3.beadandó feladat dokumentáció

Készítette:

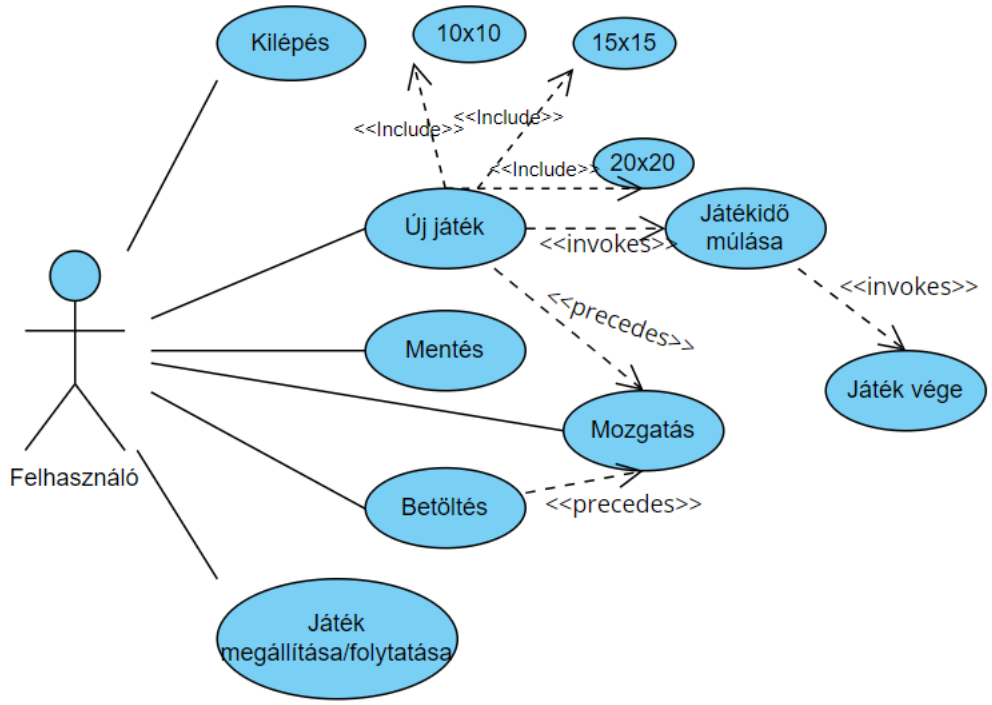
Amamou Martin
E-mail: amamoumartin@gmail.com

Feladat:

Készítsük programot, amellyel a klasszikus kígyó játékot játszhatjuk. Adott egy $n \times n$ 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:

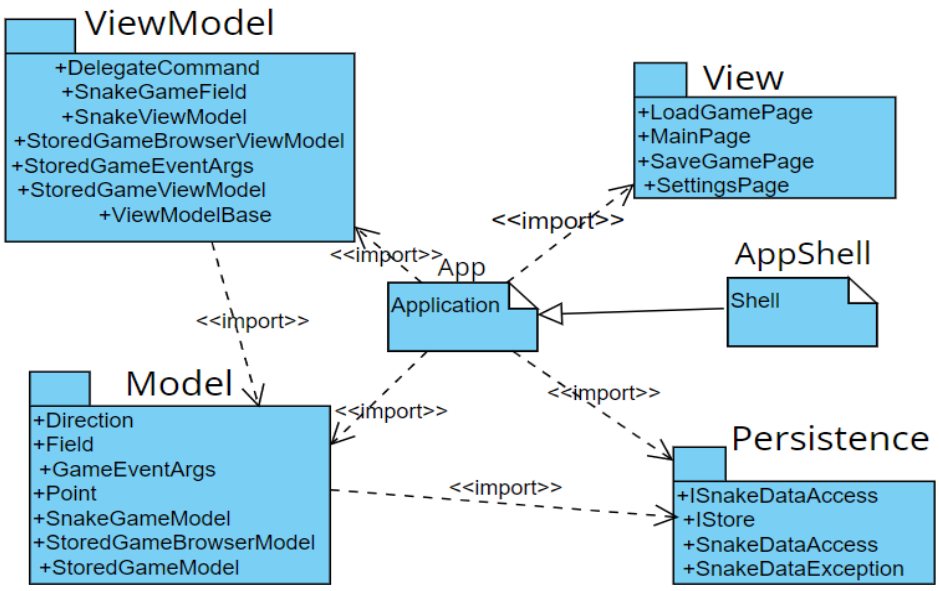
- A feladatot .NET MAUI alkalmazásként, elsődlegesen Windows és Android platformon valósítjuk meg. Az alkalmazás négy lapból fog állni. Az alkalmazás portré tájolást támogat.
- A játék négy képernyőn fog megjelenni.
    - Az első képernyő (Játék) tartalmazza a játéktáblát, az új játék, játék indításához/ szüneteltetéséhez, valamint a beállításokhoz szükséges gombokat a lap tetején. Emellett a játék állását (elfogyasztott tojások száma) és a kígyó irányításához szükséges gombokat tartalmazza a lap alján.
    - A második képernyőn van lehetőség betöltésre, illetve mentésre.
    - 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 játékot háromfajta pályamérettel játszhatjuk: 10x10(6 fal), 15x15(12 fal), 20x20(18 fal). A program indításakor a 15x15-ös pálya töltődik be.
- A program indításakor a játékmenet áll, a játékot a ▶ gombbal indíthatjuk el, illetve a ⏸ gombbal szüneteltetjük.
- A játék automatikusan feldob egy dialógusablakot, amikor vége a játéknak (falnak vagy önmagának ütközik a kígyó, vagy megtelik a tábla).
- A használati esetek az 1. ábrán láthatóak.



