2. Beadandó feladat dokumentáció

Készítette:

Németh Lehel

email: hqsbfp@inf.elte.hu

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 játékban három különböző méretű „pályán” játszhatunk, mely azt határozza meg, hogy a játékteret mekkora kockákra osszuk fel. A Kicsi módban 13x13 mátrixra van felosztva a terület, így a kígyó is nagyobb, kevesebb helye van mozogni és gyorsabb. Így a közepes (25x25), illetve a nagy pálya (50x50) esetén könnyebb a játék. A játék addig megy, amíg a játékos ki nem megy a pályáról vagy neki nem megy saját magának. Kezdetben a játékos egy 5 egység hosszú kígyót irányít, ami minden megevett gyümölccsel egyre hosszabb lesz.

A programot WPF grafikus felületen valósítottam meg. A grafikai megvalósítás minimalista, hogy jobban illeszkedjen a retro stílusba.

Indítás után a felhasználó két gombot lát. A Start Game átirányítja egy menüben, ahol ki tudja választani, hogy melyik pályán játsszon, a választás után már egyből indul is a játék. Ha viszont a Leaderboard-ra kattint, akkor pedig megnézheti a korábbi játékosok neveit és elért pontszámát. Ha vége a játéknak, akkor a felhasználó megadhatja a nevét, amit a program eltárol a többi eredmény között. Ha nem adja meg, akkor névtelenül kerül mentésre az eredmény.

Tervezés:

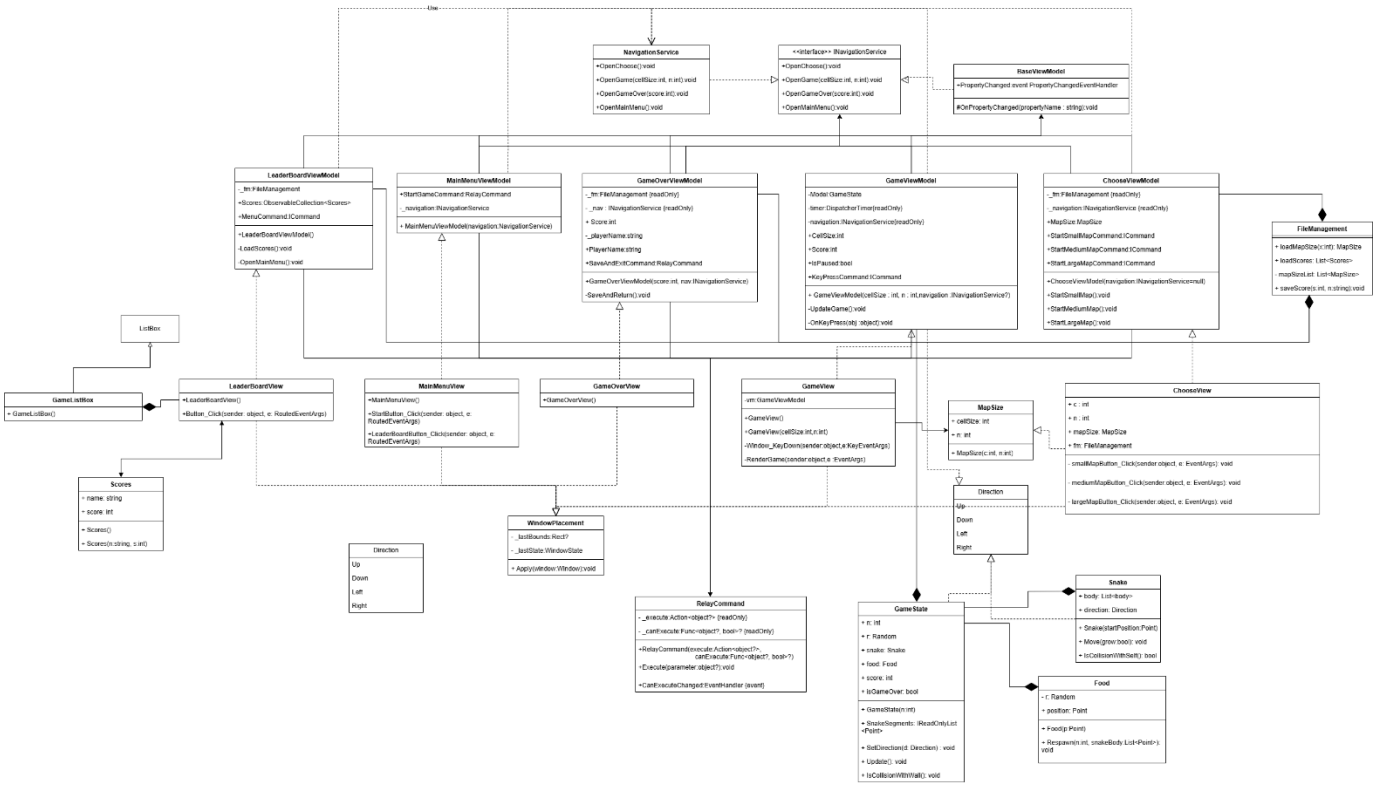
- Programszerkezet:
    - A programot MVVM architektúrában valósíjuk meg. A megjelenítés a View, a modell a Model, a kettő közötti kapcsolatért a Viewmodel felel,

