1. BEADANDÓ FELADAT DOKUMENTÁCIÓ

Készítette:

Losonczi Dániel
Neptun-kód: DON140
E-mail: dlosonczi1@gmail.com

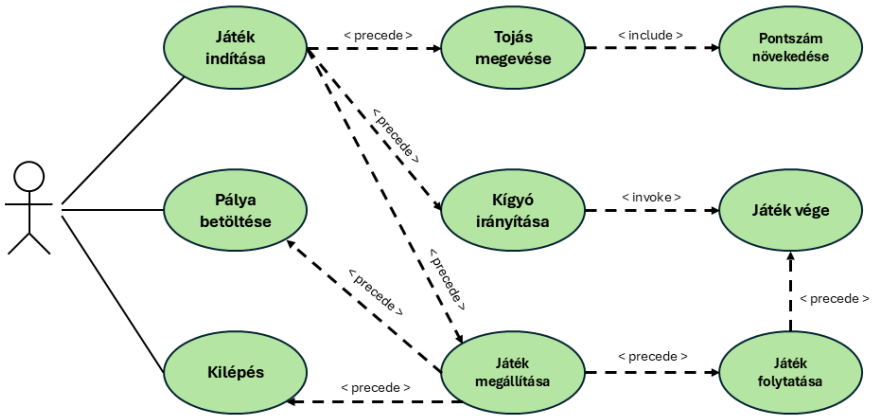
Feladat:

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 Avalonia alkalmazásként, elsődlegesen asztali alkalmazásként és Android platformon valósítjuk meg, amely egy fő nézetből fog állni. Az alkalmazás horizontális tájolást támogat.
- Az ablakban elhelyezünk egy menüt, amely a GroupBox osztállyal lesz megvalósítva és annak láthatósága fogja a teljes menü láthatóságát vezérelni. A menüben 3 gomb foglal helyet: Play (Resume), Load Map és Exit. Ezen felül megjelenítésre kerül, ha az adott játékmenet szüneteltetve van (Paused) vagy véget ért (Game Over), ekkor a megfelelő pontszám kijelzésével társulva. Ha van játék pálya betöltve, akkor azt a jobb alsó sarokban egy felirat jelzi.
- A játékos a balra nyíl (vagy a), illetve jobbra nyíl (vagy d) billenytűk segítségével forgathatja a kígyó menetirányát. Ezen felül az ESC billentyű lenyomásával szüneteltetheti a játékmenetet, illetve újból elindíthatja azt.

- A játéktáblát (pályát) egy $n \times n$ -es PictureBox-okból álló rács reprezentálja. Ezen elemek színe jelzi a különböző játék elemeket:
  - fehér: *üres*
  - piros: *fal*
  - zöld (világos): *kígyó teste*
  - zöld (sötét): *kígyó feje*
  - ködös fehér: *tojás*
- Amikor a játék véget ér, a menü jelenik meg a játéktér felett, kijelezve az aktuális pontszámot. A következő esetekben ér véget a játék:
  - nincs több hely (hova mozogni vagy tojást elhelyezni)
  - kígyó feje a fallal ütközik
  - kígyó feje annak testével ütközik
- A különböző játék pályákat a „Load Map” cimkével ellátott gombbal lehet betölteni, ekkor egy dialógus nyílik, ahol a megfelelő „.save” kiterjesztéssel rendelkező file-okat reprezentálják a különböző játék tereket.
- A felhasználói esetek a következő ábrán láthatóak:



Tervezés:

- Programszerkezet:
  - A szoftvert négy projektből építjük fel: a modellt és a perzisztenciát tartalmazó osztálykönyvtárból (.NET Standard Class Library), valamint a .NET Avalonia projektjeiből (platformfüggetlen osztálykönyvtár és platformfüggő végrehajtható projektek), amelyet így Windows és Android operációs rendszerekre is le tudunk fordítani.
  - A programot MVVM architektúrában valósíjuk 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 nézetmodellt és a nézetet, biztosítja a kommunikációt, valamint felügyeli az adatkezelést.
  - 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 kap helyet.
- Perzisztencia:
  - Az adatkezelés feladata a Snake pályával kapcsolatos információk (mérete és falak helyzete) betöltésének biztosítása.
  - A pálya beolvasás lehetőségét az **IDataAccess** interfész adja meg, amely lehetőséget biztosít egy játék pálya betöltésére a **Load** és **LoadAsync** metódusán keresztül, amely a pálya dimenzióját és a falak helyzetét adja vissza. Később látjuk majd, hogy csak egyetlen osztály valósítja meg ezen interfészt, viszont a létezésére magyarázat, hogy így más formátumú pálya fájlok betöltése is könnyen implementálható.
  - Az előbb tárgyalt interfészt a **SaveFileDataAccess** osztály implementálja és biztosít lehetőséget, hogy a „.save” formátumú fájlokból pályát olvassunk be. Rossz formátum esetén **ExtensionException** kivételt dob, illetve rossz mentés fájl esetén **FormatException**-t dob. A mobilos platformok támogatása érdekében az osztály egy **Stream** (adatfolyam) segítségével is létrehozhatjuk, illetve a hatékonyabb adatkezelés érdekében a **Load** függvény aszinkron függvényét használjuk.
  - A „.save” fájlok felépítése:
    - az elsősor tartalmazz a pálya x és y irányú méretét
    - a rákövetkező sorok pedig a falak relatív helyzetét, ahol a ’#’ karakter jelöl falat és bármely másik üres mezőt (preferált a space üres mező jelölésére)

- Modell:
  - A modell lényegi részét a **GameModel** osztály valósítja meg, amely kezeli a pálya állapotát, beleértve a kígyó mozgatását, pálya generálását, tojás elhelyezését, a játék időben történő előre léptetését, illetve a játék szakaszainak időszerűségét.
  - A **ViewModel**-lel esemény kezelőn keresztül kommunikál:
    - ResetTiles: *akkor invokálódik, ha új játékot kezdünk vagy ha épp létre jött a model objektumunk*
    - UpdateTile: *feladata, hogy a **ViewModel** felé közvetítse egyetlen játék mező típusának megváltozását (pl.: lép egyet a kígyó vagy tojást helyezünk le)*
    - StopEvent: *akkor invokálódik, ha a játék véget ér vagy a játékos szüneteltetni azt*
  - Fontos talán még kiemelni, hogy ezen osztály kezeli az időmúlását és így a kígyó léptetését egy *System.Threading.Timer* osztály példányosított objektumán keresztül.
  - A **GameModel** osztály példányosításakor nem szükséges **IDataAccess** objektumot megadnunk, ilyenkor egy üres pálya inicializálódik 15 egység mérettel.
- Nézetmodell:
  - A nézetmodell megvalósításához felhasználjuk az MVVM Toolkit csomagból elérhető általános utasítás (RelayCommand), valamint egy ős változásjelző (ObservableObject) osztályt.
  - A nézetmodell feladatait a **MainViewModel** osztály látja el, amely parancsokat biztosít a játék elindításához, pálya betöltéséhez, a játékból való kilépéshez, illetve billentyűk lenyomásának érzékeléshez. 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 megfelelő irányba forgatja a kígyót a játékos billentyű leütéseiből következően.
  - A pálya elemeire egy külön **ViewModelBase**-ből származó osztályt hozunk létre (**RectangleViewModel**), amely az elemek megjelenítési tulajdonságait tárolja. A téglalapokat egy felügyelt gyűjteménybe helyezzük a nézetmodellben (**Tiles**).
- Nézet:
  - A nézetet ablakok, egyedi vezérlők és dialógusablakok használatával valósítjuk meg.
  - A **MainView** osztály, mint **UserControl** leszármazott tartalmazza a játéktáblát, amelyet egy **Grid** segítségével valósítunk meg, amelyben **Rectangle** elemeket helyezünk el.

