3. Beadandó feladat dokumentáció

Készítette:
etimaly

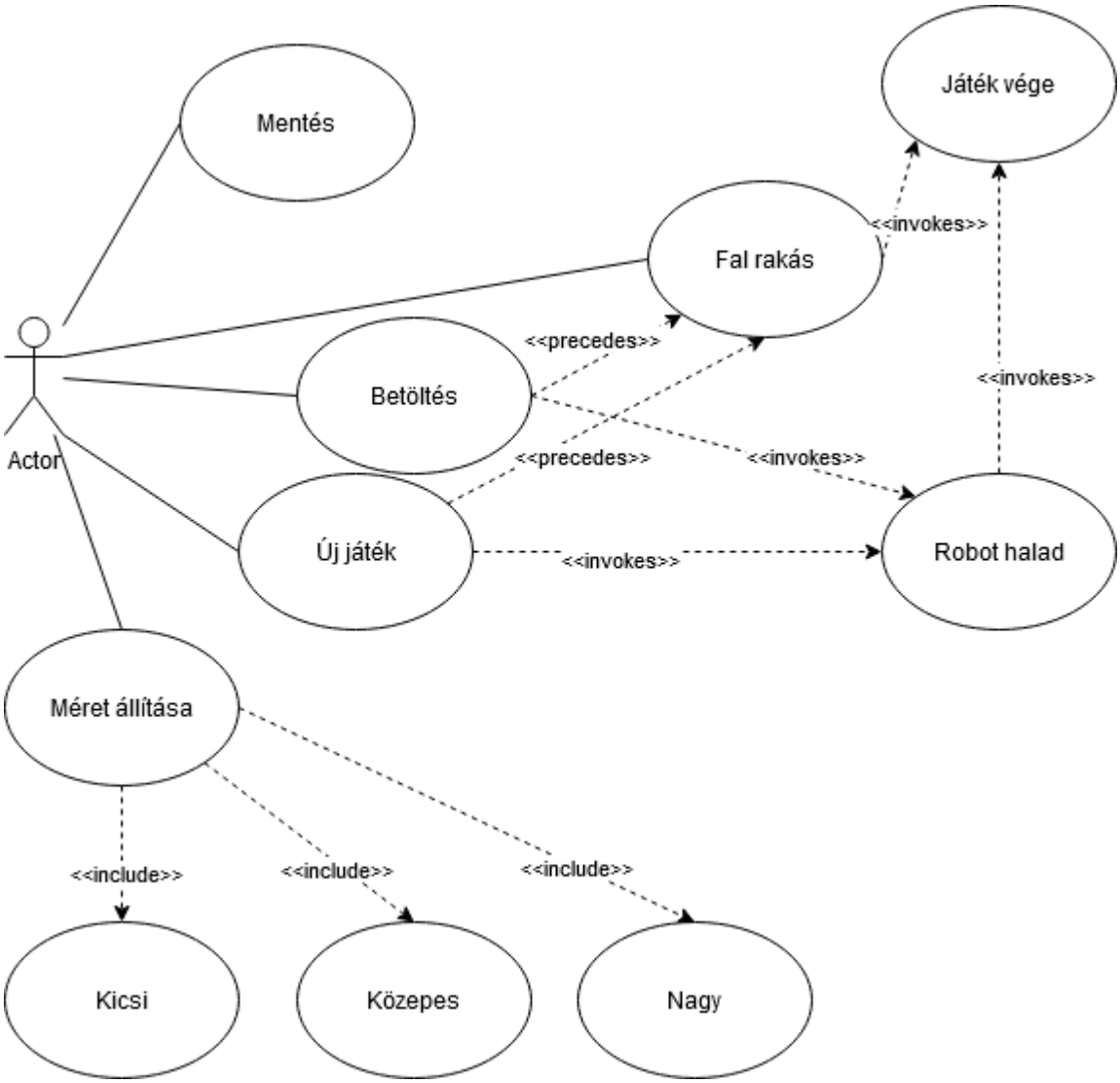
Feladat:

Elszabadult robot
Készítsünk programot, amellyel a következő játékot játszhatjuk.
Adott egy $n \times n$ mezőből álló játékpálya, amelyben egy elszabadult robot bolyong,
és a feladatunk az, hogy betereljük a pálya közepén található mágnes alá, és így
elkapjuk.
A robot véletlenszerű pozícióban kezd, és adott időközönként lép egy mezőt (vízszintesen, vagy függőlegesen) úgy, hogy általában folyamatosan előre halad
egészen addig, amíg falba nem ütközik. Ekkor véletlenszerűen választ egy új irányt, és arra halad tovább. Időnként még jobban megkergül, és akkor is irányt
vált, amikor nem ütközik falba.
A játékos a robot terelését úgy hajthatja végre, hogy egy mezőt kiválasztva falat
emelhet rá. A felhúzott falak azonban nem túl strapabíróak. Ha a robot ütközik a
fallal, akkor az utána eldől. A ledőlt falakat már nem lehet újra felhúzni, ott a robot később akadály nélkül áthaladhat.
A program biztosítson lehetőséget új játék kezdésére a pályaméret megadásával
(7 × 7, 11 × 11, 15 × 15), valamint játék szüneteltetésére (ekkor nem telik az idő,
nem lép a robot, és nem lehet mezőt se kiválasztani). Ismerje fel, ha vége a játéknak, és jelenítse meg, hogy milyen idővel győzött a játékos. A program játék
