1. Beadandó feladat dokumentáció

# Készítette:

Golyha Gergő

A7MMZ1

E-mail: [golyhagergo@gmail.com](mailto:golyhagergo@gmail.com)

# 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:

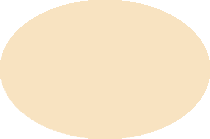
* előre, hátra, balra, jobbra: egy mezőnyi lépés a megadott irányba, közben a robot iránya nem változik.
* fordulás balra, jobbra: a robot nem vált mezőt, de a megadott irányba fordul.
* tűz: támadás előre a lézerágyúval.
* ütés: támadás a támadóököllel.

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.

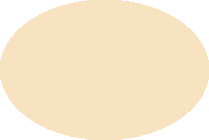
A program biztosítson lehetőséget új játék kezdésére a pályaméret megadásával (4 × 4, 6 × 6, 8 × 8), valamint játék mentésére és betöltésére. Ismerje fel, ha vége a játéknak, és jelenítse meg, melyik játékos győzött. Játék közben folyamatosan jelenítse meg a játékosok aktuális sérülésszámait.

# Elemzés:

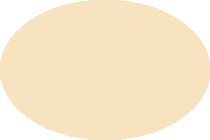
* A feladatot egyablakos asztali alkalmazásként Windows Forms grafikus felülettel valósítottam meg.
* A játékot három táblamérettel játszható: 4 x 4, 6 x 6, 8 x 8. A program indításkor   
  6 x 6-os méretet állít be, és automatikusan új játékot indít.
* Az ablakban felépítése: felül menü a következő menüpontokkal: Options (New Game (4 x 4, 6 x 6, 8 x 8), Load Game, Save Game, Exit); alul státuszsor, amely a játékosok hátralevő életerejét (ahányszor még eltalálhatják) jelzi.
* A játéktáblát egy N x N címkékből álló rács reprezentálja.
* A két felhasználó nyomógombok segítségével ad meg parancsokat, amit egy-egy listában jelenít meg a program. Egyszerre csak egy felhasználó adhat meg parancsokat, amikor is nem látja a másik által megadott parancsokat. Amikor mind a két felhasználó megadta a parancsait, akkor elkezdődhet a kiértékelés, ahol egyszerre egy pár parancs kerül kiértékelésre, párhuzamosan. A kiértékelés gombnyomásra halad előre a következő parancs párosra.
* A párhuzamos kiértékelés egyszerűsítése érdekében bevezettem egy prioritást, ami alapján sorrendiség alakul ki a két parancs kiértékelése között: a kisebb prioritásszámú parancs előnyt élvez a nagyobb prioritásszámúval szemben. Növekvő sorrendben a prioritások: lépés, forgás, ütés, lövés.
* A játék automatikusan feldob egy dialógusablakot, amikor vége a játéknak, azaz az egyik játékos életereje elfogyott. Szintén dialógusablakokkal történik a mentés, illetve betöltés, mindkét esetben a fájlneveket a felhasználó adja meg.
* A felhasználói esetek az 1. ábrán láthatóak.



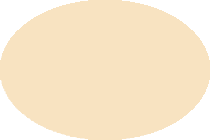
**Betöltés**



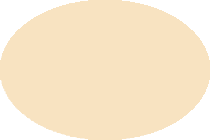
**Kilépés**



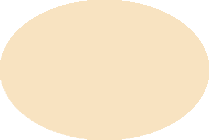
**Mentés**



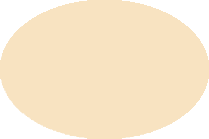
**Lépés kiértékelése**



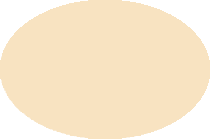
**Lépés megadása**



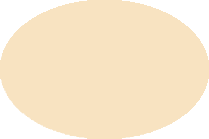
**Új Játék**



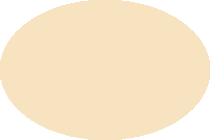
**4 x 4**



**6 x 6**



**8 x 8**



**Játék vége**

**Felhasználó**

«precede»

«precede»

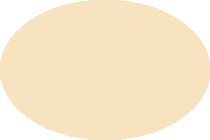
«precede»

«invoke»

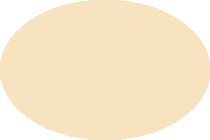
«include»

«include»

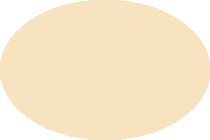
«include»



**1-es játékos győz**



**2-es játékos győz**



**döntetlen**

«include»

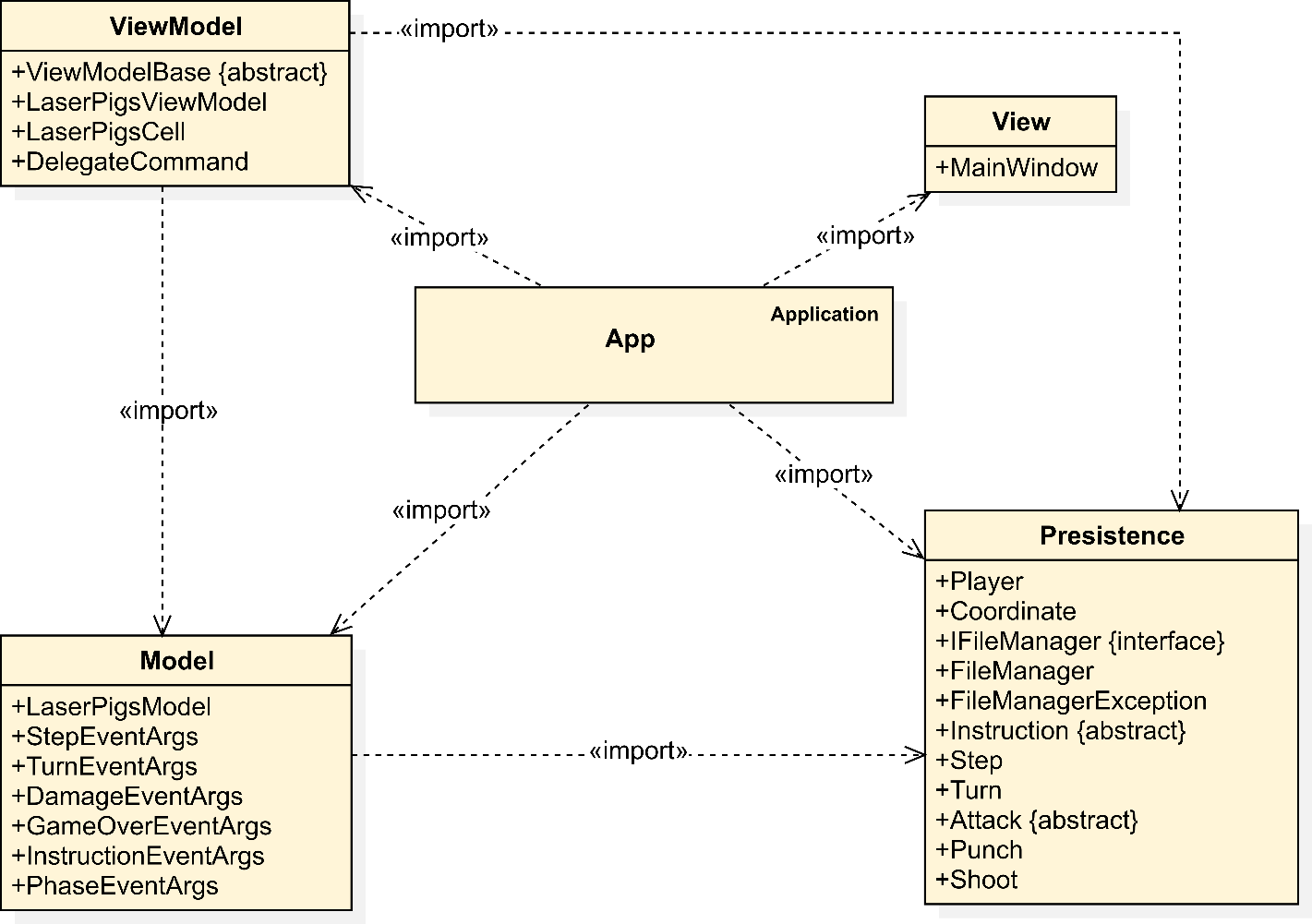
«include»

«include»

**1. ábra: Felhasználói-esetek diagramja**

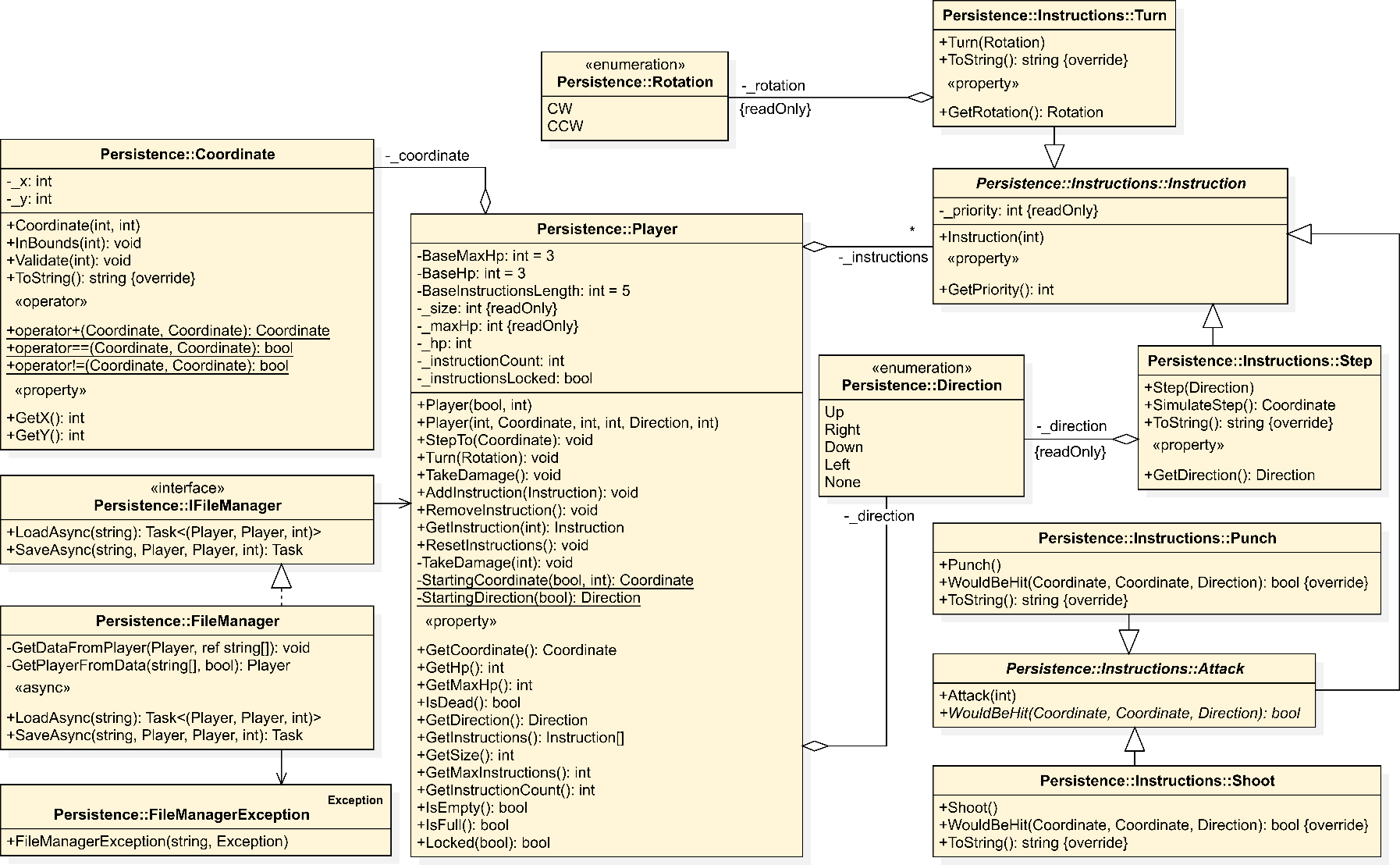
# Tervezés:

* Programszerkezet:
  + A programot MVVM architektúrában valósítottam meg, ennek megfelelően **View**, **Model**, **ViewModel** és **Persistence** névtereket valósítottam 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 2. ábrán látható.
  + A program szerkezetét két projektre osztottam implementációs megfontolásból: a **Persistence** és **Model** csomagok a program felületfüggetlen projektjében, míg a **ViewModel** és **View** csomagok a WPF függő projektjében kapnak helyet.



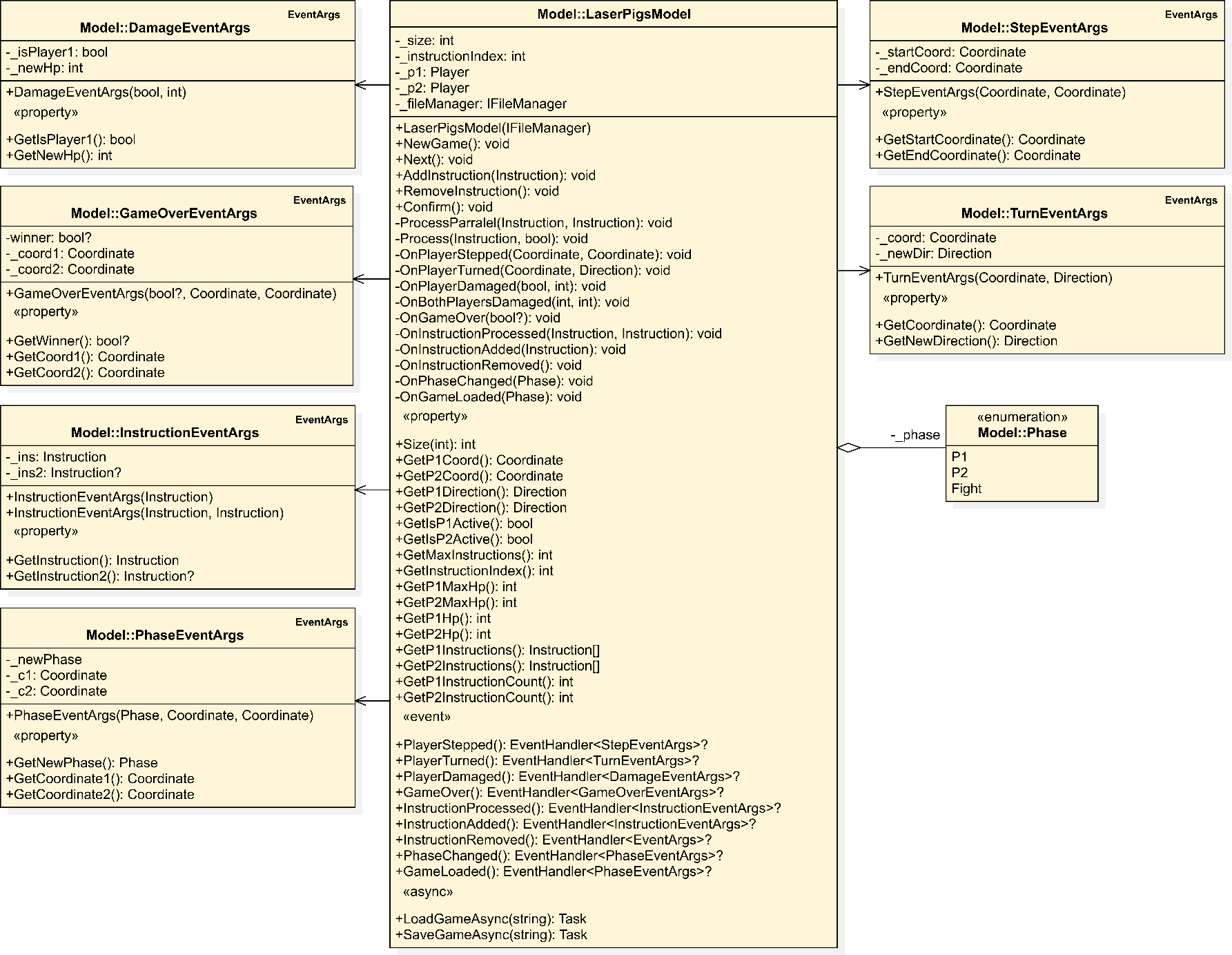
**2. ábra: Az alkalmazás csomagdiagramja**

* Perzisztencia (3. ábra):
  + Az adatkezelés feladata a játékosokkal kapcsolatos információk tárolása, valamint a betöltés és mentés biztosítása.
  + A **Player** osztály egy érvényes játékost biztosít (azaz mindig ellenőrzi a beállított értékek). Egy **Player** eltárolja a játéktábla mérete (\_**size**), a saját koordinátáját ezen a táblán (\_**coordinate**), a kezdeti (tehát maximum) és jelenlegi életerejét (\_**maxHp**, \_**hp**), az irányt, amerre néz (\_**direction**), a parancsai tömbjét (\_**instructions**), valamint ennek zároltságát és a benne tárolt parancsok számát **(\_instructionsLocked, \_instructionCount**).
  + A **Player** adattagjai létrehozásakor alapértelmezett esetben a feladat által definiált értéket veszik fel, csak a kezdő koordináta és a kezdeti irány függ a játékos számától és a játéktábla méretétől. Természetesen van lehetőség paraméterezetten is **Player** objektumot létrehozni, ez fájlból való betöltéskor kulcsfontosságú.
  + A **Player** osztály lehetőséget ad az adatok lekérdezésére a számos tulajdonságán keresztül, parancsok szabályos hozzáadására, törlésére, lekérdezésére és zárolására (**AddInstruction, RemoveInstruction, ResetInstructions, GetInstruction, Locked**), a koordináta és az irány szabályos változtatására (**StepTo, Turn**), valamint az életerő szabályos csökkentésére (**TakeDamage**).
  + A hosszú távú adattárolás lehetőségeit az **IFileManager** interfész adja meg, amely lehetővé teszi a játékosok betöltését (**LoadAsync**), valamint mentését (**SaveAsync**). A műveleteket hatékonysági okokból aszinkron módon valósítottam meg.
  + Az interfészt szöveges fájl alapú adatkezelését a **FileManager** osztály valósítja meg. A fájlkezelés során fellépő hibákat a **FileManagerException** kivétel jelzi.
  + A program az adatokat szöveges fájlként tudja eltárolni, amelyek az **lspf** 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, a második a parancs tömbök hosszát, a harmadik pedig a feldolgozandó parancs sorszámát, ha abban a fázisban tart a játék. A 5 sorban az első, majd 5 sorban a második játékos adatai szerepelnek: koordináta szóközzel elválasztva; kezdeti és jelenlegi életerő szóközzel elválasztva; irány, amerre a játékos néz; a parancsok tömbje; ennek a tömbnek a zároltsága.
    - * A parancsok tömbjének formátuma: a parancsok vesszővel vannak elválasztva, ezen belül pedig szóközzel vannak elválasztva a paraméterezett parancsok (lépés, fordulás), és az adott parancs azonosítója áll az első helyen
  + A **Coordinate** osztály kényelmes megoldást biztosít a koordináták kezelésére, definiálva van rajtuk az összeadás, az egyenlőség vizsgálat, valamint a validáció és a „visszalökés” is (ha nem lenne a táblán az adott koordináta, akkor ,,visszalöki” úgy, hogy rajta legyen).
  + Az **Instruction** absztrakt osztály valósítja meg a parancsokat, alapból csak a prioritását tárolja el, a többi funkcionalitás a gyerekosztályokban (**Step, Turn, Attack, Punch, Shoot**) van megvalósítva.
    - * A **Step** osztály képes egy lépés „vektort” (delta koordinátát) visszaadni, amit a játékosok léptetése használ fel.
      * A **Turn** osztályt a játékosok forgatása használ fel.
      * Az **Attack** absztrakt osztály kibővíti az **Instruction** osztályt egy új metódussal, amely logikai értéket ad vissza. Ez az érték attól függ, hogy az adott támadás eltalálná-e a másik játékost. Ennek a megvalósítása a gyerekosztályok feladata.
        + A **Punch** osztály megvalósítja az imént említett metódust úgy, hogy a támadó körüli szomszédos cellákat találja el.
        + A **Shoot** osztály megvalósítja az imént említett metódust úgy, hogy amerre támadó néz, arra egy vonalban az összes cellát eltalálja.



