

Beadandó Feladat

Hallgató: Szabó Ábel

Neptun: U8ZXXS

Email: sz.abel221@gmail.com

Feladat leírása

Készítsünk programot, amellyel a következő játékot játszhatjuk.

Adott egy $n \times n$ elemből álló játékpálya, amelyben Maci Lacival kell piknikkosarakra vadászunk. A játékpályán az egyszerű mezők mellett elhelyezkednek akadályok (pl. fa), valamint piknikkosarak. A játék célja, hogy a piknikkosarakat minél gyorsabban begyűjtsük.

Az erdőben vadőrök is járőröznek, akik adott időközönként lépnek egy mezőt (vízszintesen, vagy függőlegesen). A járőrözés során egy megadott irányba haladnak egészen addig, amíg akadályba (vagy az erdő szélébe) nem ütköznek, ekkor megfordulnak, és visszafelé haladnak (tehát folyamatosan egy vonalban járőröznek). A vadőr járőrözés közben a vele szomszédos mezőket látja (átlósan is, azaz egy 3×3 -as négyzetet).

A játékos kezdetben a bal felső sarokban helyezkedik el, és vízszintesen, illetve függőlegesen mozoghat (egyesével) a pályán, a piknikkosárra való rálépéssel pedig felveheti azt. Ha Maci Lacit meglátja valamelyik vadőr, akkor a játékos veszít.

A pályák méretét, illetve felépítését (piknikkosarak, akadályok, vadőrök kezdőpozíciója) tároljuk fájlban. A program legalább 3 különböző méretű pályát tartalmazzon.

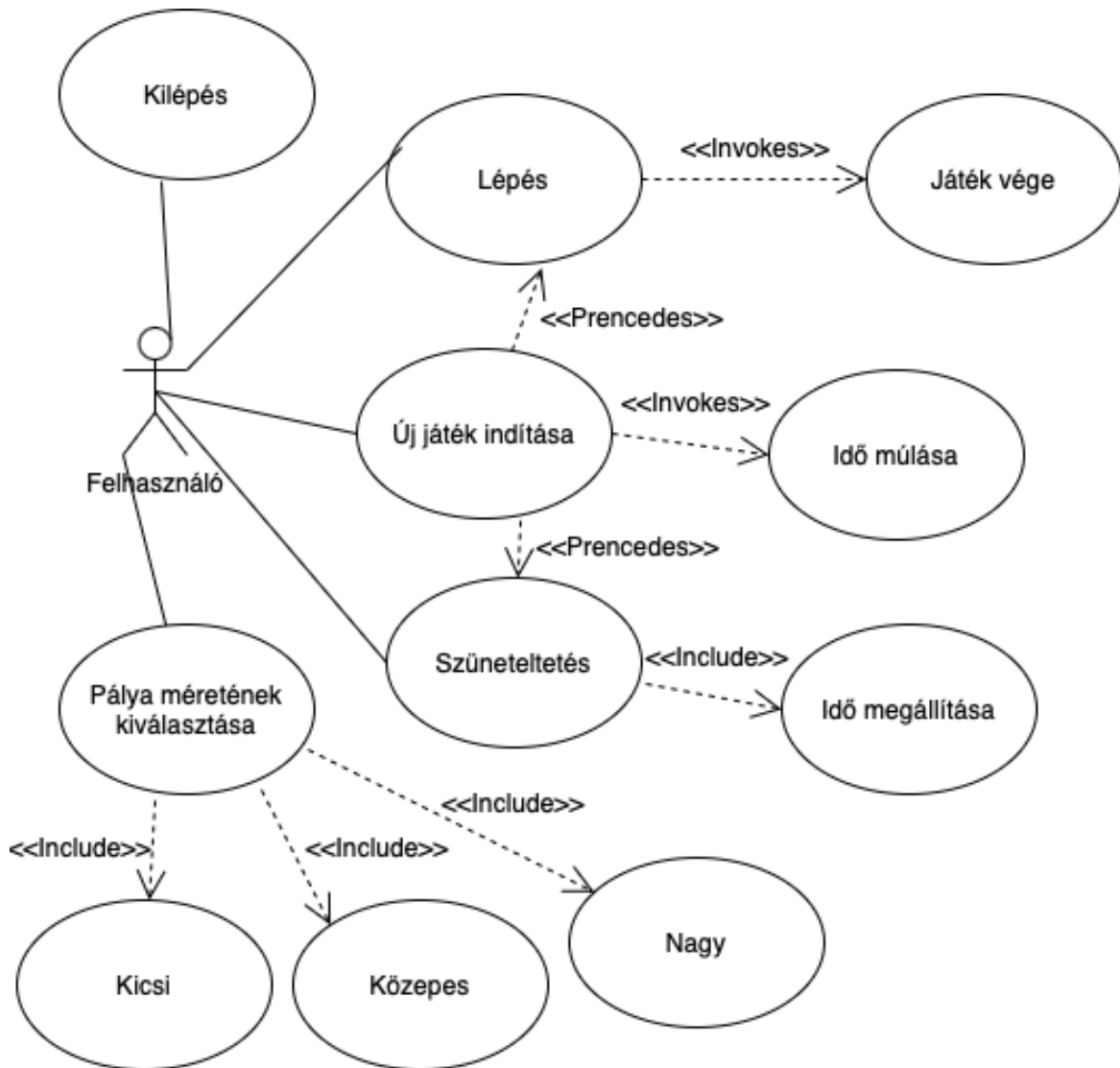
A program biztosítson lehetőséget új játék kezdésére a pálya kiválasztásával, valamint játék szüneteltetésére (ekkor nem telik az idő, és nem léphet a játékos). Ismerje fel, ha vége a játéknak, és jelezze, győzött, vagy veszített a játékos. A program játék közben folyamatosan jelezze ki a játékidőt, valamint a megszerzett piknikkosarak számát.

Elemzés

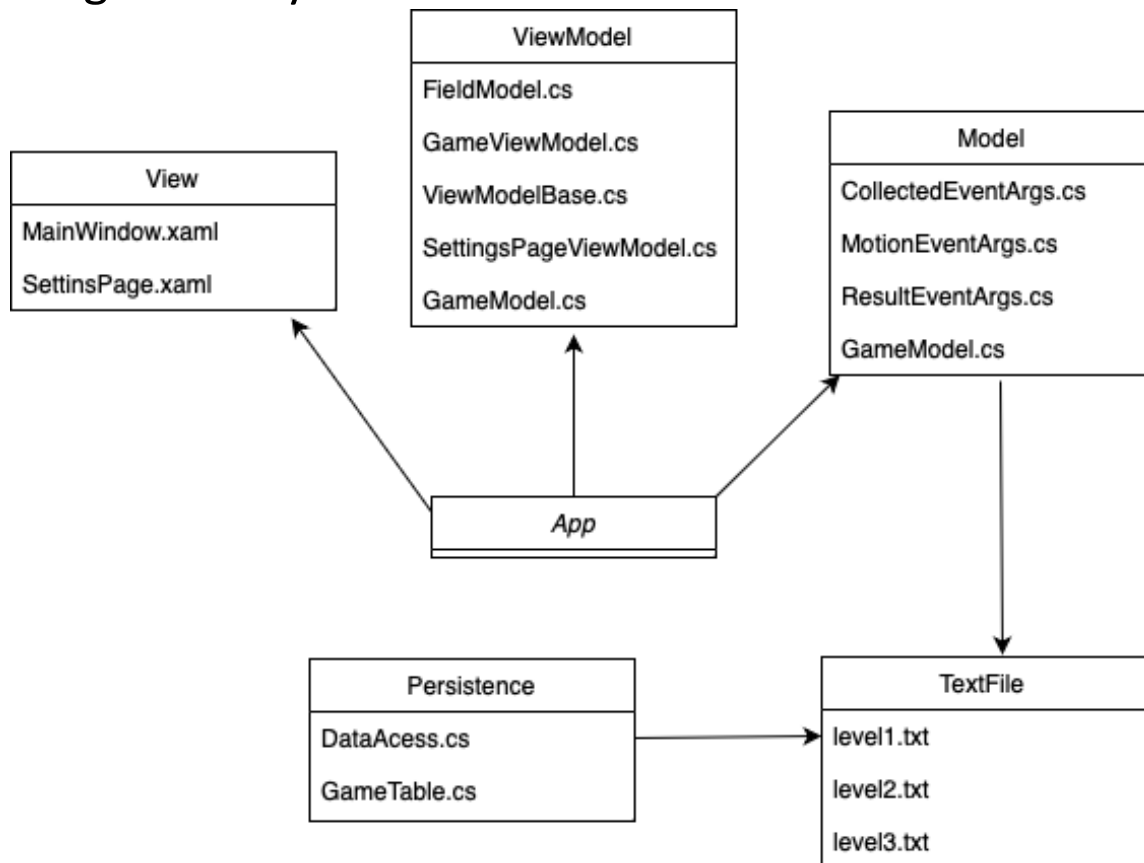
- A játékot három pályán lehet játszani. Kicsi pálya (7x7-es tábla 1 vadőr található rajta és 2 kosarat kell összegyűjteni), Közepes pálya (10x10-es tábla, 3 vadőr található rajta és 3 kosarat kell összegyűjteni), Nagy pálya (11x11-es tábla, 5 vadőr található rajta és 5 kosarat kell összegyűjteni). A program indításkor automatikusan a kicsi pályát tölti be.
- A program egy egyablakos alkalmazásként lesz megvalósítva WPF grafikus felülettel
- Az ablakban található egy menü, amiben új játékot lehet kezdeni (Új játék), meg lehet állítani a játékot (Szünet) (ha meg van nyomva a Szünet gomb akkor helyette egy Folytatás gomb jelenik meg amivel lehet folytatni a játékot.), pályát lehet választani (Pálya választása)
- A játékot egy $n \times n$ méretű (7x7, 10x10, 11x11) rács ábrázolja ahol a W,A,S,D gombokkal lehet irányítani a hőst aki a kezdetben a bal felső sarokban helyezkedik el