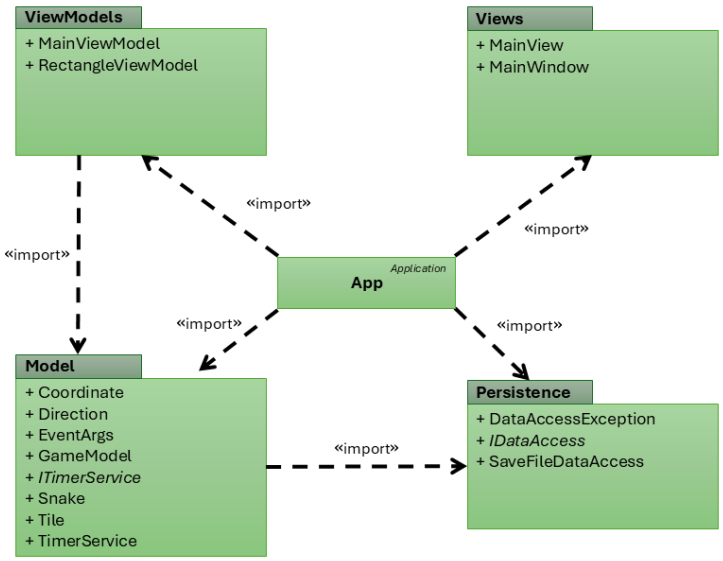
    - A **MainWindow** ablakba egyszerűen a **MainView** vezérlőt ágyazzuk be. Ilyen módon a felület asztali alkalmazásokban ablakos alkalmazásként, mobil platformon pedig lapként is megjeleníthető.
    - Betöltéshez a StorageProvider osztály által az **OpenFilePickerAsync** metódusokon keresztül biztosított, operációs rendszer specifikus dialógus ablakokat (lapokat) használjuk.
    - Felugró üzenetek megjelenítéséhez a **MessageBox.Avalonia** NuGet csomagot használjuk.
- Környezet:
    - Az **App** osztály feladata az egyes rétegek példányosítása, összekötése, a nézetmodell, valamint a modell eseményeinek lekezelése, és ezáltal a játék, az adatkezelés, valamint a nézetek szabályozása.
    - Ellátja a felugró ablakok megjelenítését, azok kezelését, melyre a nézetmodelltől kap utasítást eseménykezelőkön keresztül.
    - A **OnFrameworkInitializationCompleted** metódus felüldefiniálásával kezeljük a nézet platform specifikus megjelenítését, továbbá az alkalmazás életciklusát a megfelelő eseményekre történő feliratkozással.

Tesztelés:

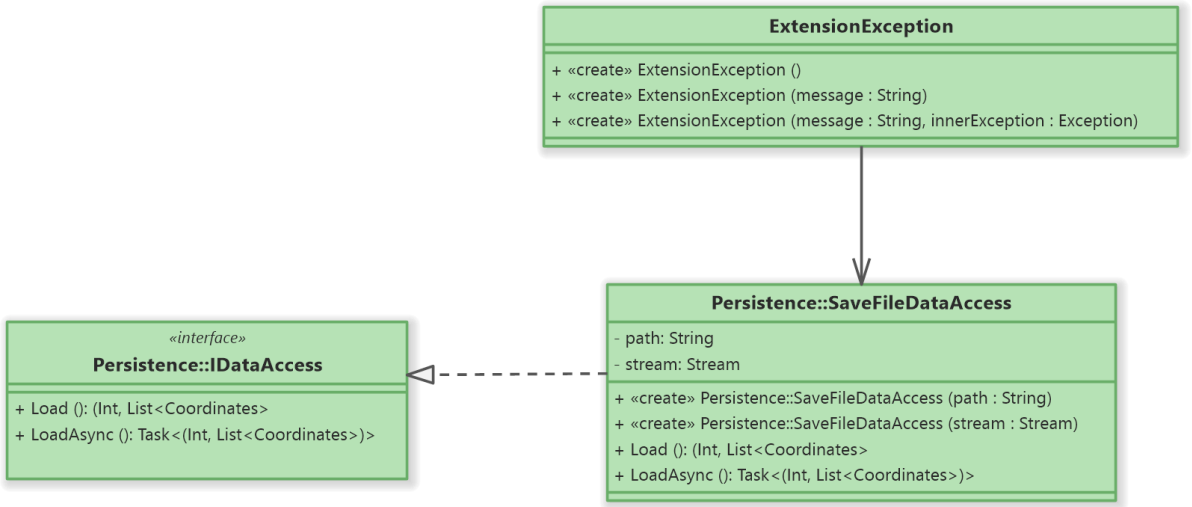
- A modell funkcionalitása egységtesztek segítségével lett ellenőrizve a **SnakeModelTest** osztályban.
- Az alábbi tesztesetek kerültek megvalósításra:
    - **ModelInitializeTest**: amely leellenőrzi, hogy megfelelő módon lett-e inicializálva az újonnan létrehozott modell objektum
    - **TurnLeftTest, TurnRightTest**: melyek letesztelik, hogy ha a modellben balra (illetve jobbra) szeretnénk forgatni a kígyót, akkor az a **TimerTick** bekövetkezte után tényleg hatással van-e a kígyó irányára
    - **IncreaseScoreTest**: azt vizsgálja, hogy nő-e a pont, miután a kígyó megeszik egy tojást
    - **SelfCollisionTest**: amely leteszteli, hogy ha kígyó sajátmagával ütközik, akkor véget ér-e a játék
    - **SaturatedMapTest**: azt vizsgálja, hogy ha már nincs üres terület a pályán, véget ér-e a játék
    - **WallCollisionTest**: amely leellenőrzi, hogy ha fallal ütközik a kígyó, akkor véget ér-e a játék
    - **WrongFormatTest**: azt teszteli, hogy hibát vált-e ki rossz formátumú fájl beolvasásaű

