**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.

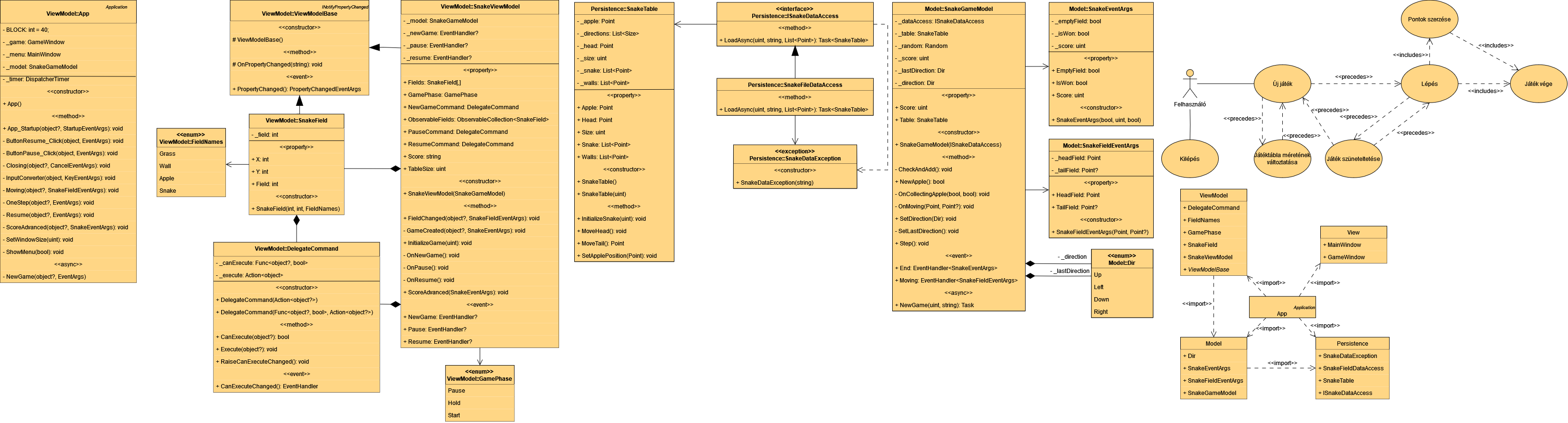
A pályák méretét, illetve felépítését (falak helyzete) tároljuk fájlban. A program legalább 3 különböző méretű pályát tartalmazzon.

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:**

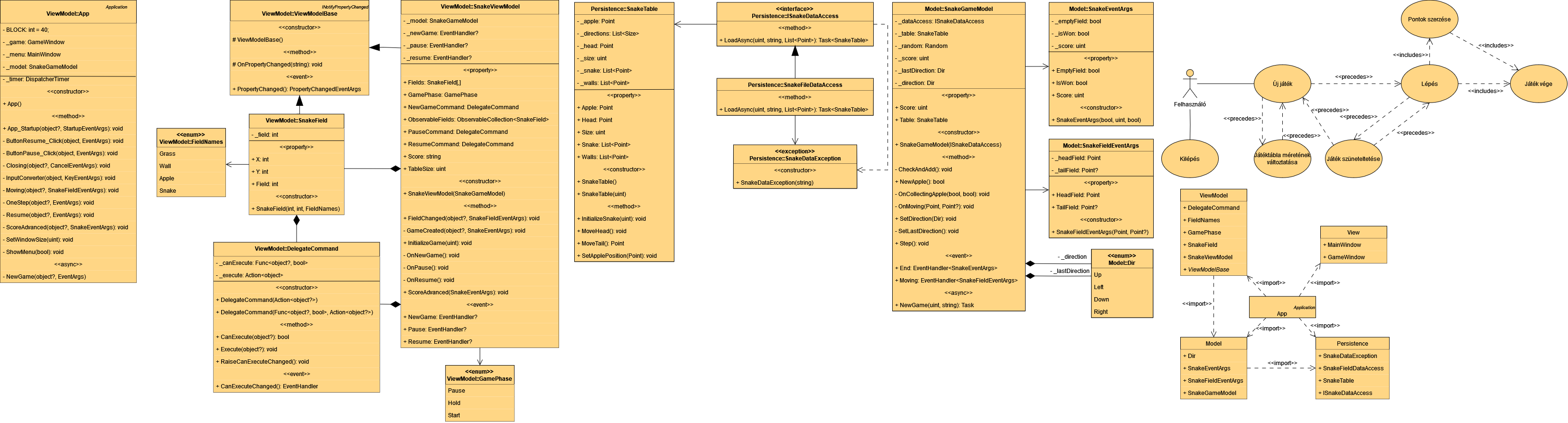
* A játékot egy 10 és 20 közötti egységes oldalhosszú pályán játszhatjuk, amit a menüben elhelyezett csúszkán állíthatunk be.
* A feladatot kétablakos asztali alkalmazásként Windows Presentation Foundation grafikus felülettel valósítjuk meg.
* Az ablakban elhelyezünk egy menüt a következő gombokkal: 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 W,A,S,D gombok 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 programot MVVM architektúrában valósítjuk meg, 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 program csomagszerkezete a 2. ábrán látható.
* A program szerkezetét két projektre osztjuk implementációs megfontolásból: a Persistence és Model csomagok a program felületfüggetlen projektjében, míg a ViewModel és View csomagok a WPF függő projektjében kapnak helyet.

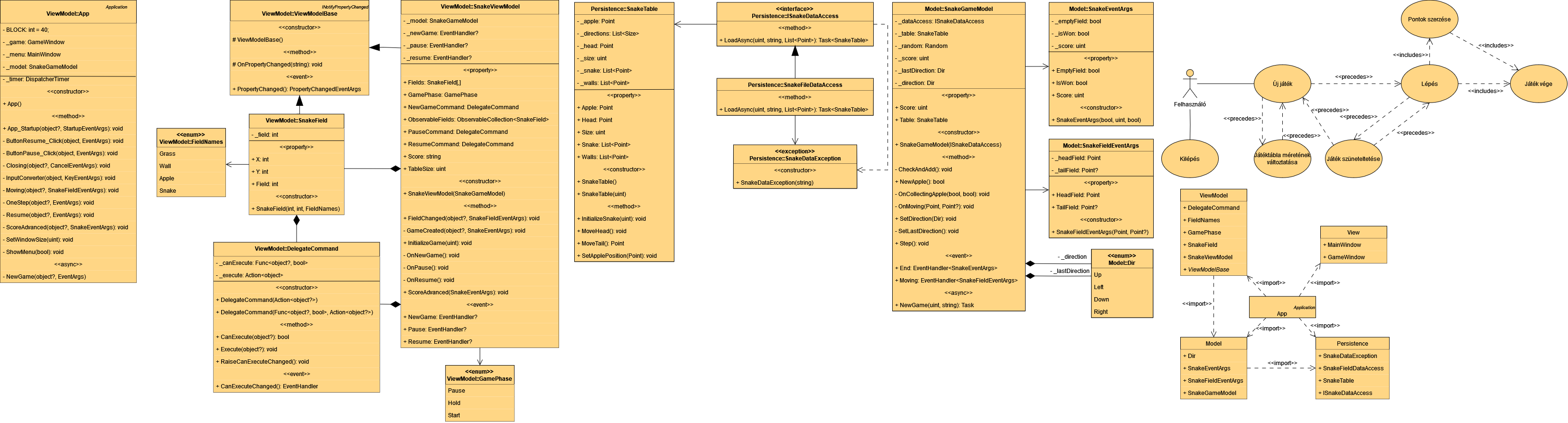
****

**2. ábra: Az alkalmazás csomagdiagramja**

* Perzisztencia (3. ábra):
* Az adatkezelés feladata a 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

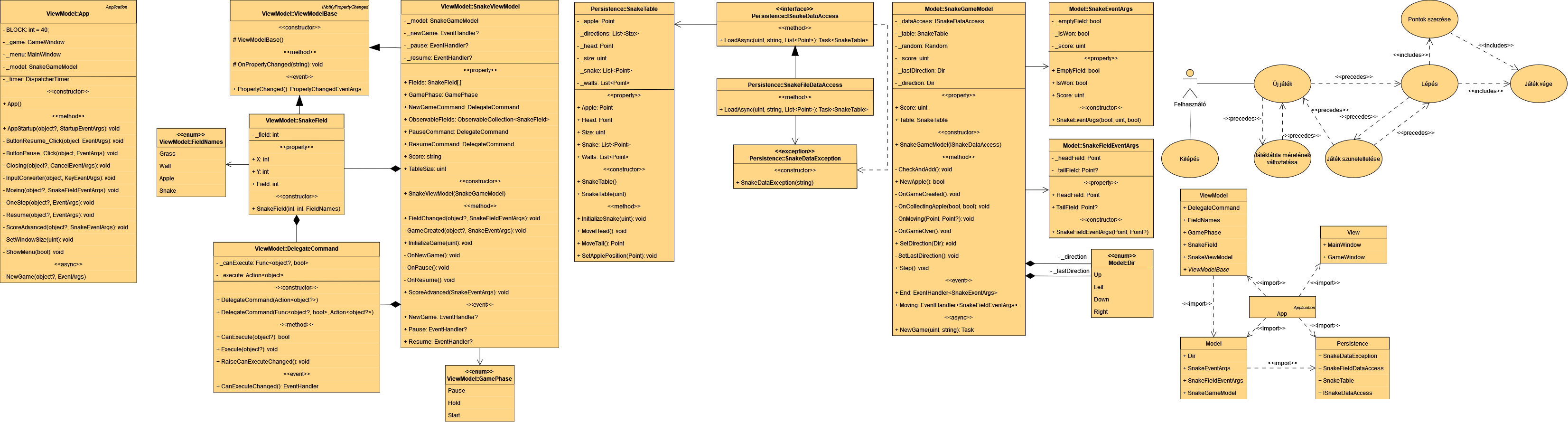
**SnakeDataException** kivétel jelzi.

* A 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**

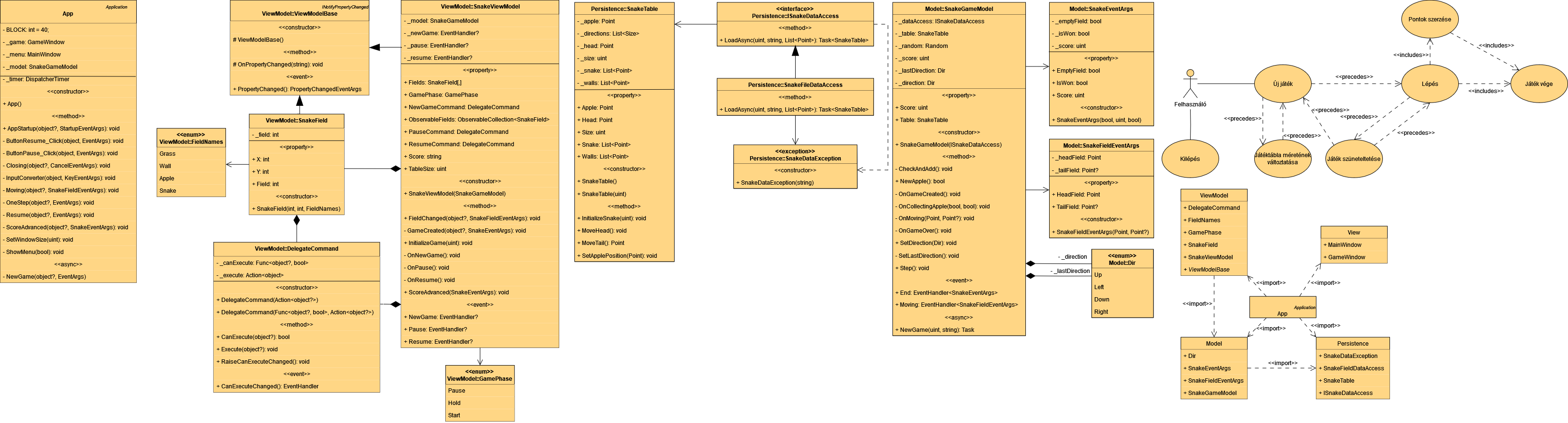
* Nézetmodell (5. ábra):
* A nézetmodell megvalósításához felhasználunk egy általános utasítás **(DelegateCommand)**, valamint egy ős változásjelző **(ViewModelBase)** osztályt.
* A nézetmodell feladatait a **SnakeViewModel** osztály látja el, amely parancsokat biztosít az új játék kezdéséhez, játék 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 nézet két képernyőt tartalmaz, a **MainWindow** és a **GameWindow** osztályt. A nézet egy rácsban tárolja a játékmezőt, a gombokat, a feliratokat és a menüt. A játékmező egy **ItemsControl** vezérlő, ahol dinamikusan felépítünk egy rácsot **(UniformGrid)**, amely feliratmezőkből áll. Minden adatot adatkötéssel kapcsolunk a felülethez, továbbá azon keresztül szabályozzuk a feliratok színét is.
* A fájlnév bekérését betöltéskor, valamint a figyelmeztető üzenetek megjelenését beépített dialógusablakok segítségével végezzük.
* Környezet (6. ábra):
* Az **App** osztály feladata az egyes rétegek példányosítása **(AppStartup)**, összekötése, a nézetmodell, valamint a modell eseményeinek lekezelése, és ezáltal a játék, az adatkezelés, valamint a nézetek szabályozása.
* A játék időbeli kezelését egy időzítő végzi (**\_timer**), amelyet mindig aktiválunk játék során, illetve inaktiválunk, amennyiben a pillanat állj (**Resume**) gombot aktiváljuk.



**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.