- A játéknak akkor van vége, ha a hős minden kosarat összegyűjtött, ekkor megjelenik egy dialógusablak amely kiírja, hogy mennyi idő alatt sikerült összegyűjteni a kosarakat, viszont ha közben a hőst meglátja egy vadőr (ami a vele szomszédos mezőket látja) akkor a játékos veszített, ezt is jelzi egy dialógusablak.

Felhasználói esetek



Program könyvtárszerkezete



Tervezés

■ Programszerkezet

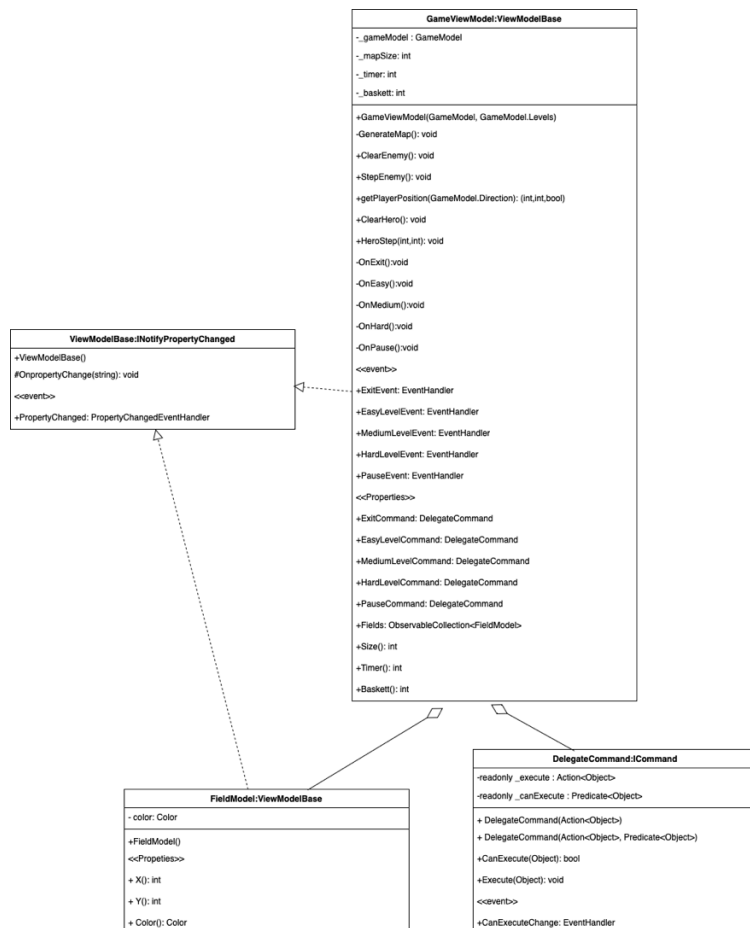
- A program négyrétegű architektúrában van megvalósítva (modell, nézet, nézetmodell és perzisztencia). A modell a Model mappába a nézet a View mappába, a nézetmodell a ViewModel és a perzisztencia a Persistence mappába található. Emellett található egy TextFiles mappa, amibe a játék felépítését tároló txt fájlok vannak.