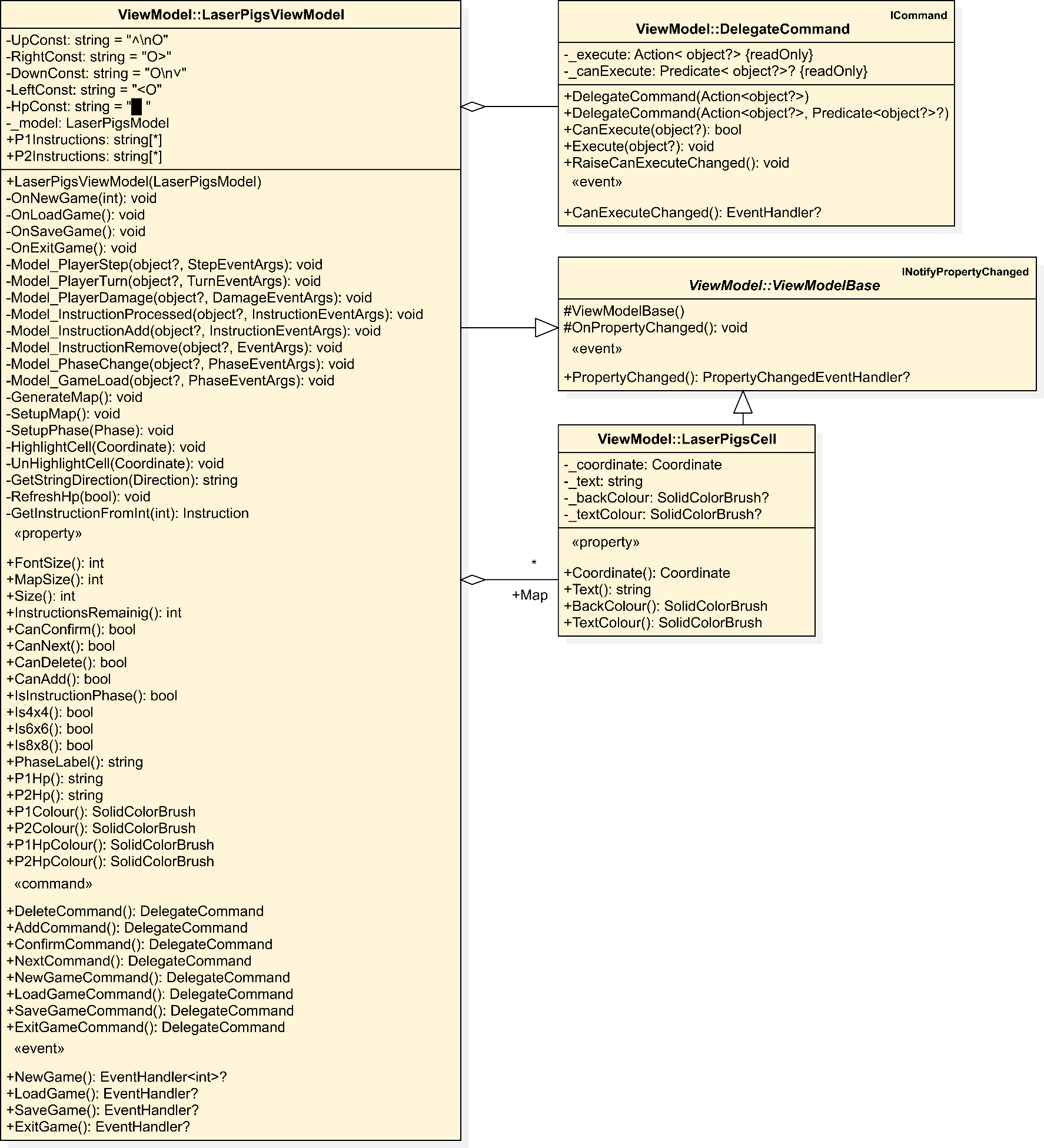
**3. ábra: A Persistence csomag osztálydiagramja**

* Modell (4. ábra):
  + A modell lényegi részét a **LaserPigsModel** osztály valósítja meg, amely szabályozza a két játékost, valamint a játék egyéb paramétereit, mint a játék fázisa (**\_phase**) és a kiértékelendő parancs indexe (**\_instructionIndex**). A típus lehetőséget ad új játék kezdésére (**NewGame**), valamint parancsok kezelésére (**AddInstruction, RemoveInstruction, Next**) és a fázis szabályos módosítására (**Confirm**).
  + A játék betöltéséről saját esemény (**GameLoaded**) tájékoztat, ekkor frissül a nézet szinte összes eleme.
  + A játékosok állapotváltozásáról (lépés, fordulás, sebesülés) számos eseménytájékoztat. Az események különböző argumentuma tárolja a megváltozott játékos új adatait.
  + A parancsok módosításáról (hozzáadás, törlés, kiértékelés) és a fázis változásáról is **különböző** események tájékoztatnak. Az események különböző argumentuma tárolja ezek megjelenítéséhez szükséges adatokat.
  + A játék végét egyedi esemény (**GameOver**) jelzi, ennek az argumentuma (**GameOverEventArgs**) tárolja a nyertes játékost.
  + Törekedtem a kommunikációs hatékonyságra, tehát csak azok az adatok kerülnek az események argumentumába, amelyek ténylegesen megváltoznak.
  + A modell példányosításkor megkapja az adatkezelés felületét, amelynek segítségével lehetőséget ad betöltésre (**LoadGameAsync**) és mentésre (**SaveGameAsnyc**)
  + A modell új játék kezdése esetén az akkor eltárolt mérettel kezd új játékot.



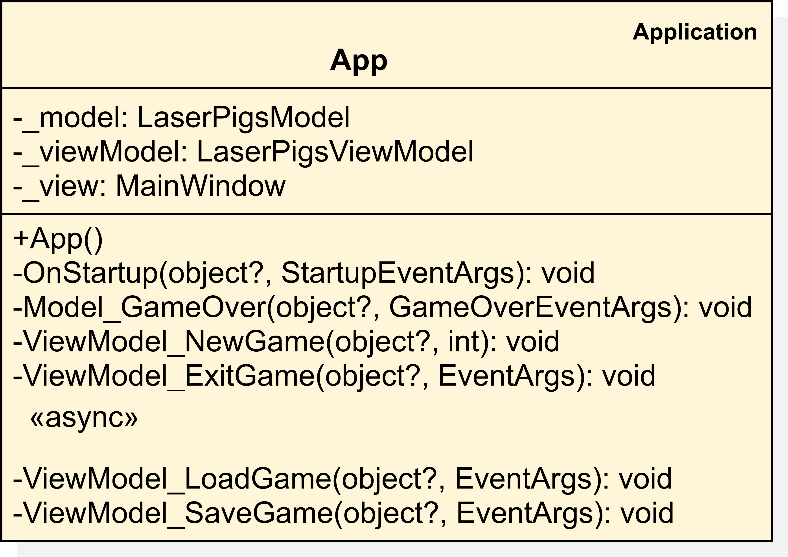
**4. ábra: A Model csomag osztálydiagramja**

