**1. Beadandó feladat dokumentáció**

**Készítette:**

Trefiman Viktor Ádám

E-mail: dvzcbt@inf.elte.hu

**Feladat:** Snake

Készítsük programot, amellyel a klasszikus kígyó játékot játszhatjuk.

Adott egy 𝑛 × 𝑛 elemből álló játékpálya, amelyben akadályok (falak) találhatóak. A játékos egy kezdetben 5 hosszú kígyóval indul a képernyő közepén, amely vízszintesen, illetve függőlegesen halad rögzített időközönként a legutoljára beállított irányba. A kígyóval elfordulhatunk balra, illetve jobbra. A pályán véletlenszerű pozícióban mindig megjelenik egy tojás, amelyet a kígyóval meg kell etetni. Minden etetéssel eggyel nagyobb lesz a kígyó. A játék célja, hogy a kígyó minél tovább elkerülje az ütközést az akadályokkal, a pálya szélével, illetve saját magával.

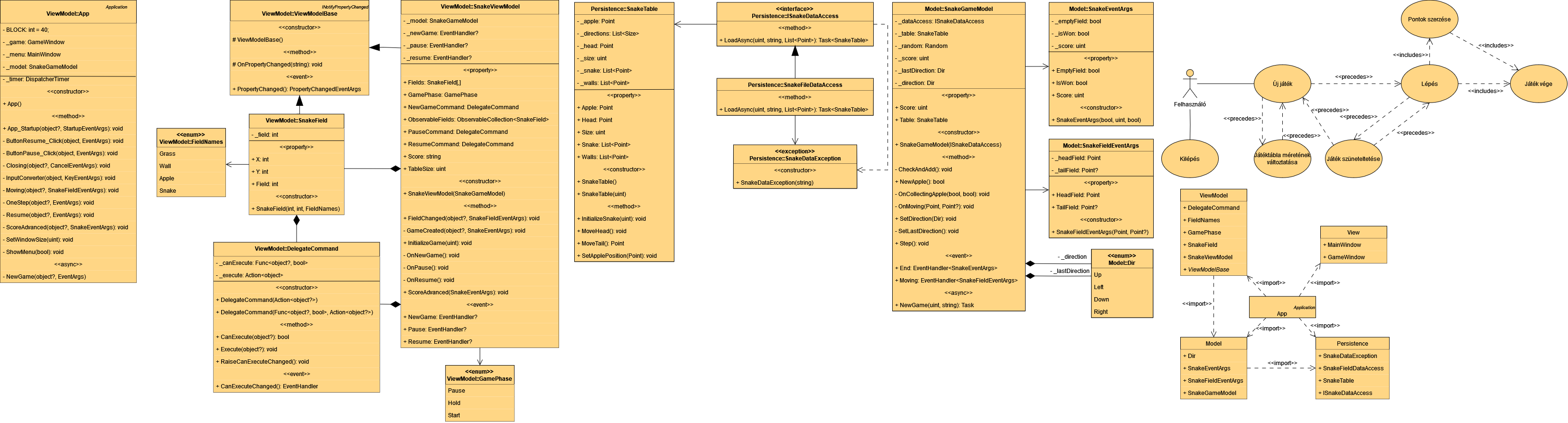
A pályák méretét, illetve felépítését (falak helyzete) tároljuk fájlban. A program legalább 3 különböző méretű pályát tartalmazzon.

A program biztosítson lehetőséget új játék kezdésére a pálya kiválasztásával, valamint játék szüneteltetésére (ekkor nem telik az idő, és nem mozog a kígyó).

Továbbá ismerje fel, ha vége a játéknak. Ekkor jelenítse meg, hány tojást sikerült elfogyasztania a játékosnak.