míg a perzisztencia a Persistence névtérben helyezkedik el. A program csomagszerkezete a fenti ábrán látható.

- Perzisztencia:
  - Az adatkezelés (**FileManagement** osztály) feladata, hogy beolvassa a mapSize.txt-ből a kiválasztott pálya méretét, valamint a játék végén elmenti a scores.txt-be a játékos nevét (ha nem ad meg nevet, akkor Anonymous néven lesz elmentve) és az elért pontszámát.
- Modell
  - A **Snake** osztály egy **body** listában tárolja egy kígyó minden elemének pontos pozícióját, ami folyamatosan változik, ahogy a kígyó mozog. Ha megeszik egy gyümölcsöt, akkor a lista egy elemmel hosszabb lesz. A **direction** enum azt adja meg, hogy milyen irányban mozog a kígyó. A Move függvény kezeli a kígyó mozgását, ez adja hozzá az új elemet a body listába és törli a legvégét, ha a kígyó nem evett. Az **IsCollisionWithSelf** függvény vizsgálja, hogy a kígyó beleütközött-e a saját farkába.
  - A **Food** osztály minden meghívásakor egy random pontot generál, ahol megjelenik az étel. Ha megeszi a kígyó, akkor a **Respawn** függvény újragenerálja egy random helyre, ami nem lehet soha a kígyó pozíciója.
  - A **GameState** osztály felel a játék motorjáért. Ez hívja meg a Food és a Snake osztályokat is. Az **Update** osztály folyamatosan hívódik meg, ez figyeli, hogy véget ért-e a játék, ez hívja meg a kígyó Move és a food Respawn függvényét. Az **IsCollisionWithWall** figyeli, hogy a kígyó nekiment-e a falnak.
- View:
  - A **View**-ban van egy származtatott osztályom a **GameListBox**. Erere azért volt szükség, hogy a korábbi eredmények kiírtatása is megfeleljen a játékom stílusának és korlátozottak voltak a testreszabási lehetőségeim.
  - A **MainMenuView** a játék főmenüje, Ez tartalmaz egy **StartGameButton**-t, és egy **leaderBoardButton**-t. A StartGameButton betölti a **ChooseView**-t, a leaderBoardButton pedig a **LeaderboardView**-t.
  - A **ChooseView** tartalmaz három gombot (Small, Medium, Large), amivel a pálya méretét lehet kiválasztani. A gombok megnyomása után betölt a **GameView** és elkezdődik a játék a kiválasztott méretű pályán.
  - A **LeaderboardView** egy GameListBoxot tartalmaz, amin megjelennek a korábbi játékosok eredményei, csökkenő sorrendben. Ez alatt található egy gomb, ami visszavisz a főmenübe.

- A **GameView** felel a játék megjelenítéséért. Ez rajzolja ki a kígyót és a gyümölcsöket, írja ki a bal felső sarokba a pontszámot, és a **Pause** menüt. A **Window_KeyDown** figyeli a gombnyomást.
- A **GameOverView** akkor tölt be, amikor véget ér a játék. Ez kiírja az elért pontszámot és egy textboxot tartalmaz, ahova beírhatja a nevét a felhasználó. A Menu gomb visszaviszi a játékost a főmenübe, valamint elmenti az elért eredményét.