* Nézetmodell (5. ábra):
  + • A nézetmodell megvalósításához felhasználtam egy általános utasítás (**DelegateCommand**), valamint egy ős változásjelző (**ViewModelBase**) osztályt.
  + A nézetmodell feladatait a **LaserPigsViewModel** osztály látja el, amely parancsokat biztosít az új játék kezdéséhez, játék betöltéséhez, mentéséhez, valamint a kilépéshez. A parancsokhoz eseményeket kötöttem, amelyek a parancs lefutását jelzik a vezérlőnek. A nézetmodell tárolja a modell egy hivatkozását (\_**model**).
  + A játékmező számára egy külön mezőt biztosítottam (**LaserPigsCell**), amely eltárolja a pozíciót, szöveget, a háttér – és szövegszínt. A mezőket egy felügyelt gyűjteménybe helyeztem a nézetmodellbe (**Map**).
  + A listadobozok tartalmát szintén felügyelt listák tárolják, így a megjelenített elemek frissítés autómatikusan történik.
  + A számos vezérlő gombhoz parancsok tartoznak, ezek a parancsok hívják meg a modell megfelelő metódusait.
  + A nézetmodell kezeli a modell majdnem összes eseményét, ezzel frissítve a nézetet.
* Nézet:
  + A nézet csak egy képernyőt tartalmaz, a **MainWindow** osztályt. A nézet egy rácsban tárolja a játékmezőt, a menüt és a státuszsort. A játékmező egy **ItemsControl** vezérlő, ahol dinamikusan felépítettem egy rácsot (**UniformGrid**), amely címkékből áll. Minden adatot adatkötéssel kapcsoltam a felülethez, továbbá azon keresztül szabályzom a címkék színét és szövegét, valamint a gombok állapotát (ki/be kapcsolt).
  + A fájlnév bekérését betöltéskor és mentéskor, valamint a figyelmeztető üzenetek megjelenését beépített dialógusablakok segítségével végeztem.



**5. ábra: A ViewModel csomag osztálydiagramja**

* Környezet (6. ábra):
  + aAz **App** osztály feladata az egyes rétegek példányosítása (**OnStartup**), 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 funkcionalitását egységtesztek segítségével ellenőriztem a **LaserPigsTest** osztályban.
* Az alábbi tesztesetek kerültek megvalósításra:
  + **TestNewGame**: Új játék indítása.
  + **TestLoadGameSameCoords, TestLoadGameInvalidP2, TestLoadGameInvalidInstructionIndex, TestLoadGame:** A játék modell betöltésének tesztelése mockolt perzisztencia réteggel. Hibás formátum betöltése, esemény kiváltásának ellenőrzése.
  + **TestSaveGameVerify, TesttSaveNewGame:** A játék modell mentésének tesztelése mockolt perzisztencia réteggel.
  + **TestAddInstructionFull, TestAddInstructionLocked, TestAddInstruction:** Parancs hozzáadásának tesztelése. Teli, zárolt esetben parancsok módosítása, esemény kiváltásának ellenőrzése.
  + **TestRemoveInstructionEmpty, TestRemoveInstructionLocked, TestRemoveInstruction:** Parancs törlésének tesztelése. Üres, zárolt esetben parancsok módosítása, esemény kiváltásának ellenőrzése.
  + **TestNextWrongPhase, TestNextAllDone, TestNextStep, TestNextTurn, TestNextAttackVictory, TestNextAttackDraw:** Parancs feldolgozásának tesztelése. Helytelen fázisban próbálkozás, különböző parancsok hatásának vizsgálata, játék vége, esemény kiváltásának ellenőrzése.
  + **TestConfirmP1NotFull, TestConfirmP1, TestConfirmP2NotFull, TestConfirmP2, TestConfirmFightNotDone, TestConfirm:** Fázis módosításának tesztelése. Nem zárolt vagy feldolgozatlan parancsokkal próbálkozás, esemény kiváltásának ellenőrzése.