**2. Beadandó feladat dokumentáció**

**Készítette:**

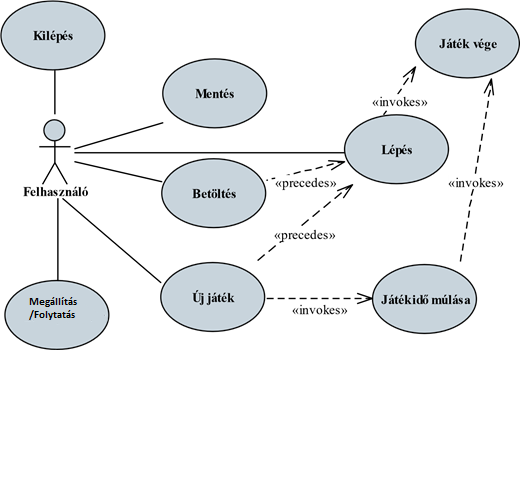
Kőrösi László E-mail: d4sxek@inf.elte.hu

**Feladat:**

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 három pályán játszhatjuk: 10x10 (5 fal),15x15 (12 fal), 20x20 (23 fal). A program indításkor nem tölt be játékot, a játékosnak magának kell elindítania/betöltenie.
* A feladatot egyablakos asztali alkalmazásként Windows Forms grafikus felülettel valósítjuk meg.
* Az ablakban elhelyezünk egy menüt a következő menüpontokkal: File (Új játék, Játék betöltése, Játék mentése, Kilépés), Megállítás/folytatás.
* A játéktáblát egy x×x nyomógombokból álló rács reprezentálja. Játék indításakor 4 gomb jelenik meg, mellyel irányíthatjuk a kígyót. Nem engedjük a tiltott lépéseket, például ha felfelé ment, akkor nem indul meg magába lefelé a kígyó.
* A játék automatikusan feldob egy dialógusablakot, amikor vége a játéknak (a kígyó elfoglalta az egész táblát, vagy ütközött fallal/magával). Szintén dialógusablakokkal végezzük el a mentést, illetve betöltést, a fájlneveket a felhasználó adja meg.
* A felhasználói esetek az 1. ábrán láthatóak.



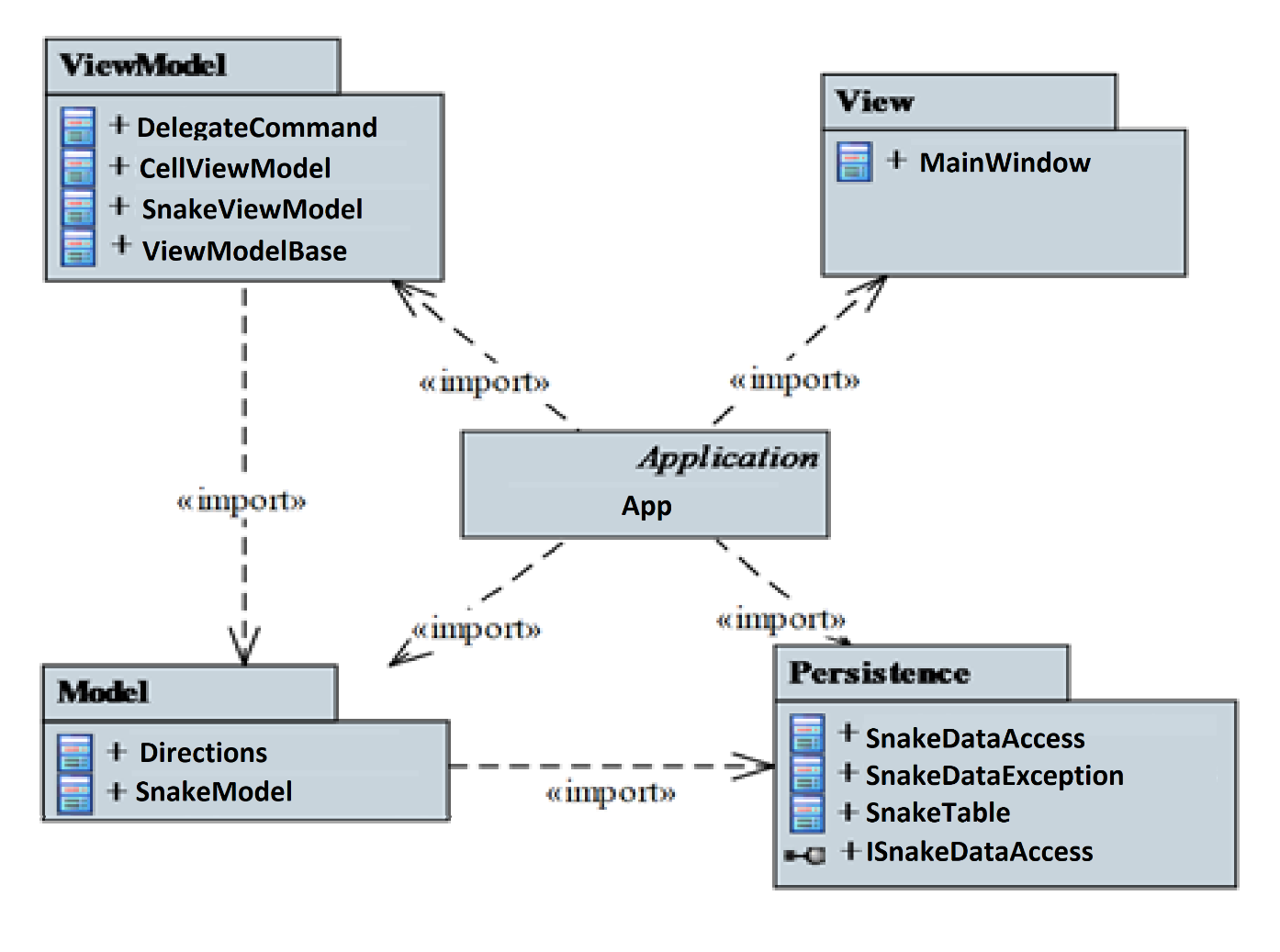
**1. ábra: Felhasználói esetek diagramja**

**Tervezés:**

* Programszerkezet:
* A programot MVVM architektúrában valósítjuk meg, ennek megfelelően

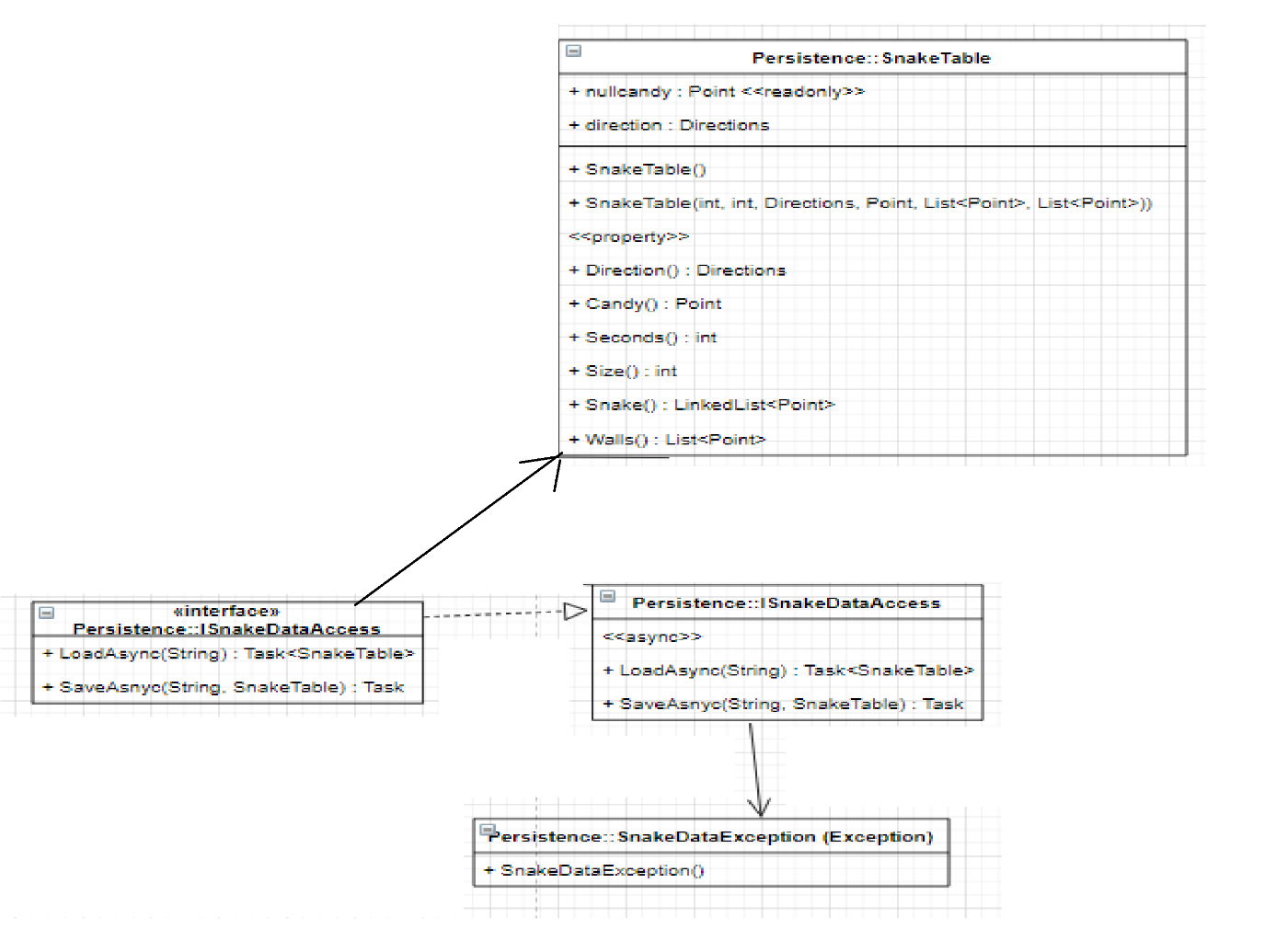
**View**, **Model**, **ViewModel** és **Persistence** névtereket valósítunk meg az alkalmazáson belül. A program környezetét az alkalmazás osztály (**App**) végzi, amely példányosítja a modellt, a nézetmodell és a nézetet, biztosítja a kommunikációt, valamint felügyeli az adatkezelést.

* A program csomagszerkezete a 2. ábrán látható.



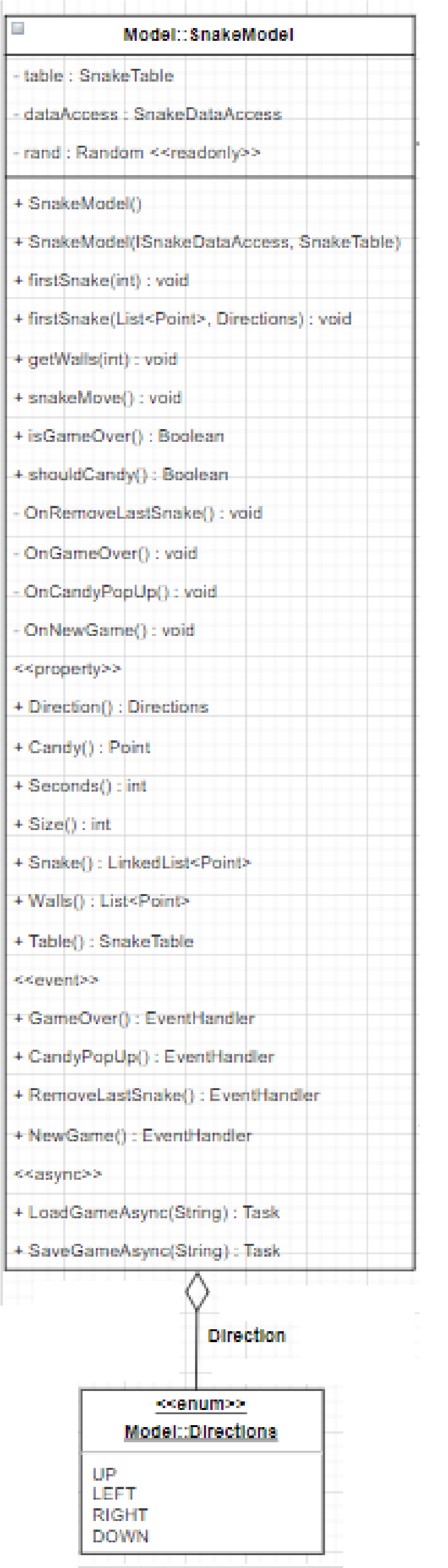
**2. ábra: Az alkalmazás csomagdiagramja**

* Perzisztencia (3. ábra):
  + Az adatkezelés feladata a Snake táblával kapcsolatos információk tárolása, valamint a betöltés/mentés biztosítása.
  + A **SnakeTable** osztály egy érvényes Snake táblát biztosít (azaz mindig ellenőrzi a beállított értékek), ahol ismerjük a kígyó helyzeteit **(Snake)**, a falak helyzeteit **(Walls)**, a cukorka helyzetét **(Candy)**, valamint az irányt **(direction)**. Snake és Walls a játék kezdetekor generált. Szabályos léptetésre (**Direction**) elvégzése.
  + A hosszú távú adattárolás lehetőségeit az **ISnakeDataAccess** interfész adja meg, amely lehetőséget ad a tábla betöltésére (LoadAsync), valamint mentésére (SaveAsync). A műveleteket hatékonysági okokból aszinkron módon valósítjuk meg.
  + Az interfészt szöveges fájl alapú adatkezelésre a **SnakeDataAccess** osztály valósítja meg. A fájlkezelés során fellépő hibákat a **SnakeDataException** kivétel jelzi.
  + A program az adatokat szöveges fájlként tudja eltárolni, melyek az **sgs** kiterjesztést kapják. Ezeket az adatokat a programban bármikor be lehet tölteni, illetve ki lehet menteni az aktuális állást.
  + A fájl egy sorból áll. Megadja a tábla méretét, a cukorka helyzetét, az irányt, az eltelt másodperceket, majd a kígyó helyzetét.



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

* Modell (4. ábra):
  + A modell lényegi részét a **SnakeModel** osztály valósítja meg, amely szabályozza a tábla tevékenységeit (tárolja a táblát **(table)** és az adatelérést **(dataAccess)**), Snake **(snakeMove)** és a cukor generálását **(shouldCandy)**. A típus lehetőséget ad új játék kezdésére **(OnNewGame)**, valamint játék vége vizsgálatára **(isGameOver)**. Új játéknál a három méret közül a megfelelővel generálja a játéktáblát.
  + A játékállapot változásáról a **CandyPopUp, RemoveLastSnake, NewGame** esemény, míg a játék végéről a **GameOver** esemény tájékoztat. Az események argumentuma (**EventArgs**) mely mindig üres.
  + A modell példányosításkor megkapja az adatkezelés felületét, amelynek segítségével lehetőséget ad betöltésre (**LoadGameAsync**) és mentésre (**SaveGameAsync**).



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

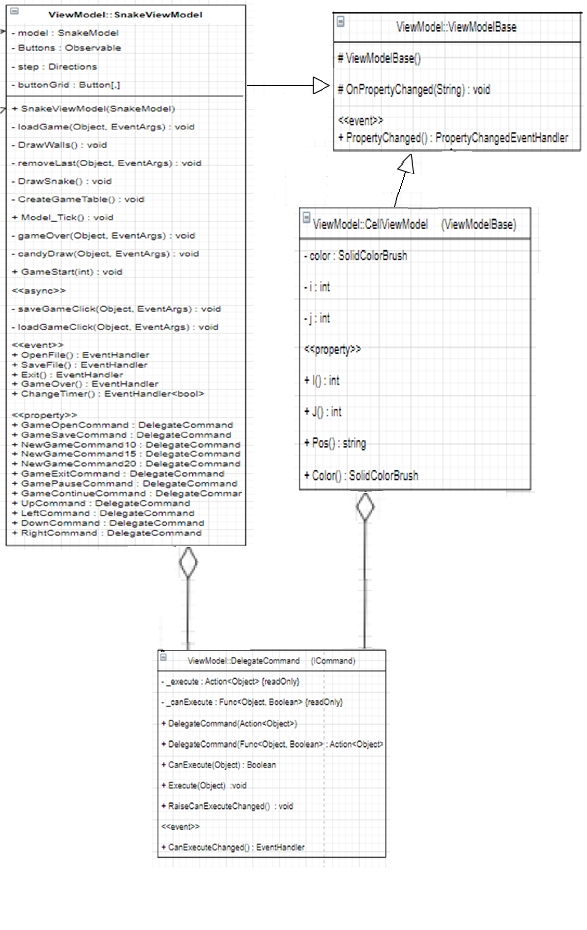
* Nézetmodell (5. ábra):
* A nézetmodell megvalósításához felhasználunk egy általános utasítás

(**DelegateCommand**), valamint egy ős változásjelző (**ViewModelBase**) osztályt.

* A nézetmodell feladatait a **SnakeViewModel** 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. Direkt nem avatkozik a játék futtatásába.
* A játékmező számára egy külön mezőt biztosítunk (**buttonGrid**), amely eltárolja a pozíciót és a mező színét. A mezőket egy felügyelt gyűjteménybe helyezzük a nézetmodellbe (**Buttons**).

**5. ábra: A nézetmodell osztálydiagramja**

* 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 felépítünk egy rácsot (**UniformGrid**), amely gombokból áll. Minden adatot adatkötéssel kapcsolunk a felülethez, továbbá azon keresztül szabályozzuk a gombok színét is.
* A fájlnév bekérését betöltéskor és mentéskor, valamint a figyelmeztető üzenetek megjelenését beépített dialógusablakok segítségével végezzük.
* Környezet (5. ábra):
* Az **App** osztály feladata az egyes rétegek példányosítása (**App\_Startup**), összekötése, a nézetmodell, valamint a modell eseményeinek lekezelése, és ezáltal a játék, az adatkezelés, valamint a nézetek szabályozása.
* A játék léptetéséhez tárol egy időzítőt is (**timer**), amelynek állítását is szabályozza az egyes funkciók hatására.



**5. ábra: A vezérlés osztálydiagramja**

**Tesztelés:**

* A modell funkcionalitása egységtesztek segítségével lett ellenőrizve a **SnakelTest** osztályban.
* Az alábbi tesztesetek kerültek megvalósításra:
* **TestNewGame10**,

**TestNewGame15**,

**TestNewGame20**: Új játék indítása, a mezők kitöltése, valamint a kígyó és falak ellőrzése.

* **TestMove**: Játékbeli lépés hatásainak ellenőrzése.
* **TestInvalidMove**: Nem megfelelő lépés ellenőrzése.
* **TestCherryGame**: A cukorka felvételének tesztelése.
* **TestLoadGame**: A játék modell betöltésének tesztelése mockolt perzisztencia réteggel.
* **TestSaveGame**: A játék modell mentésének tesztelése mockolt perzisztencia réteggel.