1. ábra:Felhasználói eset diagram

Tervezés:

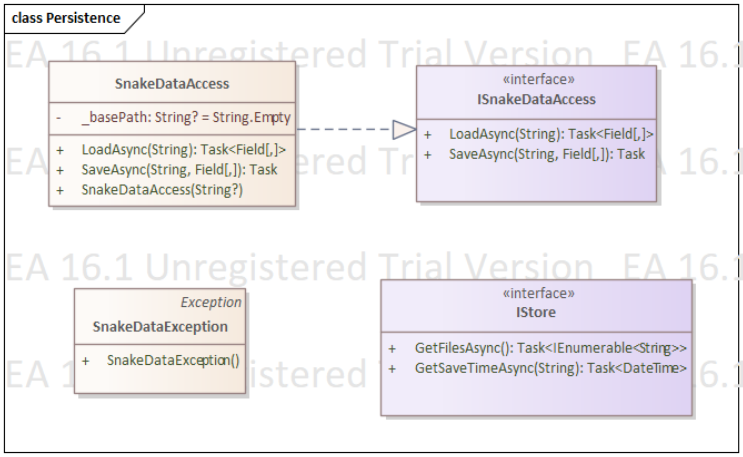
- Programszerkezet:
  - A szoftvert két projektből építjük fel: a modellt és a perzisztenciát tartalmazó osztálykönyvtárból (.NET Standard Class Library), valamint a .NET MAUI többplatformos projektből, amelyet Windows és Android operációs rendszerre is le tudunk fordítani.
  - A programot MVVM architektúrában 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. 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.
  - A program csomagdiagramja a 2. ábrán látható.



2. ábra: Az alkalmazás csomagdiagramja

Perzisztencia(3.ábra):

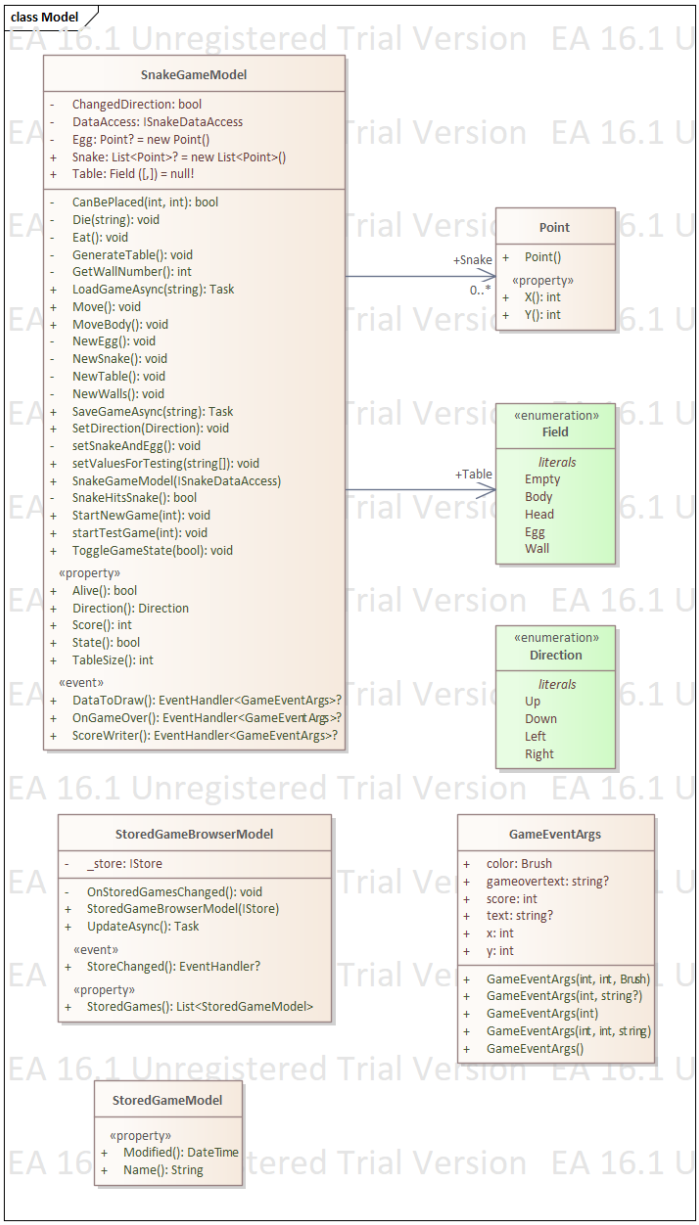
- Az adatkezelés feladata a betöltés/mentés biztosítása.
- A hosszú távú adattárolás lehetőségeit az **ISnakeDataAccess** interfész adja meg, amely lehetőséget ad a tábla betöltésére (**LoadAsync**), valamint mentésére (**SaveAsync**). A műveleteket hatékonysági okokból aszinkron módon valósítjuk meg.
- 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, amelyeket egy megadott könyvtárban (**_directory**) helyez el. Ez majd az alkalmazás platformfüggő saját adatkönyvtára lesz.
- A fájl első sora megadja a tábla méretét(**tableSize**). A fájl többi része izomorf leképezése a játéktáblának, azaz összesen **tableSize**-nyi sor következik, és minden sor **tableSize** db számot tartalmaz szóközökkel választva. Csupán 0,1,2,3,4-eket tartalmaz, a játékmező típusától függően:
  - 0- üres mező
  - 1- a kígyó teste
  - 2- a kígyó feje
  - 3- fal
  - 4- tojás



3. ábra: A Persistence csomag osztálydiagramja

Modell (4.ábra):

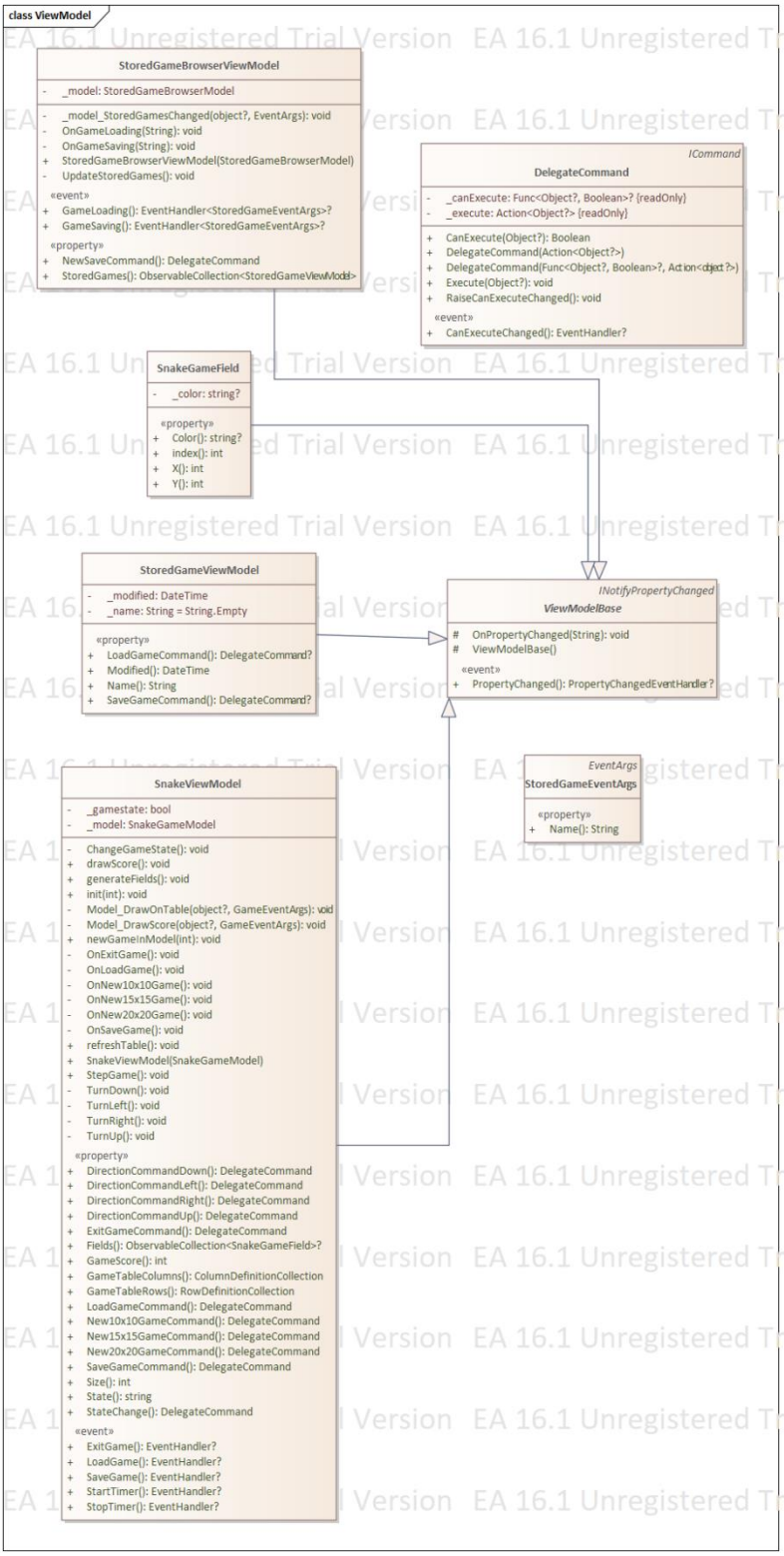
- A modell fő részét a **SnakeGameModel** valósítja meg, a tábla jelentősebb tevékenységeit kezeli, szabályozza a paramétereket: táblaméret (**TableSize**), pontszám (**Score**), irány (**Direction**), stb.
- Lehetőséget ad új játék kezdésére (**StartNewGame**)
- A modell példányosításkor megkapja az adatkezelés felületét, amelynek segítségével lehetőséget ad aszinkron módon történő betöltésre (**LoadGameAsync**) és mentésre (**SaveGameAsync**)
- Tartalmazza a játéktáblát, mely egy játékmező (**Field**) felsorolási típusból álló mátrix. • Létrehozza az elemeket (tábla, kígyó, tojás, falak) (**GenerateTable**)
- Továbbá a kígyó reprezentációját (pontokból álló lista), és folyamatait: mozgását (**MoveSnake**), étkezését (**Eat**), elpusztulását (**Die**) implementálja, emellett az ezeket biztosító ellenőrző- és segédfüggvényeket tárolja.



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 (ezáltal a nehézség kiválasztásához), továbbá 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 játék léptetését hívja meg a modellen.
- A játékmező számára egy külön mezőt biztosítunk (**SnakeGameField**), amely eltárolja a koordinátákat (x, y), a lineáris pozícióját(index), továbbá a mező színét(**Color**). A mezőket egy felügyelt gyűjteménybe helyezzük a nézetmodellbe (**Fields**)
- A tárolt játékállapotok egy-egy **StoredGameViewModel** példánnyal írhatóak le. Ezek kollekcióját nem ágyazzuk be a fő nézetmodellbe (**SnakeViewModel**), hanem a betöltéskor és mentéskor dinamikusan állítjuk elő és adjuk át a nézet számára.



