3. Beadandó feladat dokumentáció

Készítette:
Virág Sanel
E-mail: t8hgxr@inf.elte.hu

Feladat:

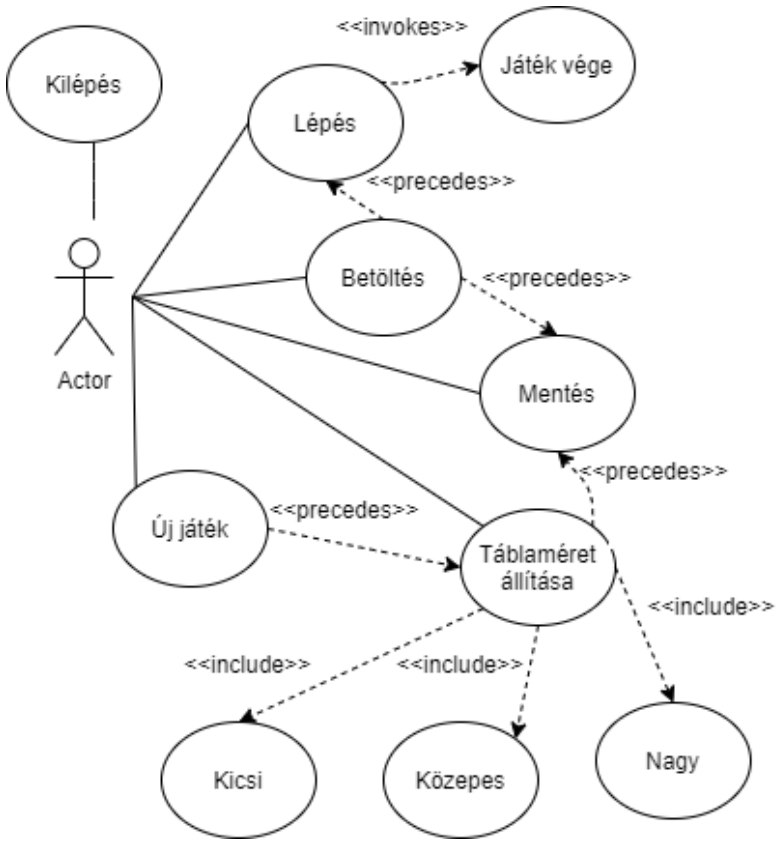
Aknakereső

Készítsünk programot, amellyel az aknakereső játék két személyes változatát játszhatjuk. Adott egy $n \times n$ mezőből álló tábla, amelyen rejtett aknákat helyezünk el. A többi mezőt szintén elrejtve tárolják, hogy a velük szomszédos 8 mezőn hány akna helyezkedik el. A játékosok felváltva léphetnek. Egy mező felfedjük annak tartalmát. Ha az akna, a játékos veszített. Amennyiben a mező nullát rejt, akkor a vele szomszédos mezők is automatikusan felfedésre kerülnek (és ha a szomszédos is nulla, akkor annak a szomszédai is, és így tovább). A játék addig tart, amíg valamelyik játékos aknára nem lép, vagy fel nem fedték az összes nem akna mezőt (ekkor döntetlen lesz a játék). A program biztosítson lehetőséget új játék kezdésére a pályaméret megadásával (6 × 6, 10 × 10, 16 × 16), valamint játék mentésére és betöltésére. Ismerje fel, ha vége a játéknak, és jelenítse meg, melyik játékos győzött (ha nem döntetlen).

Elemzés:

- A játékot három táblamérettel játszhatjuk: kicsi (6x6), közepes (10x10), nagy (16x16). Program indulásakor közepes méretű tábla jelenik meg.
- A feladatot Xamarin Forms alkalmazásként, Android platformon valósítjuk meg, amely négy lapból fog állni. Az alkalmazás portré tájolást támogat.
    - Az első képernyő (Játék) tartalmazza a játéktáblát, jelenleg soron lévő játékost a lap tetején, az új játék, valamint a betöltés és mentés gombok vannak.
    - A második képernyőn (új játék) van lehetőség új játék indítására, táblaméret kiválasztására.
    - 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.

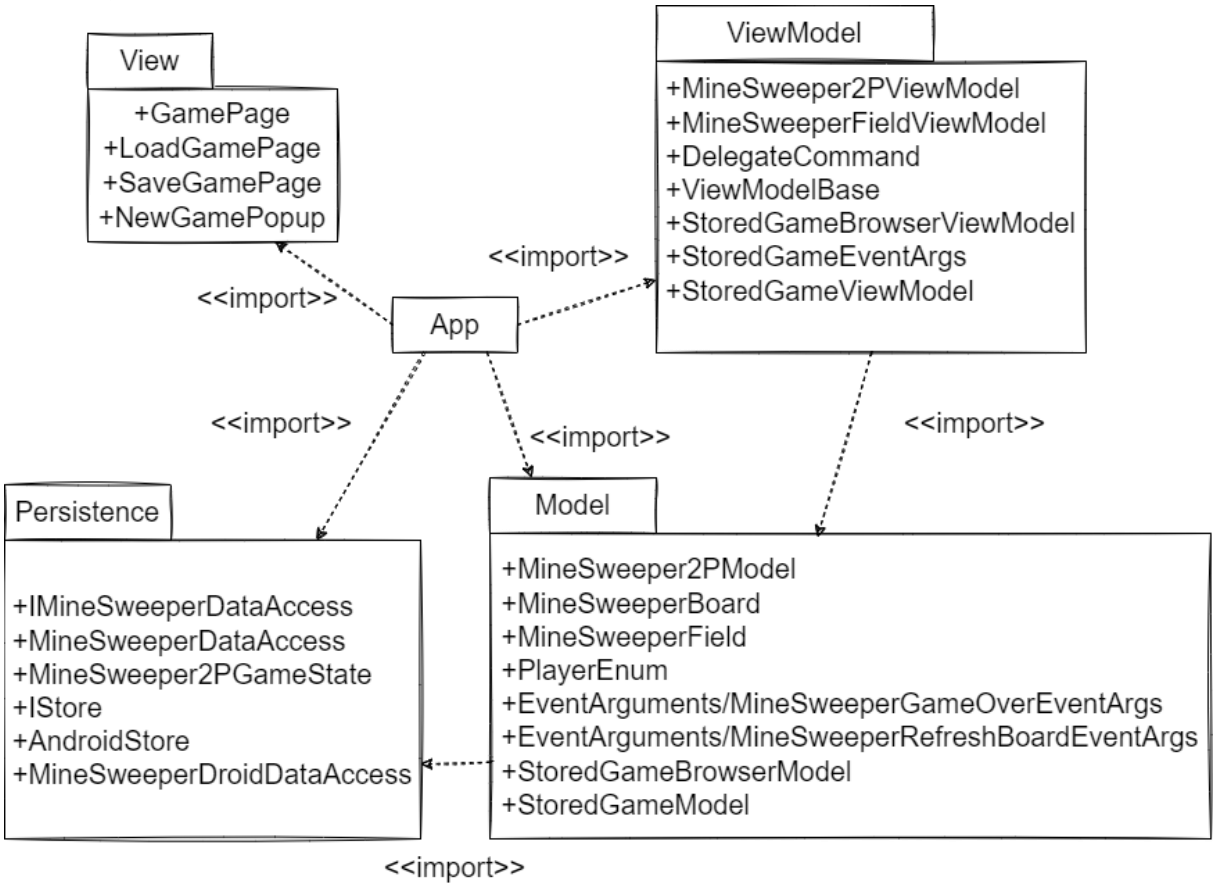
- A játéktábla a beállítot játék mérettel megegyező számú nyomógomból áll, melyeket rácsba rendezünk el. A nyomógomb érintés hatására a feladatban leírtaknak megfelelően fed fel mezőt/mezőket, majd a következő játékosra vált. Csak azokra a mezőkre lehet nyomni, melyek nem felfedettek.
- Ha egy aknát tartalmazó mezőre lépünk, akkor soron lévő játékos veszített, és a játéknak vége. Ha már nincs akna nélküli mező, a játék döntetlennel véget ér. Az összes aknát tartalmazó mező felfedődik. Ilyenkor nem váltunk át a következő játékosra. Megjelenik egy üzenet ablak, mely kiírja, hogy döntetlen-e játék, vagy azt, hogy ki nyert.
- Felhasználói esetek a követekező ábrán láthatóak:



Tervezés:

- Programszerkezet:
  - A szoftvert két projektből építjük fel, a Xamarin Forms megvalósítást tartalmazó osztálykönyvtárból (.NET Standard Class Library), valamint az Android platform projektből. Utóbbi csupán a perzisztencia Android specifikus megvalósítását tartalmazza, minden további programegységet az osztálykönyvtárban helyezünk el.

  • A programot MVVM architektúrában valósítjuk meg. A megjelenítés a **View**, a modell a **Model**, a nézetmodell a **ViewModel**, míg a perzisztencia a **Persistence** névtérben helyezkedik el. A program környezetét az alkalmazás osztály (**App**)
  végzi, amely példányosítja a modellt, a nézetmodell és a nézetet, biztosítja a kommunikációt, valamint felügyeli az adatkezelést. A program csomagszerkezete a következő ábrán látható.



- Perzisztencia
  - Az adatkezelés biztosítja a játék mentését és betöltését. Tárolja a játéktáblát, és a jelenlegi soron lévő játékost.

- **MineSweeper2PGameState** tárol egy érvényes játéktáblát, a tábla méretét, és a jelenleg soron lévő játékost. Csak a játék állapotának mentésére és betöltésére szolgál, amelyet a játékmodellből lekért adatokból (tábla, soron lévő játékos) állít elő.

- A hosszú távú adattárolás lehetőségeit az **IMineSweeperDataAccess** 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 Android adatkezelésre a **MineSweeperDroidDataAccess** osztály valósítja meg. A fájlkezelés során fellépő hibákat beépített Exception kivétel jelzi.

- A program az adatokat szöveges fájlként tudja eltárolni, melyek a felhasználó személyes könyvtárában helyezünk el.
  Ezeket az adatokat a programban bármikor be lehet tölteni, illetve ki lehet menteni az aktuális állást.
  Az állás csak akkor menthető, ha van érvényes játék (tehát el van indítva és még nincs vége).

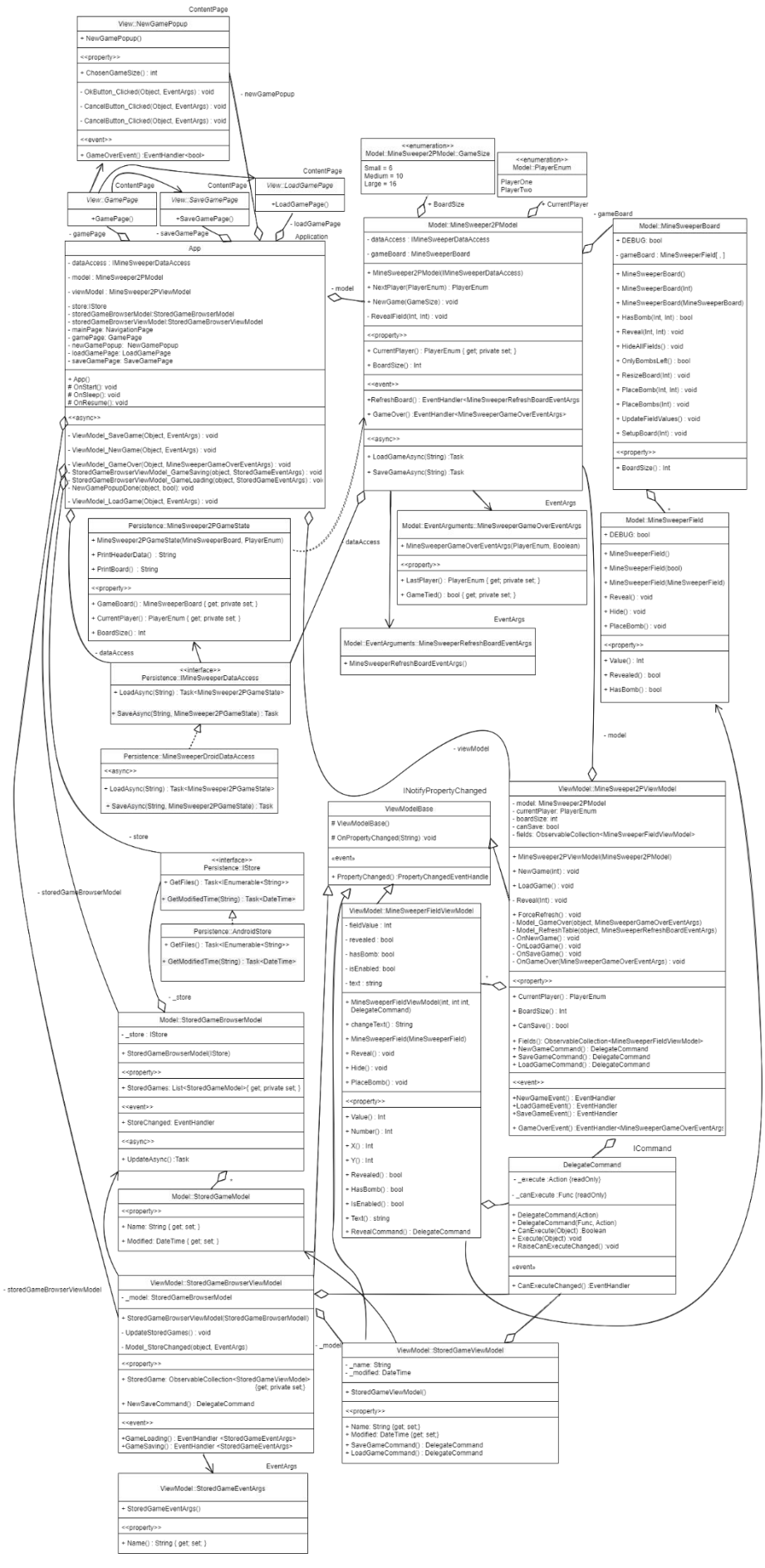
- A fájl első sora megadja a tábla méretét és a soron lévő játékost. Ezután a tábla méretnek megfelelő számú sor található melyek ugyan ennyi karaktert tartalmaznak, a táblát szimbolizálva. A nem felfedett mezők “#“ tartalmaznak. A nem felfedett és aknát tartalmazók „x”-el vannak jelölve, a felfedett, aknát tartalmazók pedig „X”-el (normális játék során ilyet nem tudunk menteni, kivételkezelés és tesztelési okokból van engedélyezve). A felfedett, nem aknát tartalmazó mezők az őket körbevevő aknák számával van jelölve.

- Modell
  - A játékmodell lényegi része a **MineSweeper2PModel**-ben található. Tartalmazza a játéktáblát (**MineSweeperBoard**), mely **MineSweeperField** típusú mezőkből áll. Emellett a **MineSweeper2PModel** számon tartja a jelenlegi játékost (**CurrentPlayer)** és a játék méretét (**BoardSize),** amit a tábla

típusából nyer ki. Lehetőséget ad új játék kezdésére, valamint mező felfedésére (**NewGame, RevealField).** Új játék kezdésekor, megadva a pálya méretét, pszeudo-véletlenül generálódnak a táblán az aknák. Kb. 25% a mezőknek fog aknát tartalmazni.
  - A modell tartalmazz egy **MineSweeperBoard** típusú játéktáblát, amely mezőit fedjük fel a játék során. Kezeli az aknák és mezőértékek generálásának (**SetupBoard, PlaceBombs, UpdateFieldValues, PlaceBombs)** és mezők felfedésének (**Reveal)** logikáját. A táblától kéri le a modell, hogy a játék döntetlen-e (**OnlyBombsLeft).**
  - A tábla **MineSweeperField** típusú mezőkből áll. E mező saját inforációit tartalmazza: felfedett-e, van-e akna rajta, értéke. A tábla a mező saját metódusait (**PlaceBomb, Reveal, Value, Revealed, HasBomb)** hívja meg táblametódusaiban
  - A modell a játéktábla frissítésének eseményét a **RefreshBoard** eseménnyel váltja ki, a játék végét pedig **GameOver** eseménnyel. A **GameOver** esemény argumentuma (**MineSweeperGameOverEventArgs)** tartalmazza, hogy a játék döntetlen-e, és hogy ki volt az utolsó játékos (nem döntetlen esetén ki volt a vesztes). A **RefreshBoard** esemény argumentuma (**MineSweeperRefreshBoardEventArgs)** nem hordoz semmilyen plusz információt, de ha a jövőben ezen változtatni kellene, segitségével ez megoldható.

  - A modell példányosításkor megkapja az adatkezelés felületét, amelynek segítségével lehetőséget ad betöltésre (**LoadGameAsync**) és mentésre (**SaveGameAsync**)
  - A játéktábla méretét **GameSize** felsorolólási típussal adhatjuk meg, amit a táblából kérdezünk le modellben.
- **Nézetmodell**
  - A nézetmodell megvalósításához felhasználunk egy általános utasítás (DelegateCommand), valamint egy ős változásjelző (ViewModelBase) osztályt.
  - A nézetmodell feladatait a **MineSweeper2PViewModel** osztály látja el, amely parancsokat biztosít az új játék kezdéséhez, játék betöltéséhez, mentéséhez. 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 játéktábla méretét szabályozza. Direkt nem avatkozik a játék futtatásába.
    - A játékmező számára egy külön mezőt biztosítunk (**MineSweeperFieldViewModel**), amely eltárolja **MineSweeperField** tulajdonságait, és mező felfedésének parancsát (**RevealCommand**). A mezőket egy felügyelt gyűjteménybe helyezzük a nézetmodellbe (**Fields**)
- **Nézet**
    - A nézetet navigációs lapok segítségével építjük fel.
    - A nézet egy főablakból áll (**GamePage**). A nézet egy rácsban tárolja a játékmezőt, a státuszsort, és az új játék, betöltés és mentés gombokat. A játékmező egy CollectionView vezérlő, ahol dinamikusan felépítünk egy rácsot (GridItemsLayout), amely gombokból áll. Minden adatot adatkötéssel kapcsolunk a felülethez.
    - Betöltés és mentés esetén felugrik egy lap (**LoadGamePage, SaveGamePage**), melyekben egy listában megjelennek az eddig elmentett játékmenetek. Mentéskor egy bemeneti mező segítségével megadhatjuk a menteni kívánt játék nevét.
    - Új játék indításakor felugrik egy lap (**NewGamePopup**), mely segítségével beállíthatjuk a kívánt játéktábla méretét. Választhatunk kicsi (6x6), közepes (10x10) és nagy (16x16) méretek közül, alapértelmezetten közepes. OK gomb megnyomásával generálódik egy új játék, beállított méretben.
- **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.
    - Kezeljük az alkalmazás életciklust, így felfüggesztéskor (**OnSleep**) elmentjük az aktuális játékállást (**AutoSave**), folytatáskor (**OnResume**) és újraindításkor (**OnStart**) pedig folytatjuk, amennyiben történt mentés.
- Program statikus szerkezete:


ContentPage
View::NewGamePopup
+ NewGamePopup()
<<property>>
+ ChosenGameSize() : int
- OkButton_Clicked(Object, EventArgs) : void
- CancelButton_Clicked(Object, EventArgs) : void
- CancelButton_Clicked(Object, EventArgs) : void
<<event>>
+ GameOverEvent() :EventHandler<bool>
- newGamePopup
ContentPage
View::LoadGamePage
+LoadGamePage()
- loadGamePage
ContentPage
View::GamePage
+GamePage()
- gamePage
ContentPage
View::SaveGamePage
+SaveGamePage()
- saveGamePage
Application
App
- dataAccess : IMineSweeperDataAccess
- model : MineSweeper2PModel
- viewModel : MineSweeper2PViewModel
- store:IStore
- storedGameBrowserModel:StoredGameBrowserModel
- storedGameBrowserViewModel:StoredGameBrowserViewModel
- mainPage: NavigationPage
- gamePage: GamePage
- newGamePopup : NewGamePopup
- loadGamePage: LoadGamePage
- saveGamePage: SaveGamePage
+ App()
# OnStart(): void
# OnSleep(): void
# OnResume(), void
<<async>>
- ViewModel_SaveGame(Object, EventArgs) : void
- ViewModel_NewGame(Object, EventArgs) : void
- ViewModel_GameOver(Object, MineSweeperGameOverEventArgs) : void
- StoredGameBrowserViewModel_GameSaving(object, StoredGameEventArgs) : void
- StoredGameBrowserViewModel_GameLoading(object, StoredGameEventArgs) : void
- NewGamePopupDone(object, bool): void
- ViewModel_LoadGame(Object, EventArgs) : void
- model
<<enumeration>>
Model::MineSweeper2PModel::GameSize
Small = 6
Medium = 10
Large = 16
<<enumeration>>
Model::PlayerEnum
PlayerOne
PlayerTwo
+ BoardSize
+ CurrentPlayer
Model::MineSweeper2PModel
- dataAccess : IMineSweeperDataAccess
- gameBoard : MineSweeperBoard
+ MineSweeper2PModel(IMineSweeperDataAccess)
+ NextPlayer(PlayerEnum) : PlayerEnum
+ NewGame(GameSize) : void
- RevealField(Int, Int) : void
<<property>>
+ CurrentPlayer() : PlayerEnum { get; private set; }
+ BoardSize() : Int
<<event>>
+RefreshBoard() : EventHandler<MineSweeperRefreshBoardEventArgs>
+ GameOver() :EventHandler<MineSweeperGameOverEventArgs>
<<async>>
+ LoadGameAsync(String) :Task
+ SaveGameAsync(String) :Task
- gameBoard
Model::MineSweeperBoard
+ DEBUG: bool
- gameBoard : MineSweeperField[,]
+ MineSweeperBoard()
+ MineSweeperBoard(Int)
+ MineSweeperBoard(MineSweeperBoard)
+ HasBomb(Int, Int) : bool
+ Reveal(Int, Int) : void
+ HideAllFields() : void
+ OnlyBombsLeft() : bool
+ ResizeBoard(Int) : void
+ PlaceBomb(Int, Int) : void
+ PlaceBombs(Int) : void
+ UpdateFieldValues() : void
+ SetupBoard(Int) : void
<<property>>
+ BoardSize() : Int
EventArgs
Model::EventArguments::MineSweeperGameOverEventArgs
+ MineSweeperGameOverEventArgs(PlayerEnum, Boolean)
<<property>>
+ LastPlayer() : PlayerEnum { get; private set; }
+ GameTied() : bool { get; private set; }
EventArgs
Model::EventArguments::MineSweeperRefreshBoardEventArgs
+ MineSweeperRefreshBoardEventArgs()
Model::MineSweeperField
+ DEBUG: bool
+ MineSweeperField()
+ MineSweeperField(bool)
+ MineSweeperField(MineSweeperField)
+ Reveal() : void
+ Hide() : void
+ PlaceBomb() : void
<<property>>
+ Value() : Int
+ Revealed() : bool
+ HasBomb() : bool
Persistence::MineSweeper2PGameState
+ MineSweeper2PGameState(MineSweeperBoard, PlayerEnum)
+ PrintHeaderData() : String
+ PrintBoard() : String
<<property>>
+ GameBoard() : MineSweeperBoard { get; private set; }
+ CurrentPlayer() : PlayerEnum { get; private set; }
+ BoardSize() : Int
- dataAccess
dataAccess
<<interface>>
Persistence::IMineSweeperDataAccess
+ LoadAsync(String) : Task<MineSweeper2PGameState>
+ SaveAsync(String, MineSweeper2PGameState) : Task
Persistence::MineSweeperDroidDataAccess
<<async>>
+ LoadAsync(String) : Task<MineSweeper2PGameState>
+ SaveAsync(String, MineSweeper2PGameState) : Task
- store
<<interface>>
Persistence::IStore
+ GetFiles() : Task<IEnumerable<String>>
+ GetModifiedTime(String) : Task<DateTime>
Persistence::AndroidStore
+ GetFiles() : Task<IEnumerable<String>>
+ GetModifiedTime(String) : Task<DateTime>
- storedGameBrowserModel
- _store
Model::StoredGameBrowserModel
- _store : IStore
+ StoredGameBrowserModel(IStore)
<<property>>
+ StoredGames: List<StoredGameModel>{ get; private set; }
<<event>>
+ StoreChanged: EventHandler
<<async>>
+ UpdateAsync() :Task
Model::StoredGameModel
<<property>>
+ Name: String { get; set; }
+ Modified: DateTime { get; set; }
- storedGameBrowserViewModel
ViewModel::StoredGameBrowserViewModel
- _model: StoredGameBrowserModel
+ StoredGameBrowserViewModel(StoredGameBrowserModel)
- UpdateStoredGames() : void
- Model_StoreChanged(object, EventArgs)
<<property>>
+ StoredGame: ObservableCollection<StoredGameViewModel> {get; private set;}
+ NewSaveCommand() : DelegateCommand
<<event>>
+GameLoading() : EventHandler <StoredGameEventArgs>
+GameSaving() : EventHandler <StoredGameEventArgs>
EventArgs
ViewModel::StoredGameEventArgs
+ StoredGameEventArgs()
<<property>>
+ Name() : String { get; set; }
- viewModel
INotifyPropertyChanged
ViewModelBase
ViewModelBase()
OnPropertyChanged(String): void
«event»
+ PropertyChanged() :PropertyChangedEventHandle
ViewModel::MineSweeper2PViewModel
- model: MineSweeper2PModel
- currentPlayer: PlayerEnum
- boardSize: int
- canSave: bool
- fields: ObservableCollection<MineSweeperFieldViewModel>
+ MineSweeper2PViewModel(MineSweeper2PModel)
+ NewGame(int) : void
+ LoadGame() : void
- Reveal(Int) : void
+ ForceRefresh() : void
- Model_GameOver(object, MineSweeperGameOverEventArgs)
- Model_RefreshTable(object, MineSweeperRefreshBoardEventArgs)
- OnNewGame() : void
- OnLoadGame() : void
- OnSaveGame() : void
- OnGameOver(MineSweeperGameOverEventArgs) : void
<<property>>
+ CurrentPlayer() : PlayerEnum
+ BoardSize() : Int
+ CanSave() : bool
+ Fields(): ObservableCollection<MineSweeperFieldViewModel>
+ NewGameCommand() : DelegateCommand
+ SaveGameCommand() : DelegateCommand
+ LoadGameCommand() : DelegateCommand
<<event>>
+NewGameEvent() : EventHandler
+LoadGameEvent() : EventHandler
+SaveGameEvent() : EventHandler
+ GameOverEvent() :EventHandler<MineSweeperGameOverEventArgs>
- model
ViewModel::MineSweeperFieldViewModel
- fieldValue : Int
- revealed : bool
- hasBomb: bool
- isEnabled: bool
- text : string
+ MineSweeperFieldViewModel(int, int int, DelegateCommand)
+ changeText() : String
+ MineSweeperField(MineSweeperField)
+ Reveal() : void
+ Hide() : void
+ PlaceBomb() : void
<<property>>
+ Value() : Int
+ Number() : Int
+ X() : Int
+ Y() : Int
+ Revealed() : bool
+ HasBomb() : bool
+ IsEnabled() : bool
+ Text() : string
+ RevealCommand() : DelegateCommand
ICommand
DelegateCommand
- _execute :Action (readOnly)
- _canExecute :Func (readOnly)
+ DelegateCommand(Action)
+ DelegateCommand(Func, Action)
+ CanExecute(Object): Boolean
+ Execute(Object): void
+ RaiseCanExecuteChanged(): void
«event»
+ CanExecuteChanged() :EventHandler
- _model
ViewModel::StoredGameViewModel
- _name: String
- _modified: DateTime
+ StoredGameViewModel()
<<property>>
+ Name: String {get; set;}
+ Modified: DateTime {get; set;}
+ SaveGameCommand() : DelegateCommand
+ LoadGameCommand() : DelegateCommand

Tesztelés:

- **MineSweeperField** tesztelése
  - **TestMineSweeperFieldValue():** mező értéknek lekérdezének és értékadásának működése
  - **TestMineSweeperFieldIfRevealed():** mező felfedése és elrejtés helyes működése
  - **TestMineSweeperFieldIfHasBomb():** akna mezőre rakásának és lekérdezésének helyes működése
  - **TestMineSweeperFieldToString():** mező helyes szövegként való kiírása függően a mező állapotától és tartalmától
- **MineSweeperBoard** tesztelése
  - **TestBoardIndexing():** játéktábla indexelésének működése
  - **TestBoardSize():** játéktábla méretének helyes frissülése
  - **TestBoardHasBomb():** játéktábla akna lerakása és lekérése
  - **TestBoardReveal():** játéktábla mező(k) felfedése
  - **TestBoardPlaceBombs ():** aknák száma
  - **TestBoardOnlyBombsLeft():** csak aknák maradtak
  - **TestBoardUpdateFieldValues():** mező értékek helyesen frissülnek
  - **TestModelNextPlayer():** játékos váltás helyes történik
  - **TestModelNewGame():** új játék helyesen jön létre
  - **TestModelGameOver():** játék vége kiváltódik
  - **TestModelRevealField():** játékesemények helyesen váltódnak ki
  - **TestModelLoadGame():** játék helyesen töltődik be