közben folyamatosan jelezze ki a játékidőt. Ezen felül szüneteltetés alatt legyen
lehetőség a játék elmentésére, valamint betöltésére.

Elemzés:

- A játékot 3 tábla méreten játszhatjuk: kicsi(7×7), közepes(11×11), nagy(15×15). A program indításkor egy kicsi táblát indít el.

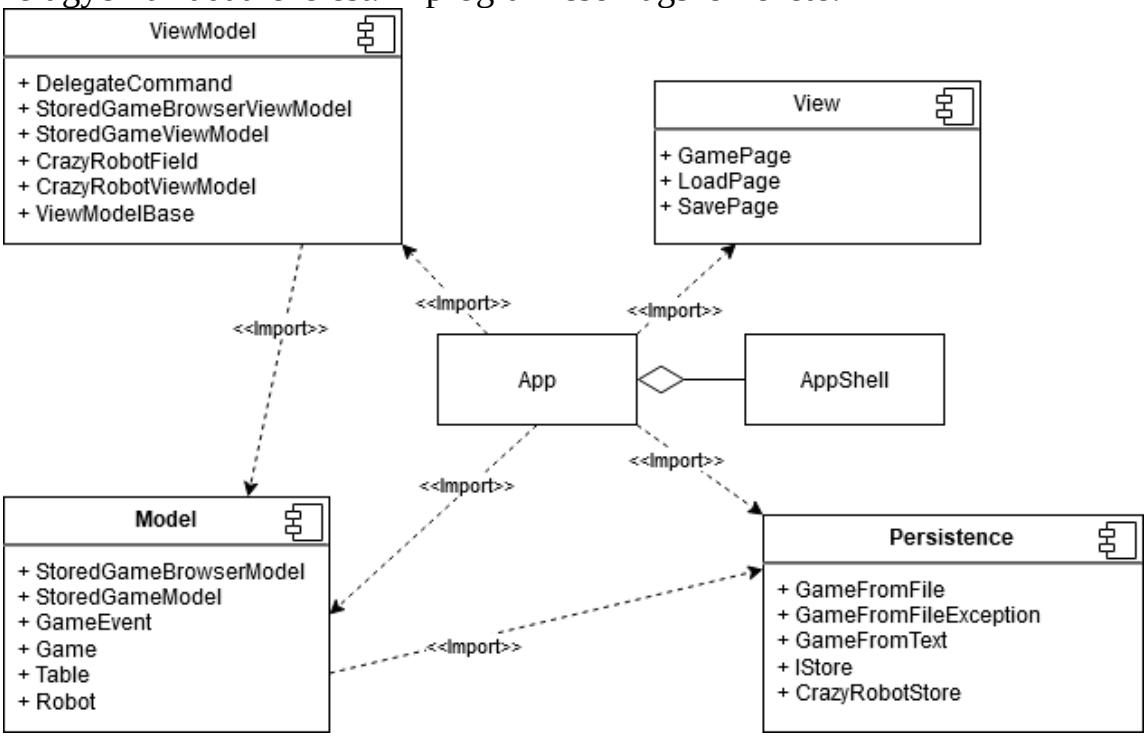
- A feladatot .NET MAUI alkalmazásként, elsődlegesen Windows és Android platformon valósítjuk meg. Az alkalmazás 3 lapbl fog állni. (játék, mentés, betöltés)
- Az ablakban egy menüt helyezünk el a következő menüpontokkal: File(New Game, Load, Save), Game(Small, Medium, Big), Pause. Az ablak alján egy menü sort helyezünk el, ami kijelzi az eltelt időt és a lerakott falak számát.
- A játéktáblát a megfelelő méret nagyságú nyomógombokból álló rács reprezentálja. A nyomógomb kattintásra pirosra változik, ezzel jelezve egy fal letételét és kikapcsolja a gombot a játék végéig. Az összetört falakat barna gombbal jelöljük és a robot helyzetét fekete gombbal.
- Amikor vége a játéknak (robot be lett terelve a tábla közepére vagy a tábla minden pontjára le lett rakva egy fal és mindegyik összetört) dialógusablak jön elő. Ekkor a mentés és szüneteltés funkciót kikapcsoljuk.
- A felhasználó esetek a következő ábrán látható:



-

Tervezés:

- Programszerkezet:
  - A programot **MVVM** architektúrában valósítjuk meg, ennek megfelelően **View**, **Model**, **ViewModel** és **Persistence** névtereket valósítunk meg az alkalmazáson belül. A program környezetét az alkalmazás osztály (**App**) végzi, amely pédányosítja a modellt, a nézetmodellt és a nézetet, biztosítja a kommunikációt, valamint felügyeli az adatkezelést. A program csomagszerkezete:



  - A program szerkezetét két projektre osztjuk implementációs megfontolásból: a **Persistence** és **Model** csomagok a program felületfüggetlen projektjében, míg a **ViewModel** és **View** csomag a WPF függő projektjében kap helyet.

- Perzisztencia:

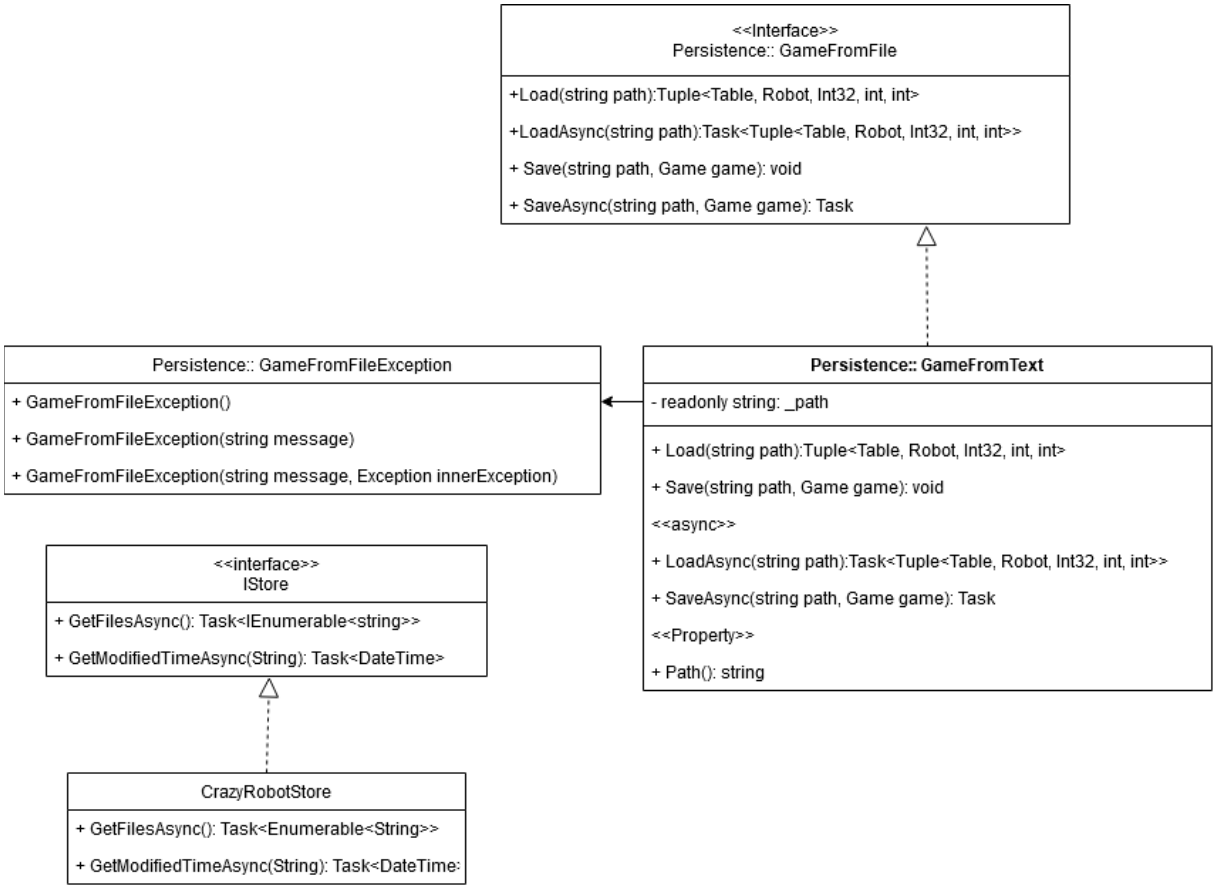
  - Az adatkezelés a robot és a táblával kapcsolatos információk tárolására, valamint a betöltés/mentés biztosítása.

  - A hosszú távú adatkezelés lehetőségeit a GameFromFile interfész biztosítja, ami lehetőséget ad mentésre (Save) és betöltésre (Load)

  - Az interfészt szöveges fájl alapú adatkezelésre a GameFromText osztály valósítja meg. A fájlkezelés során fellépő hibákat a GameFromFileException osztály kivétel jelzi.

- A program az adatokat szöveges fájlként tudja eltárolni, melyek **txt** kiterjesztésüek. Az adatokat játék vége elött bármikor el lehet menteni és új játékot betölteni a játék vége után is lehet.

- A fájl első sora megadja a tábla méretet, robot helyzetét, megőrülés esélyét, időt, falak száma, ledőlt falak száma és a robot iránya. A fájl többi része izomorf leképzése a táblának, azaz méretnek megfelelő sor darabszám és szóközökkel elválasztva 0,1,2 számok (0: üres, 1: fal, 2: lerombolt fal).



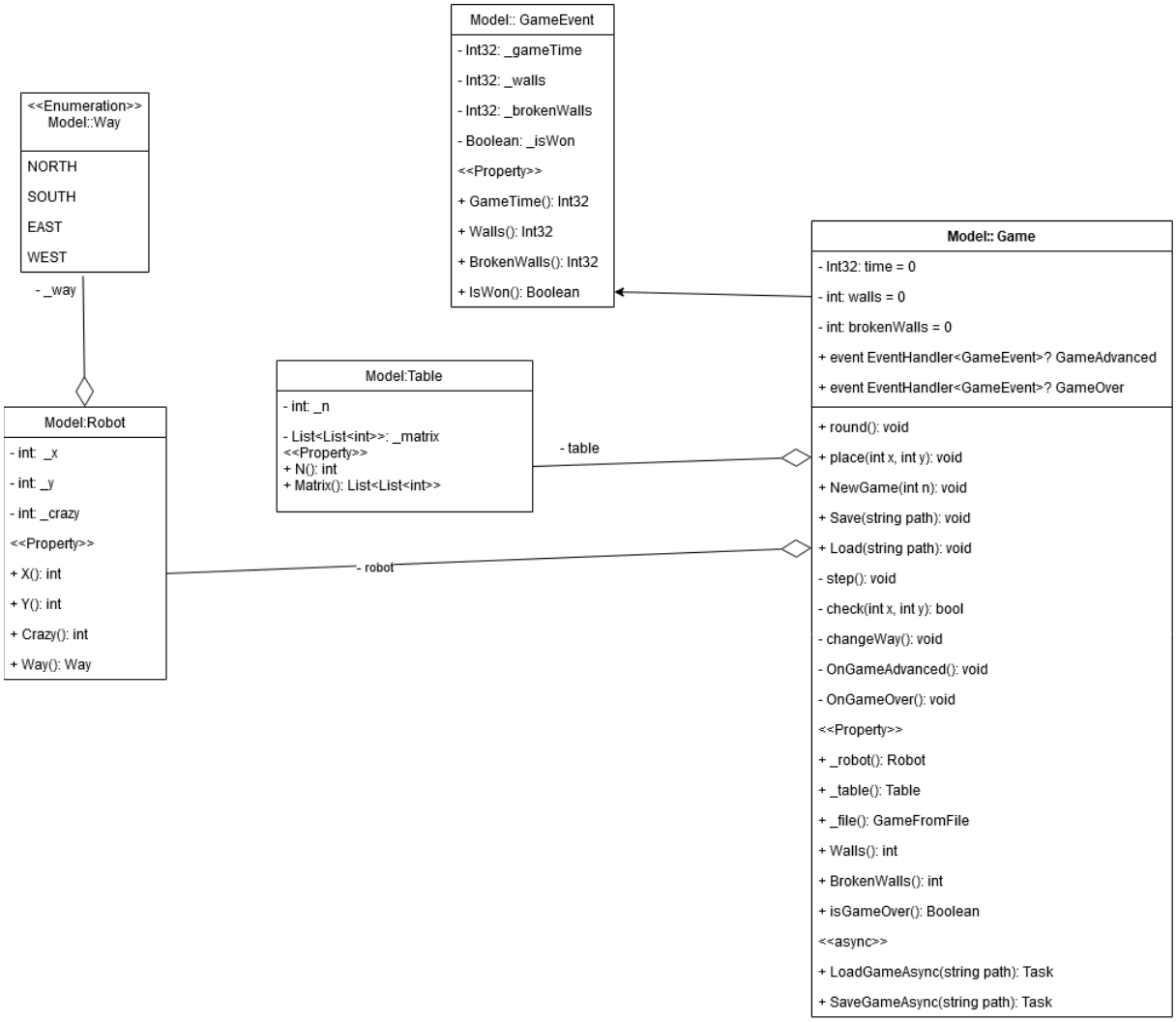
- Modell:

  - A modell lényegi részét a Game osztály valósítja meg, amely szabályozza a robot lépését, falak helyezését, valamint a játék egyéb paramétereit, úgymint az időt **(_time**), nem ledőlt falak száma (**_walls**), ledőlt falak száma(**_brokenWalls**). Az osztály lehetőséget ad új játék (**NewGame**), valamint egy kör lejátszására (**round**, léptetti a robotot és megnézi, hogy játéknak vége van-e és frissíti az adattagokat). Új játéknál megadható a tábla mérete is.

- A játékállapot változásáról a **GameAdvanced** esemény, míg a játék végéről
  a **GameOver** esemény tájékoztat. Az események argumentuma (**GameEvent**) tárolja a győzelem állapotát, a falak számát, ledőlt falak valamint a játékidőt.

- A modell pédányosításkor megkapja az adatkezelés felületét, amelyeknek segítségével lehetőséget ad betöltésre (**Load**) és mentésre (**Save**).

- A robot irányát felsorolási típuson át kezeljük.



- Nézetmodell:

  - A nézetmodell megvalósításához felhasználunk egy általános utasítás (**DelegateCommand**), valamint egy ős változás jelző (**ViewModelBase**) osztályt.

- A nézetmodell feladatait a **CrazyRobotViewModel** osztály látja el, amely parancsokat biztosít az új játék kezdéséhez, játék betöltéséhez, mentéséhez és megállításához. A parancsokhoz eseményeket kötünk, amelyek a parancs lefutását jelzik a vezérlőnek. A nézetmodell tárolja el a modell egy hivatkozását (**_model**), de csupán információkat két le tőle, illetve a tábla méretét szabályozza. Direkt nem avatkozik a játék futtatásába.

- A játékmező számára egy külön mezőt biztosítunk (**CrazyRobotField**), amely eltárolja a pozíciót, indexet, robot-e, fal-e vagy ledöntött fal-e, valamint a lépés parancsát (**StepCommand**). A mezőket egy felügyelt gyűjteménybe helyezzük a nézetmodellben (**Fields**).

- A tárolt játékállapotok egy-egy StoredGameViewModel példánnyal írhatók le. Ezek kollekcióját nem ágyazzuk be a fő nézetmodellbe (**CrazyRobotViewModel**), hanem betöltéskor és mentéskor dinamikusan előállítjuk.


CrazyRobotViewModel : ViewModelBase
- _model: Game
- Model_GameAdvanced(object, GameEvent): void
- Model_GameOver(object, GameEvent): void
- OnLoadGame(): void
- OnSaveGame(): void
- OnPauseGame(): void
- RefreshTable(): void
- StepGame(int): void
+ CrazyRobotViewModel(Game)
<<property>>
+ Fields(): ObservableCollection<CrazyRobotField>
+ GamePlacedWallsCount(): string
+ GameTime(): string
+ N(): int
+ PauseGameCommand(): DelegateCommand
+ SaveGameCommand(): DelegateCommand
+ LoadGameCommand(): DelegateCommand
+ NewGameCommand(): DelegateCommand
+ SmallCommand(): Boolean
+ MediumCommand(): Boolean
+ BigCommand(): Boolean
<<event>>
+ PauseGame(): EventHandler
+ NewGame(): EventHandler
+ LoadGame(): EventHandler
+ SaveGame(): EventHandler
ViewModelBase : INotifyPropertyChanged
OnPropertyChanged(String) : void
ViewModelBase()
<<event>>
+ PropertyChanged(): PropertyChangedEventHandler
CrazyRobotField
- _wall: bool
- _brokenWall: bool
- _robot: bool
- _isLocked: bool
<<property>>
+ Wall(): bool
+ BrokenWall(): bool
+ Robot(): bool
+ IsLocked(): bool
+ X(): int
+ Y(): int
+ Number(): int
+ StepCommand(): DelegateCommand
*
DelegateCommand : ICommand
- _canExecute: Func<Object, Boolean> {readOnly}
- _execute:Action<Object> {readOnly}
+ CanExecute(Object): Boolean
+ DelegateCommand(Action<Object>)
+ DelegateCommand(Func<Object, Boolean>, Action<Object>)
+ Execute(Object): void
+ RaiseCanExecuteChanged(): void
<<event>>
+ CanExecuteChanged(): EventHandler

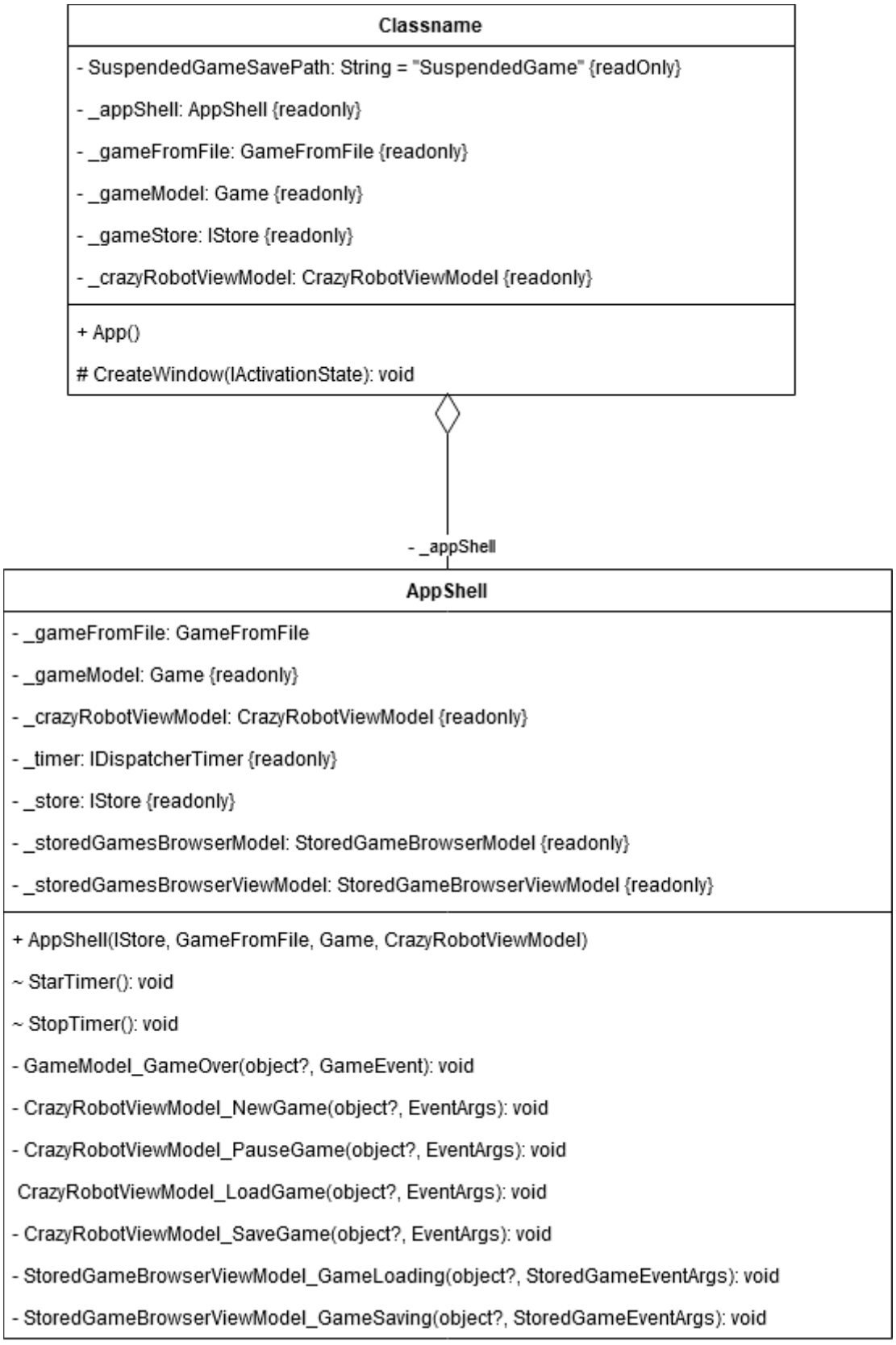
- Nézet:
  - A nézetet navigációs lapok segítségével építjük fel.
  - A GamePage osztály tartalmazza a játéktáblát, amelyet egy Grid segítségével valósítjuk meg, amelyben Button elemeket helyezünk el.
  - A LoadPage és a SavePage szolgál egy játék betöltésére vagy elmentésére.
- Környezet:

  - Az **App** osztály feladata az egyes rétegek pédányosítása, összekötése, a nézetmodell, valamint a modell eseményeinek lekezelése, és ezáltal a játék, az adatkezelés, valamint a nézetek szabályozása.

  - A CreateWindow metódus felüldefiniálásával kezeljük az alkalmazás életciklusát a megfelelő eseményekre történő feliratkozással. Így az alkalmazás felfüggesztéskor (**Stopped**) elmentjük az aktuális játékállást (**SuspendedGame**), míg folytatáskor vagy újraindításkor (**Activated**) pedig folytatjuk, amennyiben történt mentés.

  - Az alkalmazás lapjait egy AppShell keretben helyezzük el. Ez az osztály felelős a lapok közötti navigációk megvalósításáért.

  - A játék léptetéséhez tárol egy időzítőt is (**_timer**), amelynek állítását is szabályozza az egyes funkciók hatására.


Classname
- SuspendedGameSavePath: String = "SuspendedGame" {readOnly}
- _appShell: AppShell {readonly}
- _gameFromFile: GameFromFile {readonly}
- _gameModel: Game {readonly}
- _gameStore: IStore {readonly}
- _crazyRobotViewModel: CrazyRobotViewModel {readonly}
+ App()
# CreateWindow(IActivationState): void
- _appShell
AppShell
- _gameFromFile: GameFromFile
- _gameModel: Game {readonly}
- _crazyRobotViewModel: CrazyRobotViewModel {readonly}
- _timer: IDispatcherTimer {readonly}
- _store: IStore {readonly}
- _storedGamesBrowserModel: StoredGameBrowserModel {readonly}
- _storedGamesBrowserViewModel: StoredGameBrowserViewModel {readonly}
+ AppShell(IStore, GameFromFile, Game, CrazyRobotViewModel)
~ StarTimer(): void
~ StopTimer(): void
- GameModel_GameOver(object?, GameEvent): void
- CrazyRobotViewModel_NewGame(object?, EventArgs): void
- CrazyRobotViewModel_PauseGame(object?, EventArgs): void
CrazyRobotViewModel_LoadGame(object?, EventArgs): void
- CrazyRobotViewModel_SaveGame(object?, EventArgs): void
- StoredGameBrowserViewModel_GameLoading(object?, StoredGameEventArgs): void
- StoredGameBrowserViewModel_GameSaving(object?, StoredGameEventArgs): void

Tesztelés:

- A modell funkcionalitása egységtesztek segítségével lett ellenőrizve a UnitTest1 osztályban.
- Az alábbi tesztek lettek megvalósítva:
    - **NewGameSmallTest, NewGameMediumTest, NewGameBigTest**: Új játék indítás, a mezők kitöltése, falak száma, lerombolt falak száma, tábla mérete és az idő függvényében.
    - **PlaceTest, PlaceSamePlaceTest**: Fal lerakás ellenőrzése egy üres helyre és fal lerakása egy foglalt helyre.
    - **RoundWithNoWallTest, RoundWithWallTest**: Egy kör lefolyásának ellenőrzése (robot helyzete és iránya, falak száma, lerombolt falak száma, idő). Fal ütközés esetén irányt nem ellenőrzünk, mert az véletlenszerű. (a megőrülés esélye 0 vagy különben a tesztelés nem lehetséges)
    - **LoadTest**: Leteszteli a load függvény helyességét (modell betöltése).