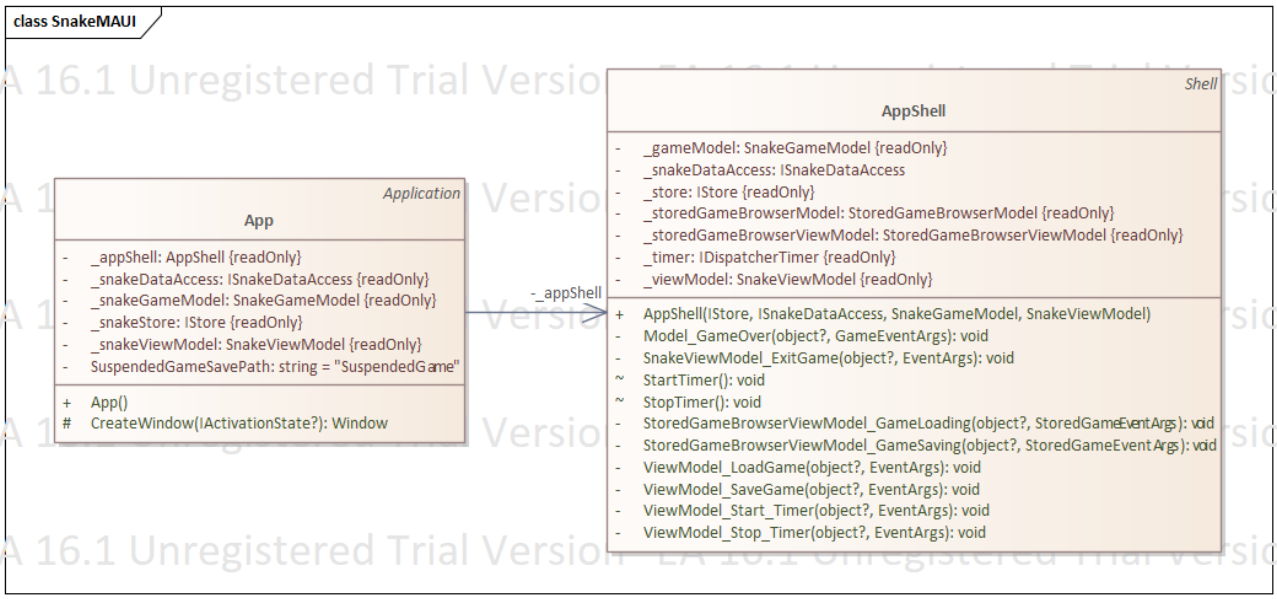
5. ábra: A nézetmodell osztálydiagramja

Nézet:

- A nézetet navigációs lapok segítségével építjük fel.
- A **MainPage** osztály tartalmazza a főoldalt,ezen belül egy játéktáblát, amelyet egy Grid segítségével valósítunk meg, amelyben Button elemeket helyezünk el. Továbbá magában foglalja az új játék indításához/ szüneteltetéséhez, valamint a beállításokhoz szükséges gombokat a lap tetején. Emellett a játék állását (elfogyasztott tojások száma) és a kígyó irányításához szükséges gombokat tartalmazza a lap alján.
- A **SettingsPage** osztály tartalmazza a betöltés, mentés gombjait.
- A **LoadPage** és a **SavePage** szolgál egy létező játékállapot betöltésére, illetve egy új mentésére.

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.



6. ábra: A vezérlés osztálydiagramja

Tesztelés:

A modell funkcionalitása egységtesztek segítségével lett leellenőrizve a **SnakeGameTest** osztályban, az alábbi tesztesetekkel:

• *TenBoardTestMethod*, *FifteenBoardTestMethod*, *TwentyBoardTestMethod* ellenőrzik a 10x10-es, 15x15ös, 20x20-as pályákon a tábla megfelelő mezőinek típusát.

• *EatTest*: azt vizsgálja, hogy a kígyó elfogyassza-e a tojást → ezáltal növekszik a pontszáma, és a testhossza

• *WallCollisionTest*: azt teszteli, hogy valóban vége lesz-e a játéknak, ha a kígyót hagyjuk falnak menni

• *BorderCollisionTest*: azt teszteli, hogy valóban vége lesz-e a játéknak, ha a kígyót hagyjuk a pálya szélének menni

• *SnakeCollisionTest*: azt teszteli, hogy valóban vége lesz-e a játéknak, ha a kígyót hagyjuk önmagának nekimenni