■ Nézet

- A View mappába található.
- A nézet felépítését egy Grid-el oldottuk meg, ami három részre van tagolva, egy menüsávra (itt lehet kiválasztani a pályát, szüneteltetni és kilépni), egy statustrippe (amibe megtalálható az összegyűjtött kosarak számlálója (DataBindinggal a Baskett proprietiből nyerjük ki) és a timer (DataBindinggal a Timer proprietiből nyerjük ki)) és egy UniformGridre amibe a pálya található
- A UniformGrid gombokból áll és a méretét DataBindinggal kapjuk meg (Size proprietiből nyerjük ki)
- A játék fontos része a gombok színezése mivel mikor változik a pályát reprezentáló matrix akkor változnak a gombok színezései is

■ NézetModell

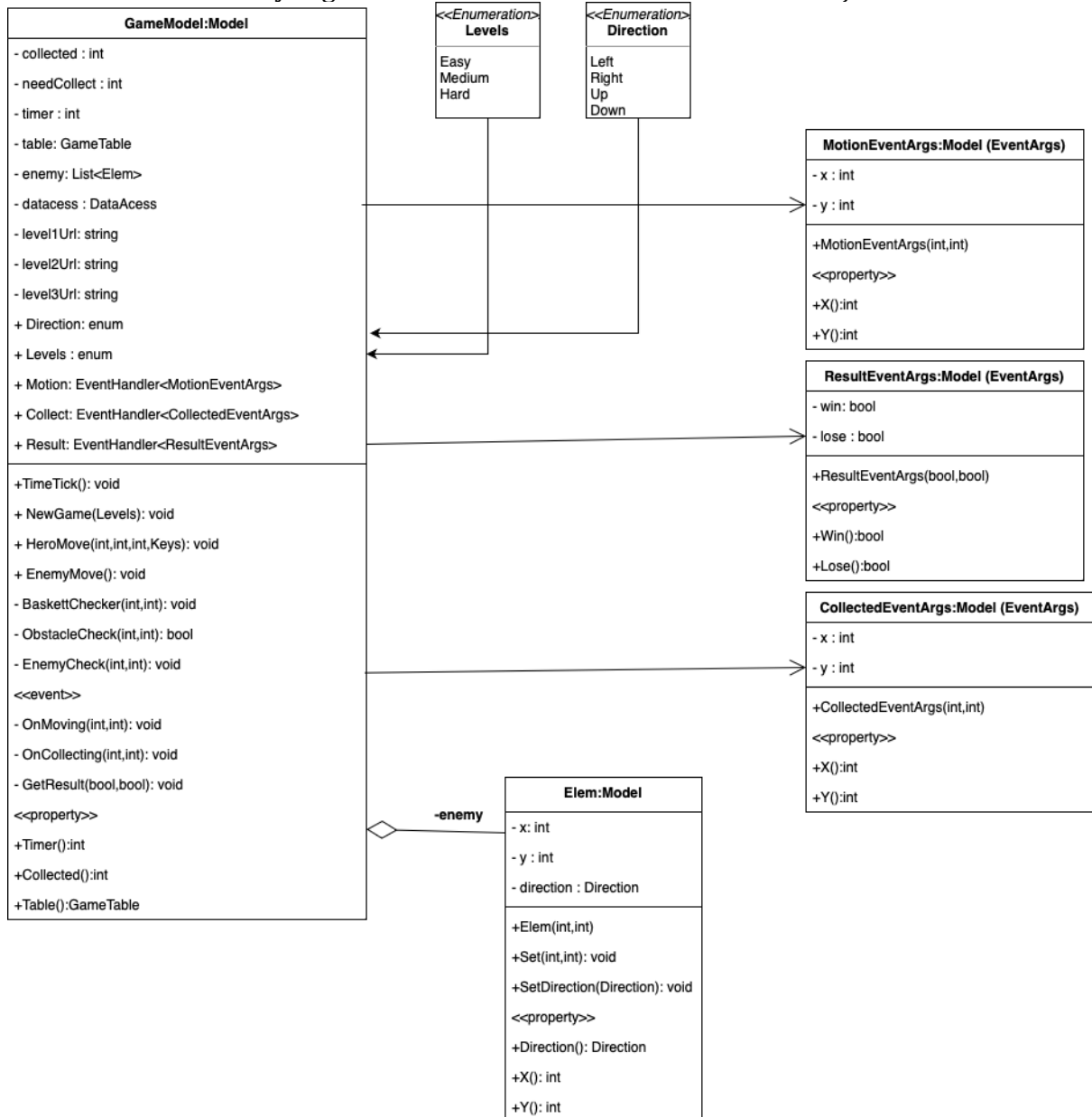
- A ViewModel mappába helyezkedik el.
- Lényege hogy a modellből kinyert adatokat továbbítja a nézet fele
- Megtalálható benne a FieldModel.cs ami leszármazik a ViewModelBase.cs (ami implementálja az INotifyPropertyChanged interfészt) osztályból és tárolja egy gomb elhelyezkedését és színét
- A DelegateCommand.cs (implementálja az ICommand interfészt) és segítségével nem kell minden Commandra különálló osztályt létrehozni.
- A nézetmodell legfontosabb osztálya a GameViewModel (ami leszármazik a ViewModelBase osztályból)
- A GameViewModel összegyűjti az összes DelegateCommandot és ő felel a megfelelő kommunikációért a View és a Modell között
- A Properties régió alatt megtalálhatóak azok a propertyk amiket ki szeretnénk nyerni a View-ba.
- A GenerateMap() metódus felel azért, hogy a pálya és a rajta található elemek megfelelően legyenek legénrálva
- A HeroMethods és az EnemyMethods régióban megtalálhatóak azok a metódusok amelyek vátoztatssák a Fields gyűjteményt amikor szükség van rájuk
- Az EventTriggers régióba találhatóak azok a metódusok amik szükségesek az események kivátásáért



■ Modell

- A modell legfontosabb részét a GameModel osztály valósítja meg.
- A játékot egy mátrix reprezentálja, ahol minden játékbeli elem egy számként van feltüntetve (1- hős, 2-ellenségek, 3-kosarak, 4-akadályok)
- A modell tárolja magában a táblát és a kiválasztott nehézségtől függően a NewGame() metódus a megfelelő pályát tölti be.
- A modellbe találhatóak még esemény argumentumok: MotionEventArgs amely tárolja a hős megváltozott pozícióját, CollectedEventArgs ami kosarak begyűjtésének pozícióját tárolja és ResultEventArgs ami a nyereség vagy a veszteség állapotát tárolja magába
- A HeroMove() metódus végzi a hős mozgását. Megfelelő billentyűt lenyomva lép a hős a megfelelő irányba (ha teheti). Az ObstacleChecker() figyel, hogy a lépni kívánt irányba van-e akadály, ha van akkor nem engedi a lépést. A BasketChecker() figyel, hogy a lépni kívánt irányba van-e kosár. Ha sikeres a lépés (nincs akadály vagy pálya széle) akkor a SetField() metódus beállítja a hős új pozícióját a mátrixba és törli a régit. Majd OnMoving() meghívásával az eseményargumentumba is beállításra kerül az új pozíció.
- Az EnemyMove() metódus végzi az ellenségek mozgását. A program úgy van megírva, hogy kezdetben minden ellenség kap egy irányt, amit követ (vízszintesen vagy függőlegesen mozog). Ha az adott ellenség akadályba

