az őt támadó karakter azonosítóját. A mezőket egy felügyelt gyűjteménybe helyezzük a nézetmodellbe (Fields)
  + A tárolt játék állapotok kollekcióit egy „StoredGameViewModel”-lel tudjuk megjeleníteni. A mentés/betöltés gomb lenyomására lehet ezeket megnyitni.



* Nézet:
  + A nézetet navigációs lapok segítségével építjük fel.
  + A „MainPage” nézetben található egy MenuBar ahonnét lehet új játékot indítani, vagy elmenteni az állapotot vagy egy korábbi mentés betölteni, új lapok segítségével. A LoadGamePage és a SaveGamePage szolgál egy létező játékállapot betöltésére, illetve egy új mentésére
  + A „MainPage” osztály tartalmazza a játéktáblát, amelyet egy Grid segítségével valósítunk meg, amelyben TODO elemeket helyezünk el.
* Vezérlés:
  + Az App osztály feladata az alkalmazás vezérlése, a rétegek példányosítása és az események feldolgozása.
  + • A CreateWindow metódus felüldefiniálásával kezeljük az alkalmazás életciklusát a megfelelő eseményekre történő feliratkozással. Így az lkalmazás felfüggesztéskor (Stopped) elmentjük az aktuális játékállást (SuspendedGame), míg folytatáskor vagy újraindításkor (Activated) pedig folytatjuk, amennyiben történt mentés.
  + Az alkalmazás lapjait egy AppShell keretben helyezzük el. Ez az osztály felelős a lapok közötti navigációk megvalósításáért

****

**Tesztelés:**  
A modell működőképességét egységtesztek ellenőrzik. Ezek a következő osztályokban és funkciók csoportjaiban teljesülnek.

BoardTest

├ PlrPosTests: Egy új játék indulásakor a malacok elhelyezkedése és életereje.

├ PerformTests: Aktív játék során bemenetek feldolgozása.

PosTest

├ AddRelativeDirectionTests: Relatív irányok összeadása

├ MoveTests: Mozgatások

├ MoveTestHitWallX: olyan mozgás amikor „X” falnak nekimegy egy malac

├ InviewTest: Eltalál-e egy malac a másik malacot a helyzeteik alapján.

├ InRadiusTest: Megtud-e ütni egy malac a másik malacot a helyzeteik alapján.

PigTest

├ ValidateTests: Lehetséges parancssorozatok validációja.

RobotDataAccessTest

├ Test1: Egy új játék elindítása, mentése betöltése.

└ Test2: Egy már folyó játék elindítása.