1. Beadandó feladat dokumentáció

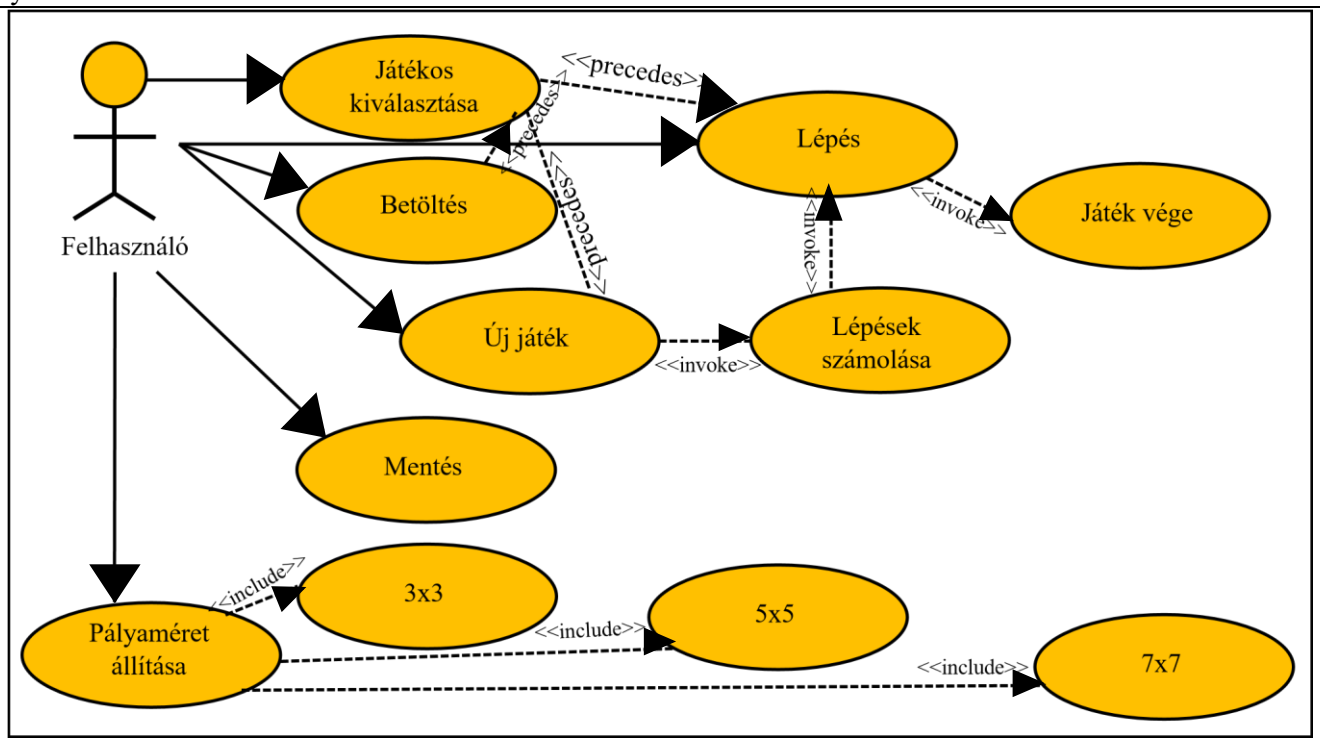
Készítette:
Ferenczy Kata
Neptun: G09BOX

Feladat:

Készítsünk programot, amellyel a következő két személyes játékot lehet játszani. Adott egy $n \times n$ mezőből álló tábla, ahol egy menekülő és egy támadó játékos helyezkedik el. Kezdetben a menekülő játékos figurája középen van, míg a támadó figurái a négy sarokban helyezkednek el. A játékosok felváltva lépnek. A figurák vízszintesen, illetve függőlegesen mozoghatnak 1-1 mezőt, de egymásra nem léphetnek. A támadó játékos célja, hogy adott lépésszámon ($4n$) belül bekerítse a menekülő figurát, azaz a menekülő ne tudjon lépni. A program biztosítson lehetőséget új játék kezdésére a táblaméret (3×3, 5×5, 7×7) és így a lépésszám (12, 20, 28) megadásával, folyamatosan jelenítse meg a lépések számát, és ismerje fel, ha vége a játéknak. Ekkor jelenítse meg, melyik játékos győzött, majd kezdjen automatikusan új játékot. Ezen felül legyen lehetőség a játék elmentésére, valamint betöltésére.

Elemzés:

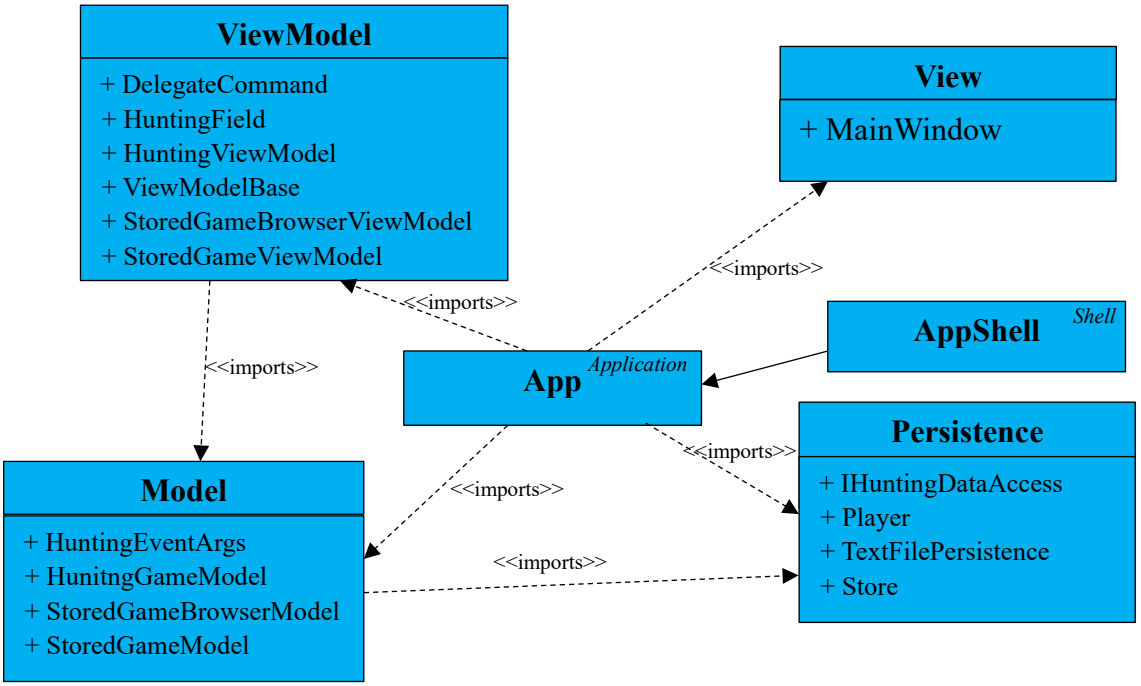
- A játékot három különböző méretű táblán lehet játszani (3x3, 5x5, 7x7), amit a felhasználó tud beállítani egy legördülő menü segítségével az ablak bal oldalán, amin kezdetben a 3x3 nehézség van kijelölve.
- A feladatot .NET MAUI alkalmazásként, elsődlegesen Windows és Android platformon valósítjuk meg. Az alkalmazás három lapból fog állni. Az alkalmazás portré tájolást támogat.
- A játék három képernyőn fog megjelenni.
    - Az első képernyő (Játék) tartalmazza a játéktáblát, a játék állását (lépések száma, következő játékos) a lap alján, az új játék, mentés, illetve betöltés gombjait a lap tetején.
    - A további két képernyő a betöltésnél, illetve mentésnél megjelenő lista, ahol a játékok elnevezése mellett a mentés dátuma is látható. Mentés esetén ezen felül lehetőség van új név megadására is.
- A játéktáblát egy, a kiválasztott méretnek megfelelő képekből álló rács reprezentálja. A kép egérkattintás hatására kiválasztja a soron következő játékos figuráját, amivel a lépést szeretné végrehajtani, majd még egy egérkattintás hatására kiválasztjuk azt az üres mezőt, ahova a játkos lépni szeretne, majd megcseréljük a két kép koordinátáját a rácson. Nem engedjük, hogy a játékos a saját figuráin kívül más képet válasszon ki, illetve, hogy a játékos nem megengedett koordinátára lépjen.
- A játék automatikusan feldob egy dialógusablakot, amikor vége a játéknak (elértük a maximális lépések számát, vagy a menekülő játékos nem tud lépni). Szintén dialógusablakokkal végezzük el a mentést, illetve a betöltést, a fájlneveket a felhasználó adja meg.
- A felhasználói esetek:



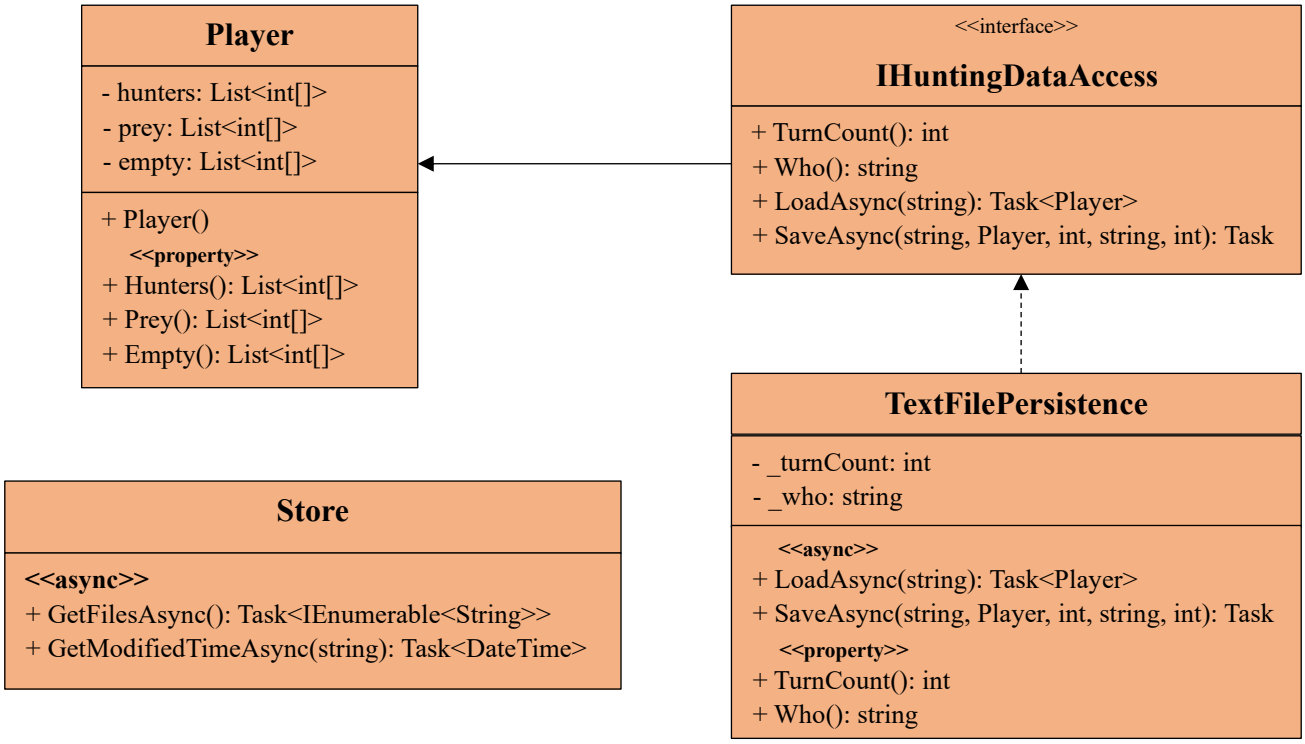
Tervezés: •

Programszerkezet:

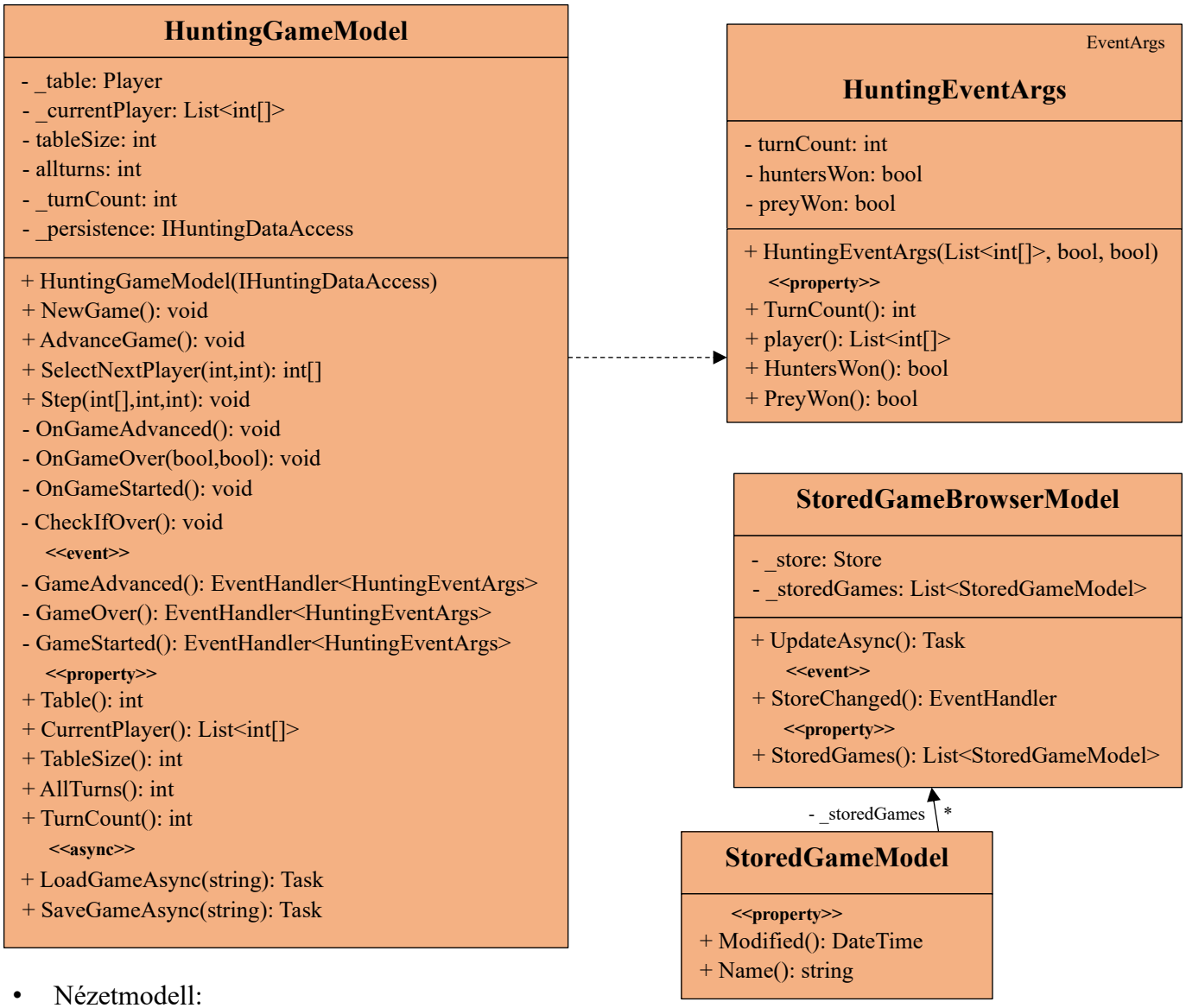
- A programot MVVM architektúrában valósíjuk meg. A megjelenítés a View, a modell a Model, a nézetmodell a ViewModel, míg a perzisztencia a Persistence névtérben helyezkedik el. A program csomagszerkezete a következő ábrán látható.
- A szoftvert két projektből építjük fel: a modellt és a perzisztenciát tartalmazó osztálykönyvtárból (.NET Standard Class Library), valamint a .NET MAUI többplatformos projektből, amelyet Windows és Android operációs rendszerre is le tudunk fordítani.
- 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 vezérlését az alkalmazás osztály (App) végzi, amely példányosítja a modellt, a nézetmodell és a nézetet, biztosítja a kommunikációt, valamint felügyeli az adatkezelést.



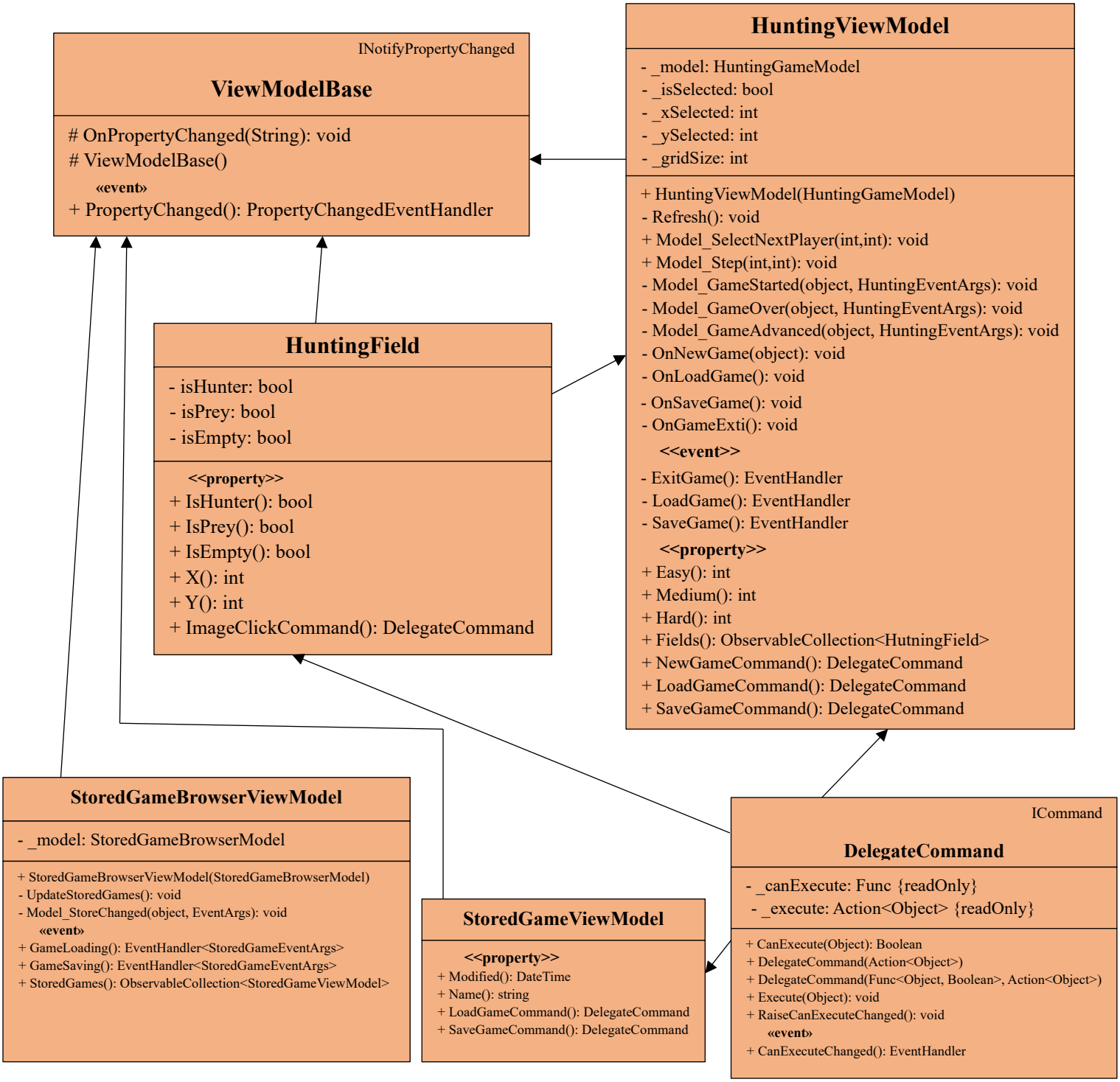
- Perzisztencia:
  - Az adatkezelés feladata a Hunting táblával kapcsolatos információk tárolása, valamint a betöltés/mentés biztosítása.
  - A Player osztály egy Player táblát biztosít, ahol ismert a támadó játékos figuráinak koordinátája (hunters), a menekülő játékos koordinátái (prey) és a tábla üres mezőinek koordinátái (empty).
    - A hosszú távú adattárolás lehetőségeit az IHuntingDataAccess interfész adja meg, amely lehetőséget ad a tábla betöltésére (Load), valamint mentésére (Save). A műveleteket hatékonysági okokból aszinkron módon valósítjuk meg.
  - Az interfészt szöveges fájl alapú adatkezelésre a TextFilePersistence osztály valósítja meg. A fájlkezelés során fellépő hibákat a DataException kivétel jelzi.
  - A program az adatokat szöveges fájlként tudja eltárolni, melyek .txt kiterjesztést kapnak. Ezeket az adatokat a programban bármikor be lehet tölteni, illetve ki lehet menteni az aktuális állást.
  - A fájl első sora megadja a támadó játékos figuráinak koordinátáit, a második sor megadja a menekülő játékos figuráinak koordinátáit, a harmadik sor megadja az üres koordinátákat. A negyedik sor pedig az eltelt lépések számát, valamint a soron következő játékost.



- Model:
  - A modell lényegi részét a HuntingModel osztály valósítja meg, amely szabályozza a tábla tevékenységeit, valamint a játék egyéb paramétereit, úgymint a lépések száma(_turnCount) és a soron következő játékos (_currentPlayer). A típus lehetőséget ad új játék kezdésére (NewGame), a következő játékos kiválasztására (SelectNextPlayer) valamint lépésre (Step). Új játéknál megadjuk a játéktábla méretét.
  - A játék végéről a GameOver esemény tájékoztat. Az események argumentuma (GameWonEventArgs) tárolja a győzelem állapotát.
  - A modell példányosításkor megkapja az adatkezelés felületét, amelynek segítségével lehetőséget ad betöltésre (Load) és mentésre (Save).
  - A játéktábla méretét a size változóban tároljuk.

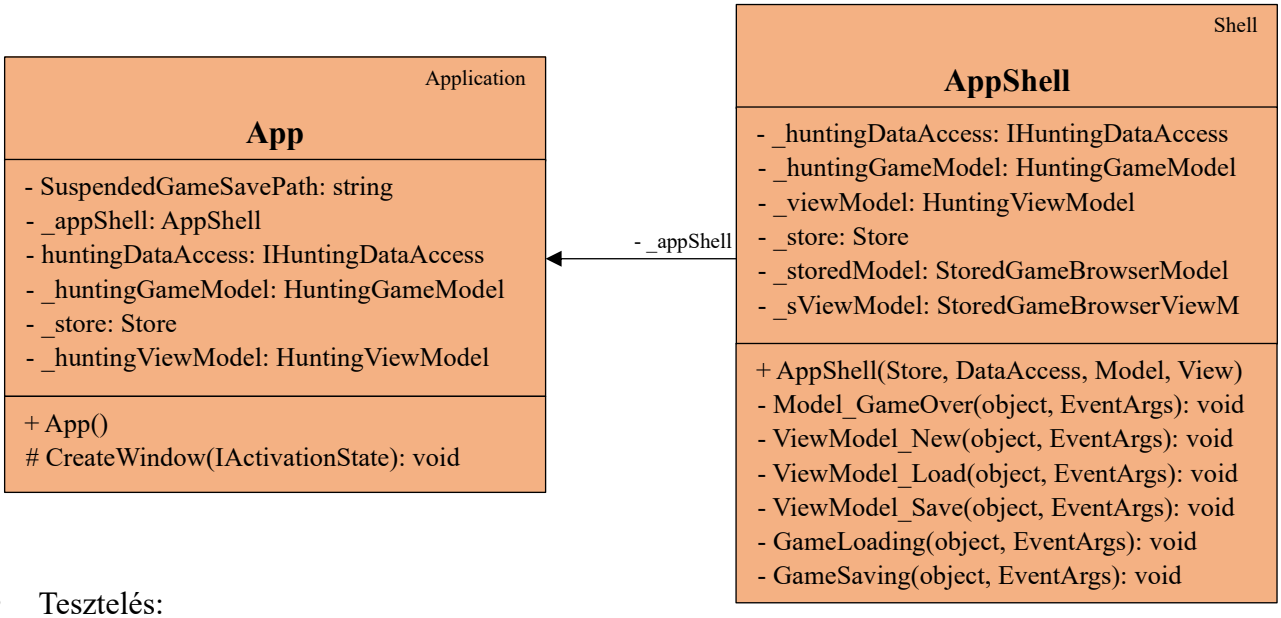


- Nézetmodell:
  - A nézetmodell megvalósításához felhasználunk egy általános utasítás (DelegateCommand), valamint egy ős változásjelző (ViewModelBase) osztályt.
  - A nézetmodell feladatait a HuntingViewModel osztály látja el, amely parancsokat biztosít az új játék kezdéséhez, játék betöltéséhez, mentéséhez, 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 tábla nagyságát szabályozza. Direkt nem avatkozik a játék futtatásába.
  - A játékmező számára egy külön mezőt biztosítunk (HuntingField), amely eltárolja a pozíciót, a mező típusát, valamint a lépés parancsát (ImageClickCommand). A mezőket egy felügyelt gyűjteménybe helyezzük a nézetmodellbe (Fields).



- Nézet:
  - A nézetet navigációs lapok segítségével építjük fel.
  - A GamePage osztály tartalmazza a játéktáblát, amelyet egy Grid segítségével valósítunk meg, amelyben Image elemeket helyezünk el.
  - A LoadPage és a SavePage szolgál egy létező játékállapot betöltésére, illetve egy új mentésére.

- Környezet:
  - 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 alkalmazás 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 funkcionalitása egységtesztek segítségével lett ellenőrizve a HuntingUnitTest osztályban.
  - Az alábbi tesztesetek kerültek megvalósításra:
  - **HuntingConstructorTest:** megnézi, hogy a NewGame() metódus hívása után helyesen inicializálódnak-e a modell konstruktorai.
  - **HuntingSelectNextPlayerTest:** megnézi, hogy a SelectNextPlayer() metódus szabályos hívása után megtörténik-e a kiválasztás, illetve, hogy a szabálytalan hívás helyesen kiváltja-e a kivételt.
  - **HuntingStepTest:** megnézi, hogy a Step() metódus szabályos hívása után megtörténik-e a lépés, illetve, hogy a szabálytalan hívás helyesen kiváltja-e a kivételt.
  - **HuntingLoadTest:** megnézi, hogy a sikertelen játék betöltés kiváltja-e a kivételt.
  - **HuntingSaveTest:** megnézi, hogy mentés után megmarad-e a tábla mentés előtti állapota.
  - **HuntingCheckIfOverTest:** megnézi, hogy a győzelem helyesen kiváltja-e a kellő eseményt.