**Elemzés:**

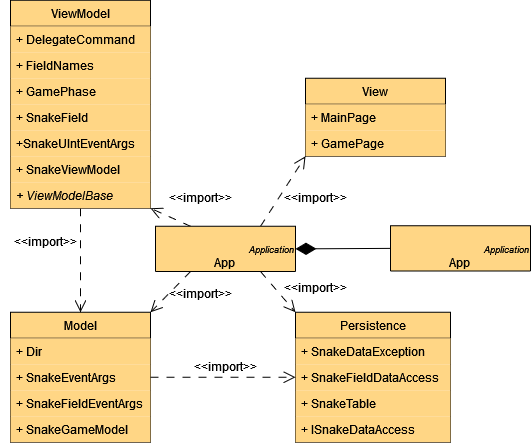
* A játékot egy 10 és 20 közötti egységes oldalhosszú pályán játszhatjuk, amit a menüben elhelyezett csúszkán állíthatunk be.
* A feladatot .NET MAUI alkalmazásként, elsődlegesen Windows és Android platformon valósítjuk meg. Az alkalmazás négy lapból fog állni. Az alkalmazás portré tájolást támogat.
* A játék két képernyőn fog megjelenni.
* Az második képernyő (Játék) tartalmazza a játéktáblát, a játék állását (megevett alma) a lap alján.
* Az első képernyőn van lehetőség új játékot indítani, valamint a játéknehézség állítására (három gombbal).
* Az ablakban elhelyezünk egy menüt a következő gombokkal: Resume (folytatás) és New Game (új játék). Az ablak alján megjelenítünk egy feliratot (Score:) és mellette egy számot, amely a pontok számát jelzi.
* A játéktáblát egy négyszögletű feliratpanelekből álló rács reprezentálja. A kígyót a sötétzöld mezők, a falakat szürke mezők, míg az almát piros mező reprezentálja. A világos zöld mezők üresek.
* A játék indításakor megjelenik a Pause (Pillanat állj) gomb, amivel előhozható a menü.
* A kígyó a tábla alatti négy gomb egyikének megnyomása után mozgásba lendül. Irányt változtatni csak az eddigi haladási iránynak merőleges irányokban tehet. Ha az alma mezőjére ér, megnövekszik egyel a hossza.
* Ha falnak vagy önmagának ütközik, esetleg kimenne a pályáról, a játék véget ér és a játékos veszít. Ha nem marad üres mező már a pályán, a játékos nyer.
* A játék végeztével a menü tetején lévő felirat közli a játékossal az eredményt.
* A felhasználói esetek az 1. ábrán láthatóak.



**1. ábra: Felhasználói esetek diagramja**

**Tervezés:**

* Programszerkezet (2. ábra):
* 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 programot MVVM architektúrában valósítjuk meg, ennek megfelelően
* View, Model, ViewModel és Persistence névtereket valósítunk meg az alkalmazáson belül.
* A megvalósításból külön építjük fel a játék, illetve a betöltés és mentés funkciót, valamennyi rétegben.
* A program vezérlését az alkalmazás osztály (App) végzi, amely példányosítja a modellt, a nézetmodell és a nézetet, biztosítja a kommunikációt, valamint felügyeli az adatkezelést.

****

**2. ábra: Az alkalmazás csomagdiagramja**

* Perzisztencia (3. ábra):
* Az adatkezelés feladata a Snake táblával kapcsolatos információk tárolása, valamint a betöltés/mentés biztosítása.
* A **SnakeTable** osztály egy érvényes Snake táblához tartozó falakat (**\_walls**), a kígyó testének pontjait (**\_snake, \_head**), illetve az alma (**\_apple**) pozícióját biztosítja (mindig ellenőrizve a beállított értékeket)
* A hosszú távú adattárolás lehetőségeit az **ISnakeDataAccess** interfész adja meg, amely lehetőséget ad a tábla betöltésére (**LoadAsync**). A műveletet hatékonysági okokból aszinkron módon valósítjuk meg.
* Az interfészt szöveges fájl alapú adatkezelésre a **SnakeFileDataAccess** osztály valósítja meg. A fájlkezelés során fellépő hibákat a