- **Vertical/HorizontalTorusTest**: azt vizsgálja, hogy a kígyó „körbetekeredik-e” a pályán (egyik oldalt ki, másikon visszajön-e)

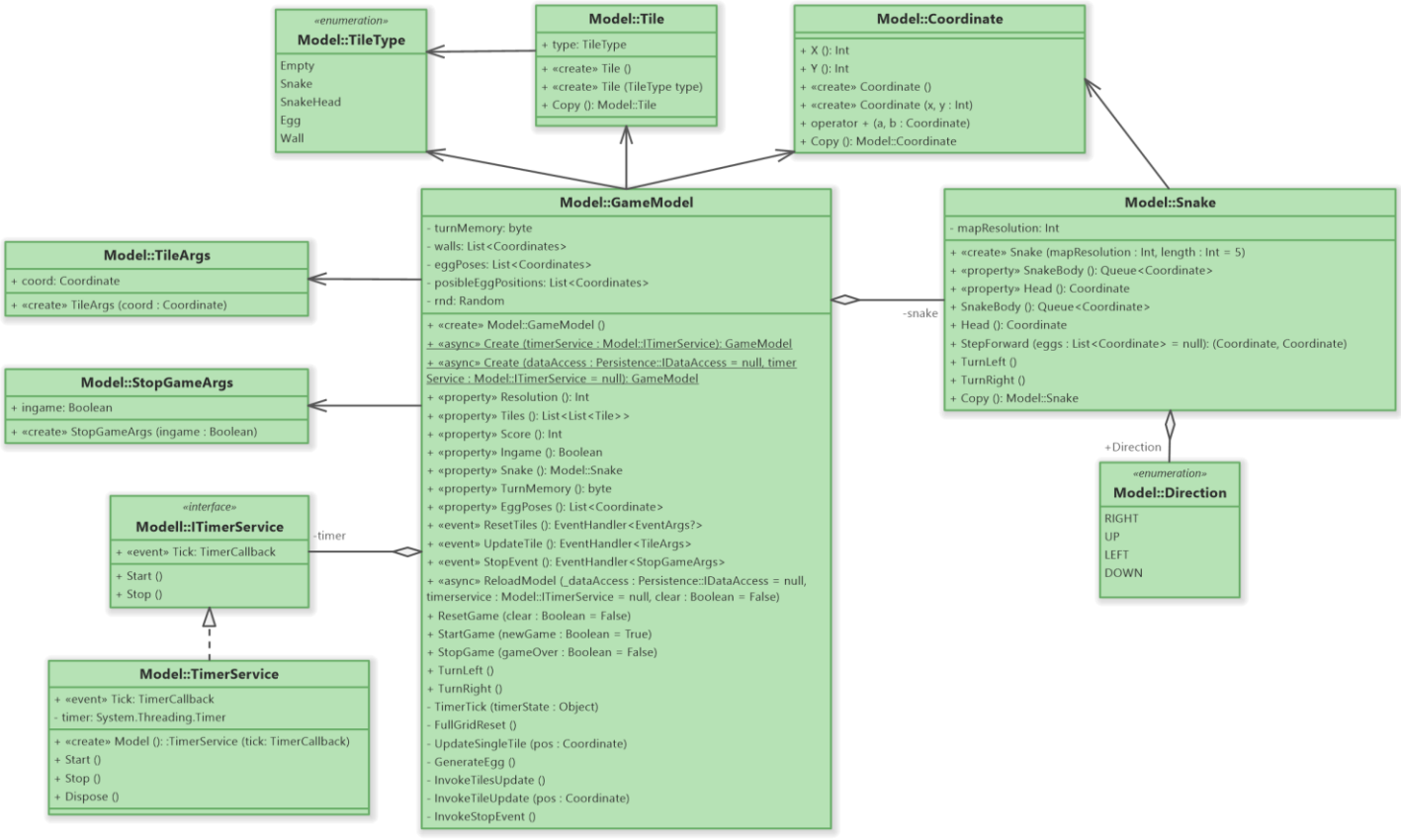
Csomagdiagram:



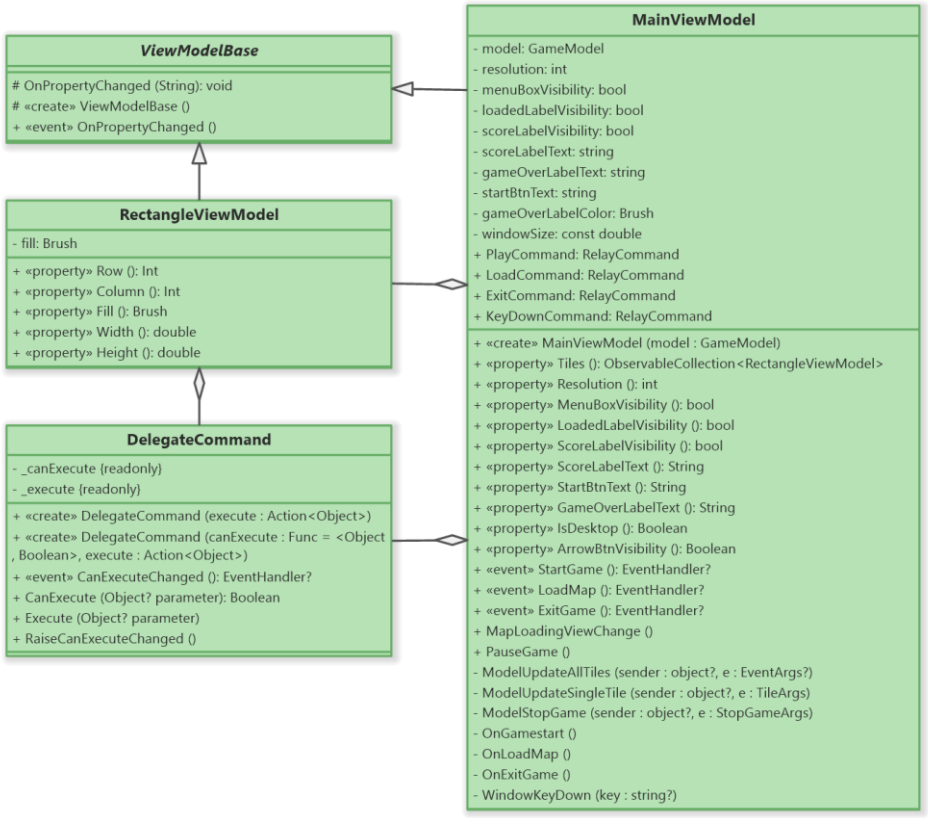
Persistence csomag osztálydiagramja:



Model csomag osztálydiagramja:



Nézetmodell osztálydiagramja:



Vezérlés osztálydiagramja: