1. **Beadandó feladat dokumentáció**

**Készítette:**Jécsák Fruzsina  
E-mail: [bq2rko@inf.elte.hu](mailto:bq2rko@inf.elte.hu)

**Feladat:**

1. Maci Laci

Készítsünk programot, amellyel a következő játékot játszhatjuk. Adott egy 𝑛×𝑛 elemből álló játékpálya, amelyben Maci Lacival kell piknikkosarakra vadásznunk. A játékpályán az egyszerű mezők mellett elhelyezkednek akadályok (pl. fa), valamint piknikkosarak. A játék célja, hogy a piknikkosarakat minél gyorsabban begyűjtsük.

Az erdőben vadőrök is járőröznek, akik adott időközönként lépnek egy mezőt (vízszintesen, vagy függőlegesen). A járőrözés során egy megadott irányba haladnak egészen addig, amíg akadályba (vagy az erdő szélébe) nem ütköznek, ekkor megfordulnak, és visszafelé haladnak (tehát folyamatosan egy vonalban járőröznek). A vadőr járőrözés közben a vele szomszédos mezőket látja (átlósan is, azaz egy 3×3-as négyzetet).

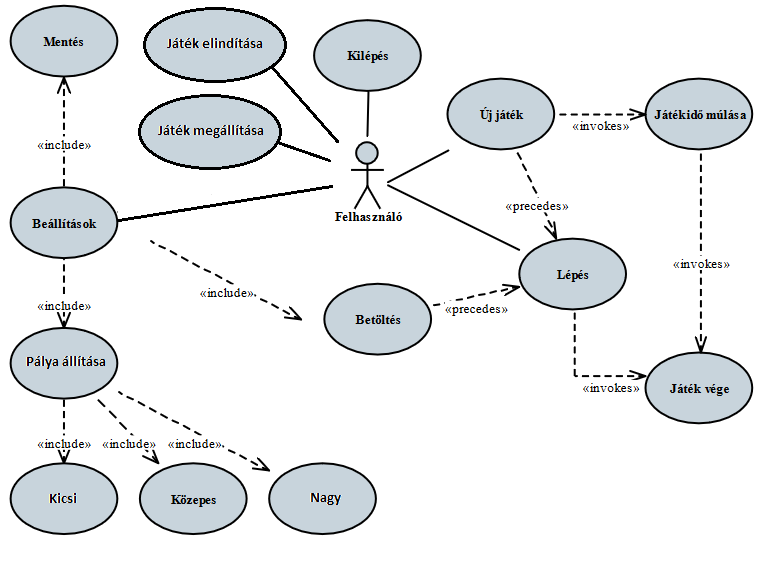
A játékos kezdetben a bal felső sarokban helyezkedik el, és vízszintesen, illetve függőlegesen mozoghat (egyesével) a pályán, a piknikkosárra való rálépéssel pedig felveheti azt. Ha Maci Lacit meglátja valamelyik vadőr, akkor a játékos veszít.

A pályák méretét, illetve felépítését (piknikkosarak, akadályok, vadőrök kezdőpozíciója) tároljuk fájlban. A program legalább 3 különböző méretű pályát tartalmazzon.

A program biztosítson lehetőséget új játék kezdésére a pálya kiválasztásával, valamint játék szüneteltetésére (ekkor nem telik az idő, és nem léphet a játékos). Ismerje fel, ha vége a játéknak, és jelezze, győzött, vagy veszített a játékos. A program játék közben folyamatosan jelezze ki a játékidőt, valamint a megszerzett piknikkosarak számát.