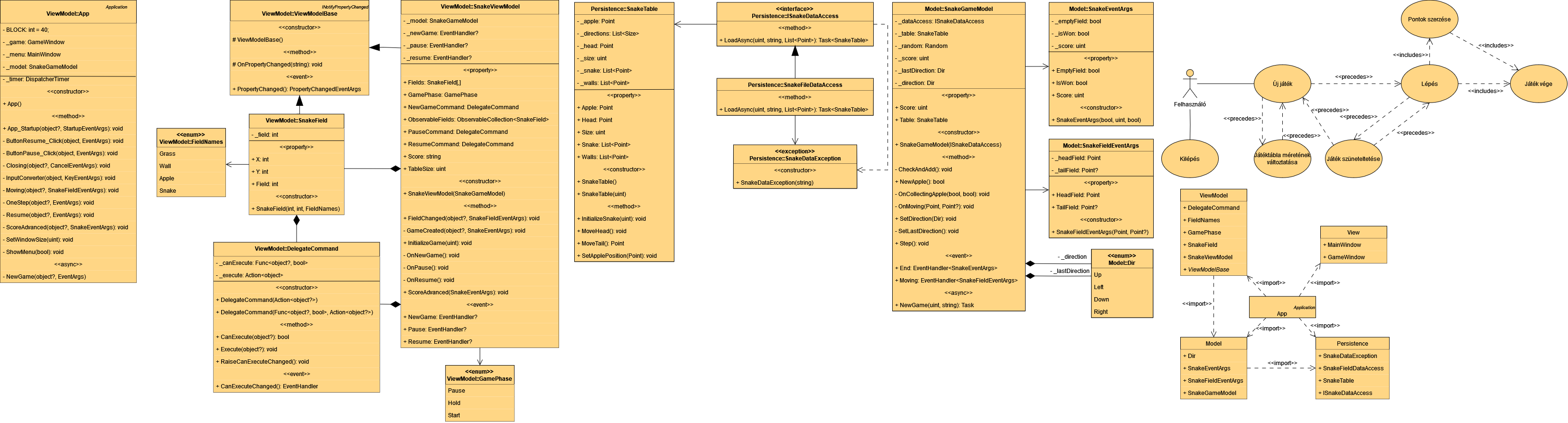
**SnakeDataException** kivétel jelzi.

* A program az adatokat szöveges fájlként tudja eltárolni, amelyeket egy

megadott könyvtárban (\_directory) helyez el. Ez majd az alkalmazás

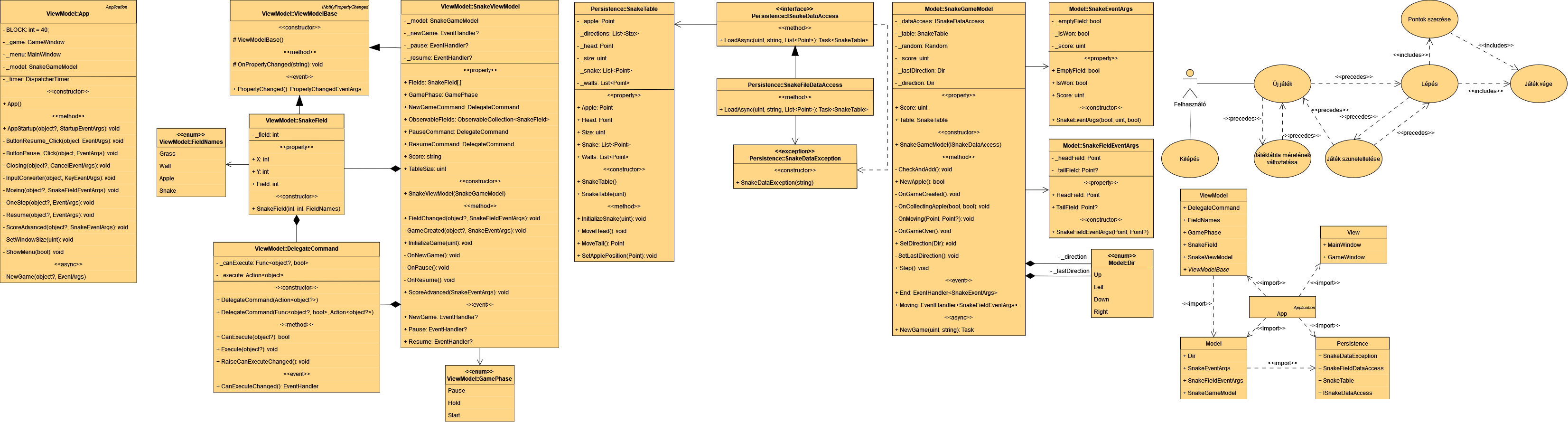
platformfüggő saját adatkönyvtára lesz.

* A fájl minden sorának az első kettőspontig eső része megadja a tábla méretét. A sor többi része a falak koordinátáit tartalmazza, pontosvesszővel elválasztva.

****

**3. ábra: A perzisztencia osztálydiagramja**

* Modell (4. ábra):
* A modell lényegi részét a **SnakeGameModel** osztály valósítja meg, amely szabályozza a kígyó tevékenységeit, valamint a játék egyéb paramétereit, úgymint az irány (**\_direction**) és a pontok (**\_score**). A típus lehetőséget ad új játék kezdésére (**NewGame**), valamint lépésre (**Step**).
* Az egy-egy lépésről a **Moving** esemény tájékoztat. Az esemény argumentuma (**SnakeFieldEventArgs**) tárolja a megváltozott mező pozícióját.
* A játékállapot megváltozásáról (pontszám növekedése) az **OnCollectingApple** esemény, míg a játék végéről az **OnGameOver** esemény tájékoztat. Az események argumentuma (**SnakeEventArgs**) tárolja a győzelem állapotát, a pontok, valamint az üres mezők számát.



**4. ábra: A modell osztálydiagramja**

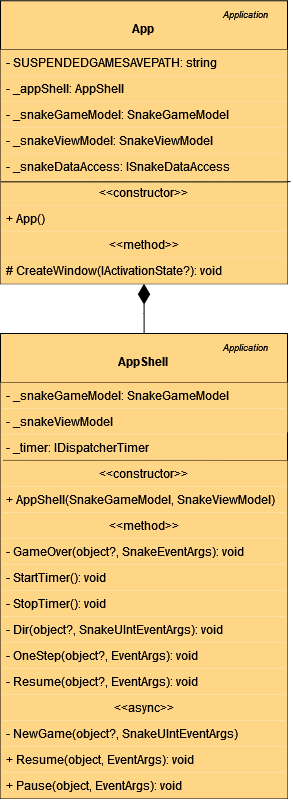
* Nézetmodell (5. ábra):
* A nézetmodell megvalósításához felhasználunk egy általános utasítás **(DelegateCommand)**, valamint egy ős változásjelző **(ViewModelBase)** osztályt.
* A nézetmodell feladatait a **SnakeViewModel** osztály látja el, amely parancsokat biztosít az új játék kezdéséhez, játék betöltéséhez, 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 játéknehézséget szabályozza. Direkt nem avatkozik a játék futtatásába.
* A játékmező számára egy külön mezőt biztosítunk **(SnakeField)**, amely eltárolja a pozíciót és a mező értékét. A mezőket egy felügyelt gyűjteménybe helyezzük a nézetmodellbe **(ObservableFields)** és egy mátrixot is létrehozunk, ami tartalmazza az elérési útjukat a gyorsabb futás érdekében.

A képen képernyőkép, szöveg látható

Automatikusan generált leírás

**5. ábra: A modell osztálydiagramja**

* Nézet:
* A nézetet navigációs lapok segítségével építjük fel.
* A **GamePage** osztály tartalmazza a játéktáblát, amelyet egy **Grid** segítségével valósítunk meg, amelyben **Label** elemeket helyezünk el.
* A **MainPage** osztály tartalmazza az új játék gombot és a vissza gombot.
* Vezérlés (6. ábra):
* 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.



**5. ábra: A modell osztálydiagramja**

**Tesztelés:**

A modell funkcionalitása egységtesztek segítségével lett ellenőrizve az úgynevezett **SnakeGameModelTest** osztályban. Az alábbi tesztesetek kerültek megvalósításra:

* **SnakeGameModelNewGameTest**,
* **SnakeGameModelStepTest**

**SnakeGameModelNewGameTest**: Új játék indítása, beolvasás, valamint a mezők helyességének ellenőrzése.

**SnakeGameModelStepTest**: Játékbeli lépés ellenőrzése minden lépés után. A lépések random generáltak. Ütköző irányú inputok, illetve a falba vagy kígyóba ütközés esetének tesztelése.