o ViewModel:
- A **MainMenuViewModel** felelős a főmenü gombjainak logikájáért. A **StartGameCommand** meghívja az **INavigationService.OpenChoose** metódust, amellyel betölti a **ChooseView**-t.
- A **ChooseViewModel** kezeli a pályaméret kiválasztását. Három parancsot tart fenn (**StartSmallMap**, **StartMediumMap**, **StartLargeMap**), amelyek betöltik a megfelelő **MapSize**-t a **FileManagement** segítségével, majd az **INavigationService** segítségével elindítják a **GameView**-t.
- A **GameViewModel** a teljes játékmenetet irányítja. Egy **DispatcherTimer** segítségével folyamatosan frissíti a játék állapotát, kezeli a pontszámot, az ütközéseket, a **pause** funkciót és a játék végét. A **KeyPressCommand** feldolgozza a felhasználói irányítási parancsokat, és szükség esetén a navigációs szolgáltatással megnyitja a **GameOverView**-t.
- A **GameOverViewModel** felel a játék végén megjelenő nézet logikájáért. Elmenti a játékos nevét és pontszámát a **FileManagement** segítségével, majd az **INavigationService** segítségével visszanavigál a főmenübe. A **SaveAndExitCommand** indítja el a mentést és a nézetváltást.
- A **LeaderBoardViewModel** tölti be a ranglistán megjelenő adatokat a **FileManagement** segítségével, és rendezve adja át azokat a **LeaderBoardView** számára. A **MenuCommand** visszanavigál a főmenübe. A ViewModel biztosítja az **ObservableCollection<Scores>** gyűjteményt, amely automatikusan frissíti a listát a View-ban.



Az UML diagram megtalálható a fájlok között, .svg és .png kiterjesztésben is.

Tesztelés:

A modell funkcionalitása egységtesztek segítségével lett ellenőrizve a wpf_snake_test osztályban.

Az alábbi tesztesetek kerültek megvalósításra:

- **SnakeTests**:
  - **Move_NonGrowing_RemovesTail**: Teszteli, hogy ha a kígyó **nem nő**, akkor a Move(false) hívás után a test **ugyanolyan hosszú marad**.
  - **Move_Growing_IncreasesLength**: Teszteli, hogy ha a kígyó **növekedést kér** (grow: true), akkor a mozgás után a test **1 elemmel hosszabb lesz**.
  - **IsCollisionWithSelf_WhenHeadOverlapsBody**: Teszteli, hogy a kígyó saját testébe ütközését helyesen érzékeli.
- **FoodTests:**
  - **Respawn_DoesNotPlaceOnSnake**: Teszteli, hogy a Respawn() metódus soha nem rakja a kaját a kígyó testére.
- **GameStateTests:**
  - **Update_IncrementsScore**: Teszteli, hogy ha a kígyó ételre lép, akkor az Update() növeli a pontszámot.
  - **Update_SetsGameOver_OnWallCollision:** Teszteli, hogy az Update() felismeri a falnak ütközést.

    - **Update_SetsGameOver_OnSelfCollision**: Teszteli, hogy a GameState felismeri a saját testnek ütközést.
- **FileManagementTests:**
    - **LoadMapSize**: Teszteli, hogy a loadMapSize() a megfelelő indexű pályaméretet tölti be a mapsize.txt-ből.
    - **SaveScore**: Teszteli, hogy a saveScore() helyesen hozzáadja az új eredményt a scores.txt fájlhoz.
- **ChooseViewModelTests:**
    - **StartSmallMap**: Teszteli, hogy a ViewModel helyesen tölti be az első pálya méretét és meghívja a navigációs szolgáltatás OpenGame() függvényét a megfelelő paraméterekkel.
- **GameViewModelTests:**
    - **KeyPressChangesDirection**: Teszteli, hogy a gombnyomás esetén megfelelő irányba változik a kígyó mozgása.
    - **KeyPressTogglesPauseTest**: Teszteli, hogy a "P" lenyomása megállítja, a következő "P" pedig folytatja a játékot. Ellenőrzi az IsPaused logika működését.
    - **NavigateGameOver**: Teszteli, hogy a GameViewModel felismeri a game over állapotot és meghívja a navigációs szolgáltatás OpenGameOver() metódusát
- **GameOverViewModelTests:**
    - **SaveAndReturn**: Teszteli, hogy a GameOverViewModel elmenti a pontszámot a scores.txt-be a saveScore() segítségével és meghívja a navigációs szolgáltatás OpenMainMenu() függvényét