vagy a pálya szélébe ütközik, akkor beállításra kerül az ellentétes irány. Ha sikeres az ellenség lépése, akkor az EnemyCheck() metódus meghívásra kerül és megnézi, hogy a szomszédos mezőkön van-e a hős. Ha ott találja a hőst, akkor GetResult() metódus új értéket állít be a játék végért felelős esemény argumentumba és ennek hatására veszített a játékos



■ Perzisztencia

- A Persistence könyvtárba található a GameTable osztály ami beállítja a pályát reprezentáló mátrixot és le tudjuk kérdezni a GetField() metódussal a megadott mezőt és ha szeretnénk, akkor a SetField() metódussal mi is be tudunk állítani egyet.
- A szöveges fájlok feldolgozását a Persistence könyvtárba található Readfile() metódus végzi és soronként feldolgozza a txt fájlban levő mátrixot és a végén visszaadja az integer mátrixot. Ha valami hiba van a txt fájlban levő mátrixal, akkor (például tartalmaz betűt vagy nem megfelelő a hossza) a megfelelő hibaüzenettel jelzi a felhasználó felé.

GameTable:Persistence
- tableSize: int
- fieldTypes: int[,]
+GameTable(int[,])
+GetField(int,int):int
+Setfiled(int,int):void

DataAccess:Persistence
- path: string
+DataAccess(string)
+ReadFile(): int[,]

■ Környezet

- Az App() feladata hogy összekösse és példányosítsa az egyes rétegeket
- Az AppStart() felel a program elindításáért
- Az AppShell() be köti a ViewModelt és a Modelt
- Az AppShellbe történik meg a beállítási ablak létrehozása és majd megnyitása és a beállítási ablakban lehet pályát megnyitani

App (Application)
-shell: Appshell
- _gameViewModel: GameViewModel
- _gameModel: GameModel
-persistence: DataAccess
#CreateWindow(Iactivationstate): Window
AppShell (Application)
- _gameViewModel: GameViewModel
-model: GameModel
-persistence: DataAccess
-timer: int
-timer: IDispatcherTimer
-settingsPageViewModel: SettingsPageViewModel
+AppShell(DataAcces,GameModel,GameViewModel)
-InitializeGame()
internal StartTimer()
internal StopTimer()
-NewGame(string): void
-TimeTick(): void
-BasketttCollected(object, CollectedEventArgs): void
-EndGame(object, ResultEventArgs): void
-UpdateHeroPosition(object, MotionEventArgs): void
-UpdateEnemyPosition(): void
<<async>>
- async Settiings(object, EventArgs): void
- async Leve1(object, EventArgs): void
- async Level2(object, EventArgs): void
- async Level2(object, EventArgs): void

Tesztelés

A program helyes működését egységtesztekkel támasztjuk alá.

A tesztek a GameTest osztály végzi.

- TestGameTableSetup(): Itt tesztelésre kerül, hogy megfelelően állítja be a tábla értékeit és hogy helyesen kérdezi le azokat.
- HeroRightMove()
- HeroLeftMove()
- HeroMoveUp()
- HeroMoveDown():
- HeroMeetWall() : Itt tesztelésre kerül, hogy megfelelően mozog e a hős és ha találkozik egy akadállyal, akkor nem halad át rajta.
- CollectBaskett(): Itt tesztelésre kerül, hogy megfelelően gyűjti össze a kosarakat és megfelelően nő az ezt számon tartó változó is
- EnemyMoveTest(): Itt tesztelésre kerül, hogy megfelelően mozog e az ellenség és ha elhalad a pálya széléig, akkor vissza fordul e
- TestFileReading(): Itt tesztelésre kerül hogy a ReadFile() metódus megfelelően olvassa e be a txt fájlban levő mátrixot.