**Elemzés:**

* A feladatot Xamarin Forms alkalmazásként, Android platformon valósítjuk meg, amely két lapból fog állni.
* A játék így csak két képernyőn fog megjelenni:
* Az első képernyő (Game) tartalmazza a játéktáblát, a játék állását (felvett piknikkosarak száma, eltelt idő) a lap alján, az új játék, a start és stop, valamint a beállítások gombjait a lap tetején.
* A második képernyőn (Settings) van lehetőség a táblaméret kiválasztására (a táblák a program elején automatikusan betöltenek a project-be elhelyezett file-okból).
* A játékot három pályával játszhatjuk: kicsi (6x6-os táblaméret, 6 piknikkosár, 4 fa, 2 őr), közepes (12x12-es táblaméret, 12 piknikkosár, 8 fa, 4 őr), nagy (18x18-as táblaméret, 18 piknikkosár, 12 fa, 6 őr). A program indításkor kicsi pályát állít be, és automatikusan új játékot indít.
* A játéktáblát egy n×n (n pályától függ) nyomógombokból álló rács reprezentálja. A nyomógombot, amin Maci Laci áll, a tábla alatt található nyilak lenyomására üres mezőre változtatjuk, Maci Lacit pedig a megadott irányba léptetjük 1-el. Emellett az idő múlásával (másodpercenként) automatikusan léptetjük az őröket a haladási irányukba. A játéktáblán van még piknikkosár és akadály(fa).
* A játék automatikusan feldob egy dialógusablakot, amikor vége a játéknak (felvettük az összes piknikkosarat vagy meglátott minket egy őr).
* A felhasználói esetek az 1. ábrán láthatóak.

****

1. **Ábra: Felhasználói esetek diagramja**

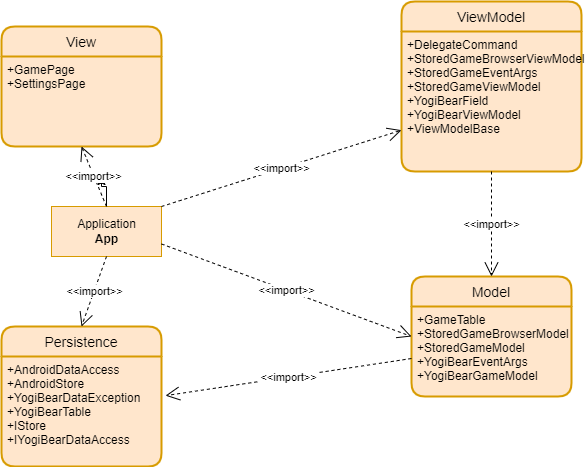
**Tervezés:**

* Programszerkezet:
* A szoftvert két projektből építjük fel, a Xamarin Forms megvalósítást  
  tartalmazó osztálykönyvtárból (.NET Standard Class Library), valamint az Android platform projektből. Utóbbi csupán a perzisztencia Android  
  specifikus megvalósítását tartalmazza, minden további programegységet az osztálykönyvtárban helyezünk el.

• A programot MVVM architektúrában valósíjuk megfelelően View, Model, ViewModel és Persistence névtereket valósítunk meg az alkalmazáson belül.

* A program vezérlését az alkalmazás osztály (App) végzi, amely példányosítja a modellt, a nézetmodellt és a nézetet, biztosítja a kommunikációt, valamint felügyeli az adatkezelést.

A program csomagszerkezete a 2. ábrán látható.

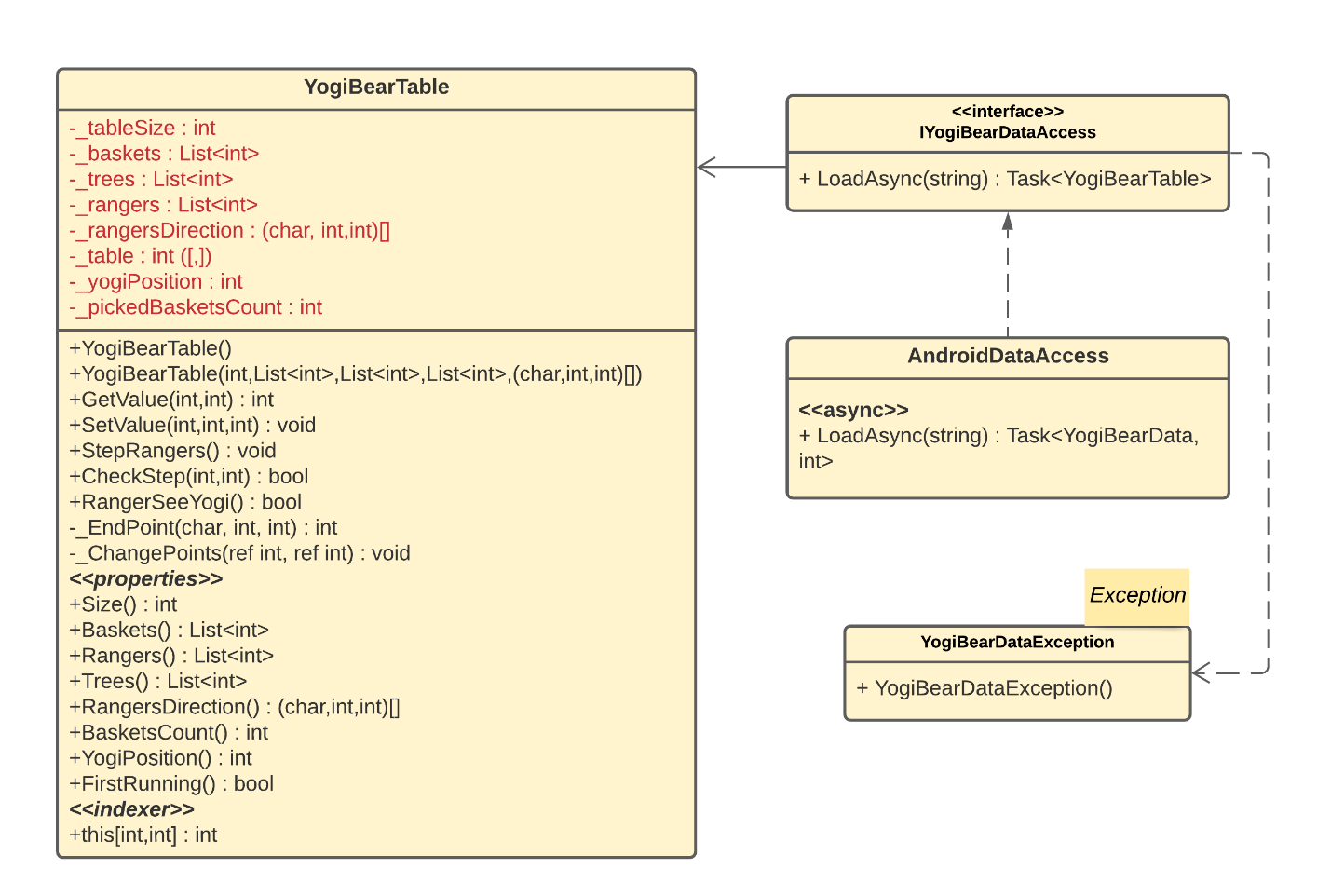


1. **Ábra: Az alkalmazás csomagdiagramja**

* Perzisztencia (3. ábra):

• Az adatkezelés feladata a YogiBear táblával kapcsolatos információk tárolása, valamint a pályák betöltésének biztosítása.   
• A YogiBearTable osztály egy érvényes YogiBear táblát biztosít (azaz mindig ellenőrzi a beállított értékeket), ahol minden mezőre ismert az értéke (\_table). Külön tároljuk a Maci Laci (\_yogiPosition), piknikkosarak (\_baskets), fák (\_trees) és az őrök pozícióját (\_rangers). Ezen kívül tároljuk azt is, hogy egy őr melyik irányba halad és mettől meddig (a sor/oszlop végéig vagy akadályig) (\_rangersDirection). A tábla alapértelmezés szerint a kicsi pálya paramétereivel jön létre, de ez a konstruktorban paraméterezhető. A tábla lehetőséget ad az állapotok lekérdezésére   
(Size, Baskets, Rangers, Trees, RangersDirection, BasketsCount, PickedBasketsCount, YogiPosition, FirstRunning, GetValue), valamint az őrök szabályos   
léptetésre (StepRangers), annak ellenőrzésére, hogy Maci Lacit látják-e az őrök (RangerSeeYogi), illetve direkt beállítás (SetValue) elvégzésére.

• A hosszú távú adattárolás lehetőségeit az IYogiBearDataAccess interfész adja meg, amely lehetőséget ad a pályák betöltésére (LoadAsync). A műveletet hatékonysági okokból aszinkron módon valósítjuk meg.   
• Az interfészt szöveges fájl alapú Android adatkezelésre a AndroidDataAccess osztály valósítja meg. A fájlkezelés során fellépő hibákat a YogiBearDataException kivétel jelzi.   
• A program a pályák adatait szöveges fájlként tudja eltárolni, melyek az txt kiterjesztést kapják. Ezeket az adatokat a program futásának kezdetekor (új játék előtt) be lehet tölteni.   
• A fájl első sorában szóközökkel elválasztva vannak az adatok. Az első a tábla mérete (n), ezt követi a kosarak pozícióinak listája, a fák pozícióinak listája, az őrök pozícióinak listája, majd az őrök iránytartásához szükséges adatok listája.



1. **Ábra: A Persistence csomag osztálydiagramja**

* Modell (4.ábra):

• A modell lényegi részét a YogiGameModel osztály valósítja meg, amely szabályozza a tábla tevékenységeit, valamint a játék egyéb paramétereit, úgymint az idő (\_gameTime), és azt, hogy éppen megy-e a játék (\_gameIsOn). A típus lehetőséget ad új játék kezdésére (NewGame), valamint lépésre (StepGame). Új játéknál megadható a kiinduló játéktábla is, különben automatikusan generálódnak kezdő mezők. Az idő előreléptetését időbeli   
lépések végzéséve (AdvanceTime) tehetjük meg.   
• A játékállapot változásáról a GameAdvanced esemény, míg a játék végéről a GameOver esemény tájékoztat. Az események argumentuma (YogiBearEventArgs) tárolja a győzelem állapotát, a lépések számát, valamint a játékidőt.   
• A modell példányosításkor megkapja az adatkezelés felületét, amelynek segítségével lehetőséget ad a pályák betöltésére (LoadGameAsync).   
• A játék pályáját a GameTable felsorolási típuson át kezeljük.

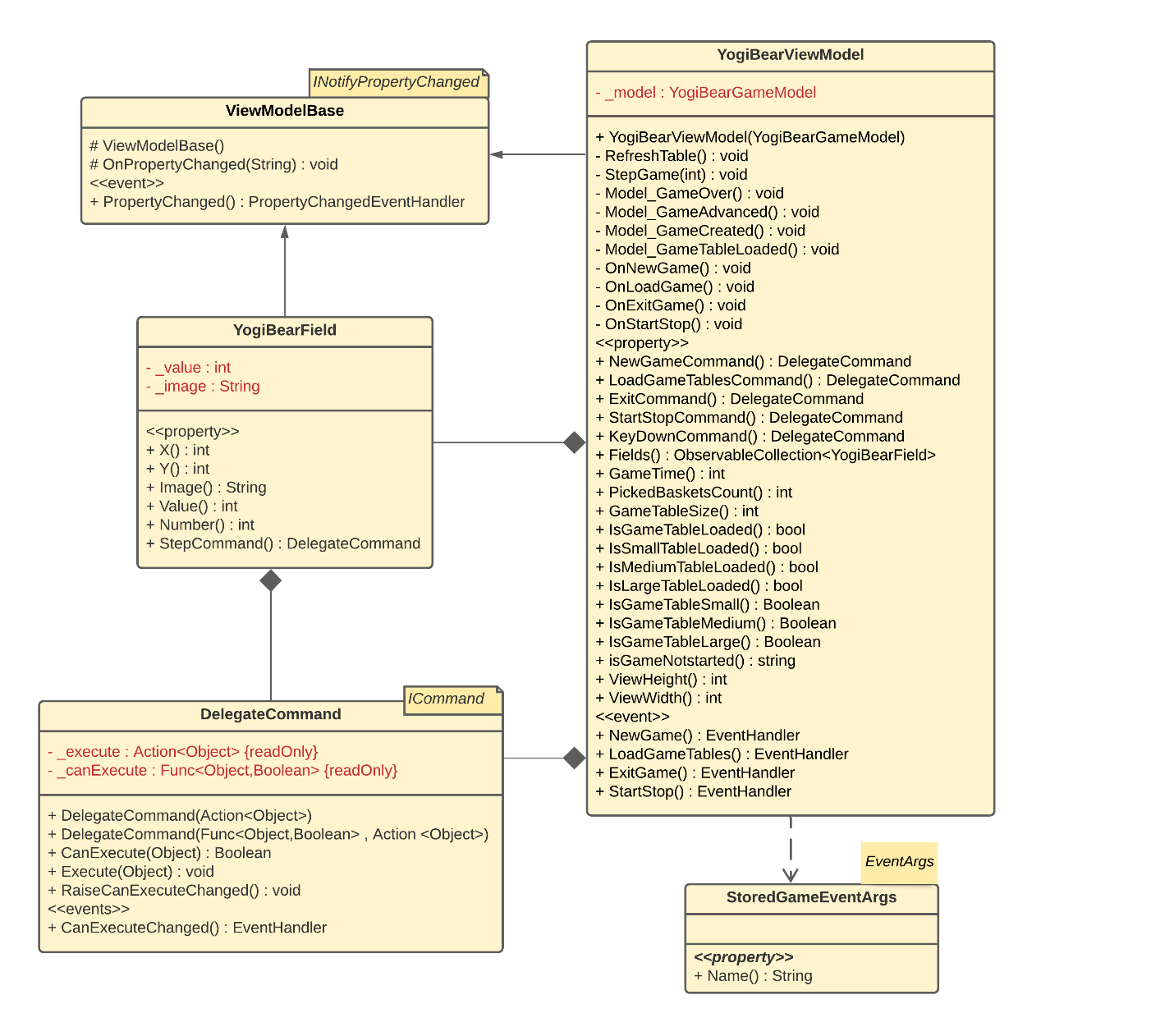
Diagram

Description automatically generated

1. **Ábra: A Model csomag osztálydiagramja**

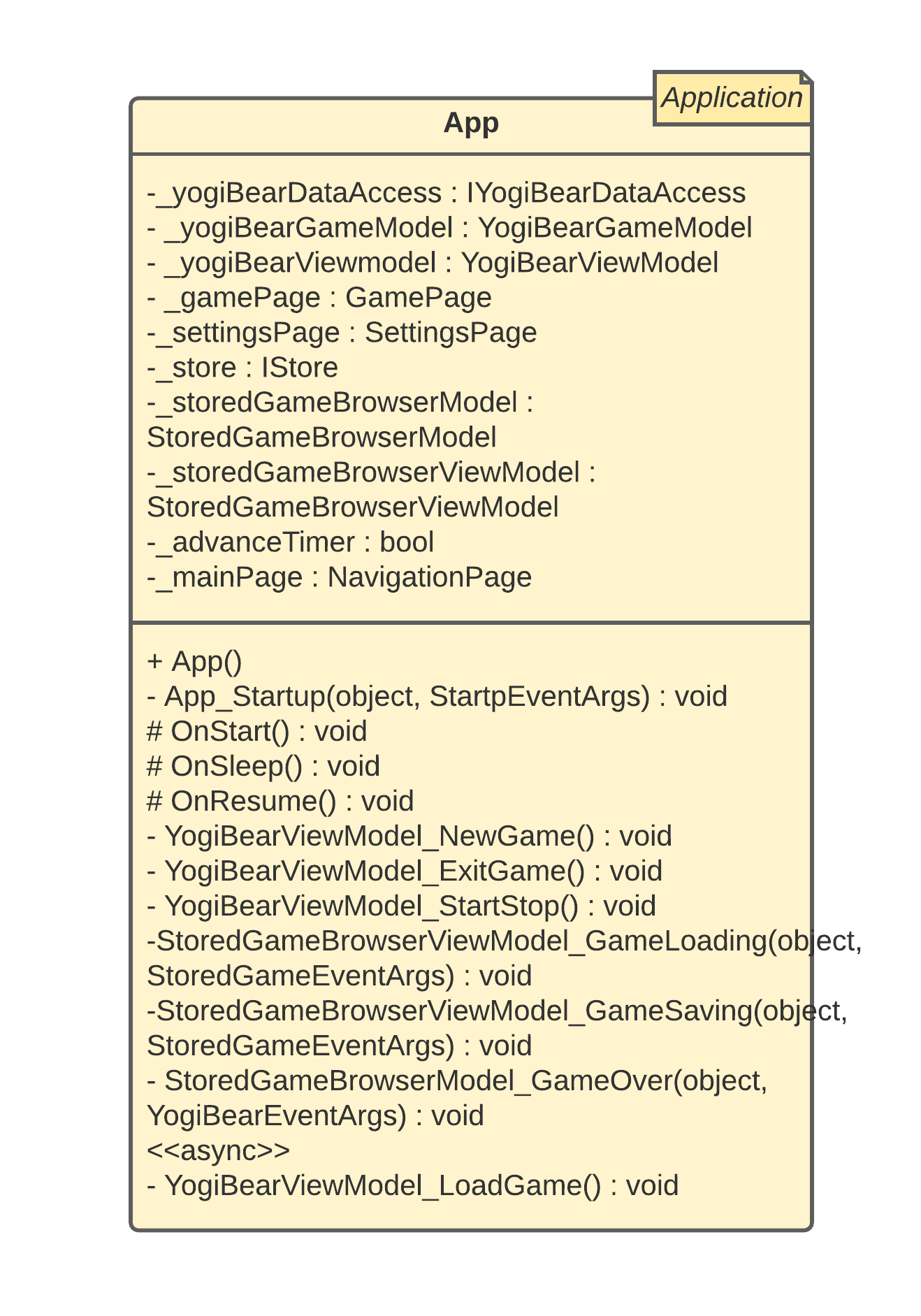
* Nézetmodell (5.ábra):

• A nézetmodell megvalósításához felhasználunk egy általános utasítás (DelegateCommand), valamint egy ős változásjelző (ViewModelBase) osztályt.  
• A nézetmodell feladatait a SudokuViewModel osztály látja el, amely parancsokat biztosít az új játék kezdéséhez, játéktáblák betöltéséhez, játékidő megállításához/elindításához, 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 játéktábla méretét szabályozza. Direkt nem avatkozik a játék futtatásába.  
• A játékmező számára egy külön mezőt biztosítunk (YogiBearField), amely eltárolja a pozíciót, a megjelenítendő kép útvonalát, valamint a lépés parancsát (StepCommand). A mezőket egy felügyelt gyűjteménybe helyezzük a nézetmodellben (Fields).



1. **Ábra: A nézetmodell osztálydiagramja**

* Nézet:
* A nézetet navigációs lapok segítségével építjük fel.
* A GamePage osztály tartalmazza a játéktáblát, amelyet egy FlowListView segítségével valósítunk meg (ez egy külső komponens), amelyben ImageButton elemeket helyezünk el.
* A SettingsPage osztály tartalmazza a Switch példányokat a tábla méretének kiválasztására.
* Környezet (6.ábra):
* Az App osztály feladata az alkalmazás vezérlése, a rétegek példányosítása és az események feldolgozása.
* Kezeljük az alkalmazás életciklust, így felfüggesztéskor (OnSleep)  
  elmentjük az aktuális játékállást (SuspendedGame), folytatáskor  
  (OnResume) és újraindításkor (OnStart) pedig folytatjuk, amennyiben  
  történt mentés.

****

1. **Ábra: A vezérlés osztálydiagramja**

**Tesztelés:**

* A modell funkcionalitása egységtesztek segítségével lett ellenőrizve a YogiBearGameModelTest osztályban.
* Az alábbi tesztesetek kerültek megvalósításra:

• YogiBearGameModelNewGameSmallTest,   
YogiBearGameModelNewGameMediumTest,   
YogiBearGameModelNewGameLargeTest: Új játék indítása, az üres mezők számának ellenőrzése, Maci Laci pozíciójának ellenőrzése   
kitöltése, valamint az idő és pálya ellenőrzése a kiválasztott pálya függvényében.   
• YogiBearGameModelStepTest: Játékbeli lépés hatásainak ellenőrzése, játék megkezdése előtt, valamint után. Több lépés végrehajtása a táblán, pozíciók ellenőrzése (ki tud-e lépni a tábláról, rá tud-e lépni az akadályra).   
• YogiBearGameModelAdcanceTimeTest: A játékbeli idő kezelésének ellenőrzése, beleértve, ha a játék éppen áll.   
• YogiBearGameModelLoadTest: A játék egy pályájának betöltésének tesztelése mockolt perzisztencia réteggel.