




2. beadandó feladat – dokumentáció

Készítette:

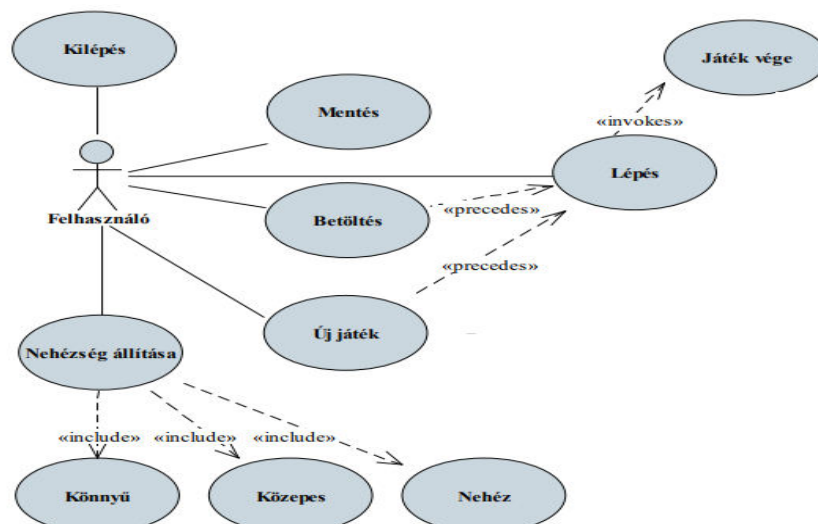
Hausknecht Bálint - B2ICWH

Feladat:

Készítsük programot, amellyel a klasszikus kígyó játékot játszhatjuk. Adott egy    lemezből álló játékpálya, amelyben akadályok (falak) találhatók. A játékos egy kezdetben 1 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 nehézségi szinttel játszhatjuk: könnyű (10x10), közepes (15x15) és nehéz (20x20). A mérkőzés nem indul automatikusan el, a játékos választ először nehézséget. Alapból 10x10 s méretű a pálya.
- A feladatot egyablakos asztali alkalmazásként Windows Presentation Foundation grafikus felülettel valósítjuk meg.
- Az ablakban elhelyezünk egy menüt a következő menüpontokkal: File (Új játék, Játék betöltése, Játék mentése, Kilépés), Beállítások (Könnyű játék, Közepes játék, Nehéz játék). Az ablak alján megjelenítünk egy státuszsort, amely a pontok számát, illetve az aktuális eseményt jelzi.
- A játéktáblát a $n \times n$ es nyomógombokból álló rács ábrázolja. A nyomógomb színének változásával tudjuk majd a kígyót illetve a tojásokat reprezentálni.
- A játék automatikusan feldob egy dialogus ablakot, ha vége a játéknak. Szintén dialogus ablakokkal végezzük el a mentést illetve a betöltést is.
- A felhasználó esetek:



Tervezés:

A program szerkezetét az MVVM architektúrának megfelelően bontjuk szét. (View,ViewModel,Model,Persistence) névtereket valósítok meg. A program környezetét az alkalmazás osztály (App) végzi, amely példányosítja a modellt, a nézetmodellt és a nézetet, biztosítja a kommunikációt, valamint felügyeli az adatkezelést.

Perzisztencia:

- A perzisztens rétegben található maga a **Játék tábla**, melyet a SnakeGameTable.cs osztály valósít meg. Ez az állomány tartalmazza 3 osztály implementálását: a fentebb említett táblát, kígyót és a kígyó darabjait. Ebből logikusan következik, hogy ezen a táblán maga a kígyó található, ami kígyó darabokból áll, illetve egy nxn-es Int32 típusú mátrix található, melyben az üres (0), kígyó darab (1), tojás (2) mezőket tárolom.
- Ezen rétegben található még az **adatkezelés** is : A hosszú távú adattárolás lehetőségeit az ISudokuDataAccess 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 SudokuFileDataAccess osztály valósítja meg. A fájlkezelés során fellépő hibákat a SudokuDataException kivétel jelzi.
- A program az adatokat szöveges fájlként tudja eltárolni, melyek az stl kiterjesztést kapják. Ezeket az adatokat a programban bármikor be lehet tölteni, illetve ki lehet menteni az aktuális állást

Modell:

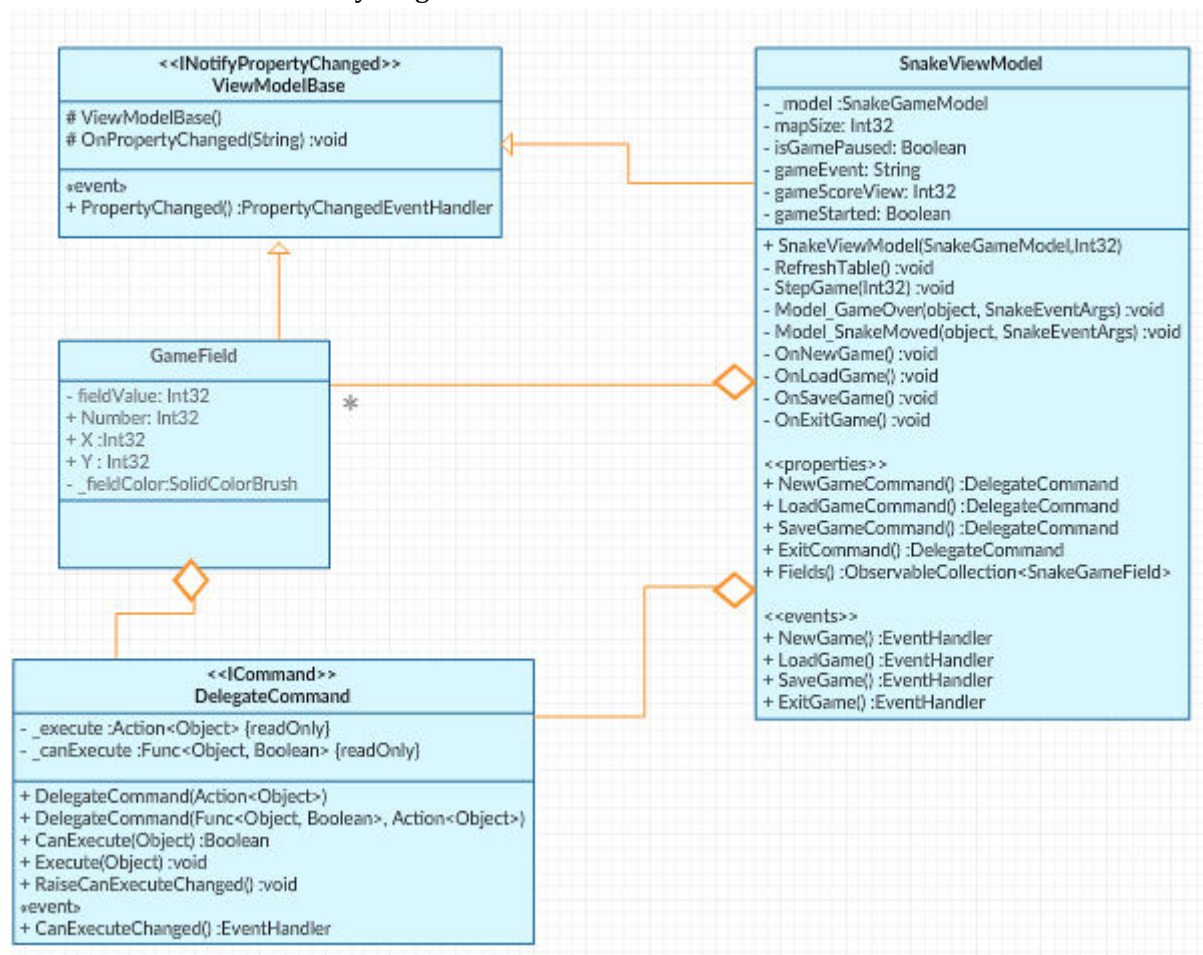
- A játéklógikához tartozó modellt a Model osztály valósítja meg (SnakeGameModel.cs), amely szabályozza a játék előrehaladását. Biztosít nekünk egy Timer-t, mely alapján a kígyó az aktuális irányba tud haladni egy úgynevezett move() függvény segítségével, melyet másodpercenként meghív. Biztosít nekünk ezen felül egy számlálót melyben a megevett tojások számát tároljuk.
- Két eseményt hoztam létre, az egyik a kígyó mozgására következik be (SnakeMoved), a másik szintén a kígyó mozgásának egy következménye, amely a játék végét mutatja (GameOver).
- A modell példányosításkor megkapja az adatkezelés felületét, amelynek segítségével lehetőséget ad betöltésre (LoadGameAsync) és mentésre (SaveGameAsync).

Nézet:

- Itt található a MainWindow.xaml, mely a felület leírására szolgál xml kód segítségével.
- Az pálya mezők, megjelenítendő attribútumok, menü pontok bindolása történik itt.

NézetModel:

- A nézetmodel 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, illetve a játéknehézséget szabályozza. Direkt nem avatkozik a játék futtatásába.
- A játék mező számára egy külön osztályt hoztam létre (GameField)
- Hozzá tartozó osztály diagram:



A modell funkcionalitása a GameModelTest.cs osztályban lett ellenőrizve. Osztályszerkezet:

