

Beadandó feladat dokumentáció

Készítette: Nick Mónika (UHH5KS)

E-mail: nickmonkavera@gmail.com

Feladat:

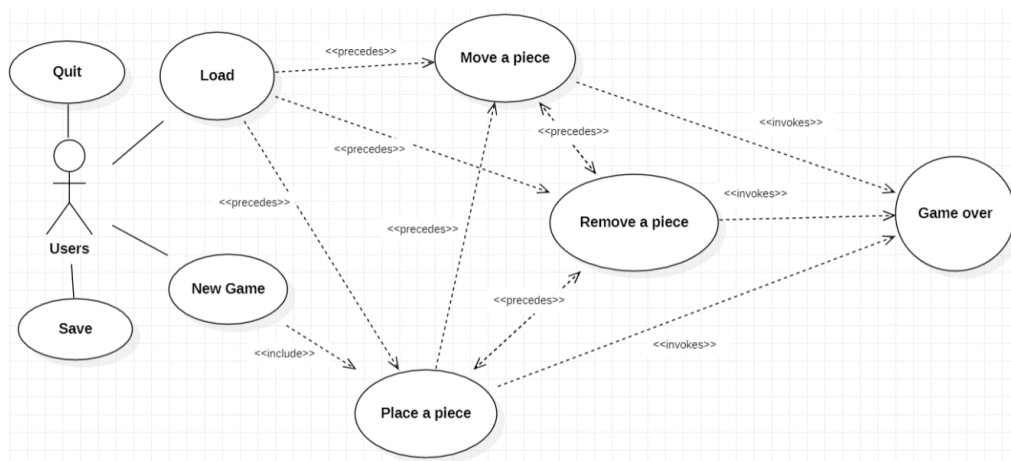
22. Malom

Készítsünk programot, amellyel a következő két személyes játékot játszhatjuk. A malomjátékban két játékos egy 24 mezőből álló speciális játéktáblán játszhatja, ahol a mezők három egymásba helyezett négyzetben helyezkednek (mindegyikben 8, a sarkoknál és a felezőpontoknál), melyek a felezőpontok mentén össze vannak kötve. Kezdetben a tábla üres, és felváltva helyezhetik el rajta bábuikat az üres mezőkre. Az elhelyezés után a játékosok felváltva mozgathatják bábuikat a szomszédos (összekötött) mezőkre. Amennyiben egy játékos nem tud mozgatni, akkor passzolhat a másik játékosnak. Ha valakinek sikerül 3 egymás melletti mezőt elfoglalnia (azaz malmot alakít ki, rakodás, vagy mozgatás közben), akkor leveheti az ellenfél egy általa megjelölt bábuját (kivéve, ha az egy malom része). Az a játékos veszít, akinek először megy 3 alá a bábuk száma a mozgatási fázis alatt. A program biztosítson lehetőséget új játék kezdésére, mentésre és betöltésére. Ismerje fel, ha vége a játéknak, és jelenítse meg, melyik játékos győzött.

Elemzés:

- A játékban három fázis van: elhelyezés, mozgatás, ugrálás. A program indításkor elhelyezéssel kezdünk, és automatikusan új játékot indul.
- A feladatot egyablakos asztali alkalmazásként Windows Forms grafikus felülettel valósítjuk meg.
- Az ablakban elhelyezünk egy menüt a következő menüpontokkal: File (Load Game, New Game, Save, Exit). Az ablak alján megjelenítünk egy státuszsort, amely azt jelzi, hogy ki jön, vagy azt, ha törlés következik, vagy, azt is, ha nem tudunk a kiválasztott mezőre lépni, vagy onnan máshova lépni, vagy törölni.
- A játéktáblát egy panel és azon 24 nyomógomb reprezentálja. A nyomógomb egérekattintás hatására megváltoztathatja a színét vagy nem. Attól függ, hogy az adott játékos helyezhet ide bábut, vagy elmozgathatja-e az adott mezőről.
- A játék kiírja a státuszsortba, amikor vége a játéknak és a bábukat már ilyenkor nem lehet mozgatni. Szintén dialógusablakokkal végezzük el a mentést, illetve betöltést, a fájlneveket a felhasználó adja meg.

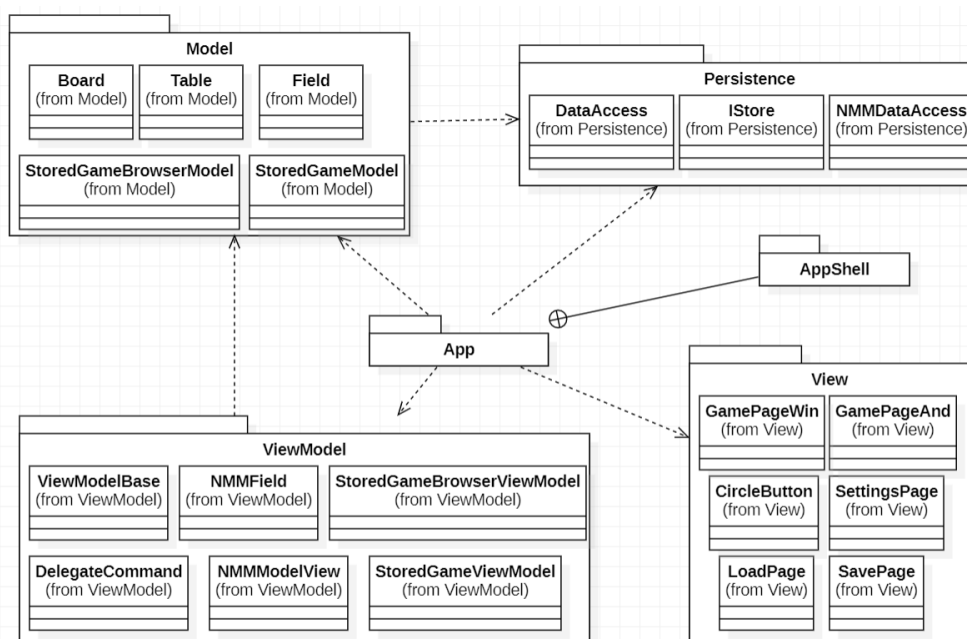
Use Case Diagramm:



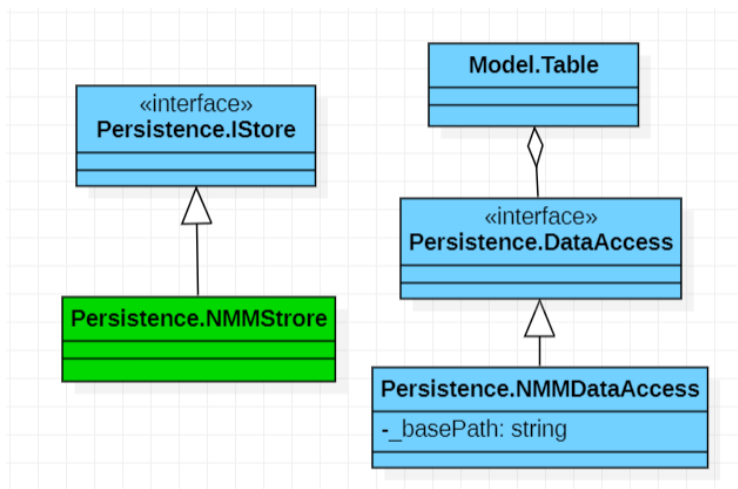
Tervezés:

Programszerkezet:

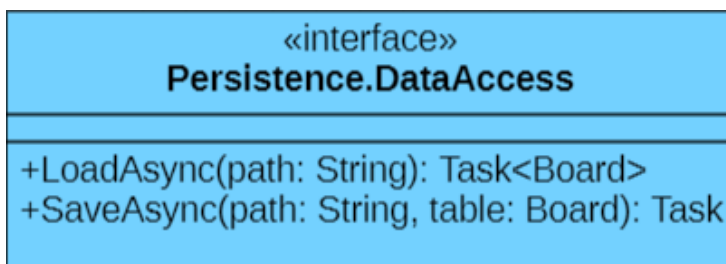
- A programot MVVM architektúrában valósítjuk meg. A megjelenítés a NMM.View, a megjelenítés modellje a NMM.ViewModel, a modell NMMModel.Model, míg a perzisztencia a NMMModel.Persistence névtérben helyezkedik el.
- A szoftvert két projektből építjük fel: a modellt és a perzisztenciát tartalmazó osztálykönyvtárból (.NET Standard Class Library), valamint a .NET MAUI többplatformos projektből, amelyet Windows és Android operációs rendszere is le tudunk fordítani.
- A megvalósításból külön építjük fel a játék, illetve a betöltés és mentés funkciót, valamennyi rétegben. Utóbbi funkcionalitást újrahasznosítjuk egy korábbi projektből, így nem igényel újabb megvalósítást.
- A program vezérlését az alkalmazás osztály (App) végzi, amely példányosítja a modellt, a nézetmodellt és a nézetet, biztosítja a kommunikációt, valamint felügyeli az adatkezelést.
- A program csomagdiagramja az ábrán látható:



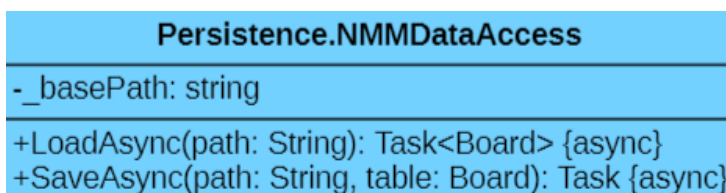
Perzisztencia:



- Az adatkezelés feladata a malom táblával kapcsolatos információk tárolása, valamint a betöltés/mentés biztosítása.
- A hosszú távú adattárolás lehetőségeit az DataAccess interfész adja meg, amely lehetőséget ad a tábla betöltésére (LoadAsync), valamint mentésére (SaveAsync). A műveleteket hatékonysági okokból aszinkron módon valósítjuk meg.

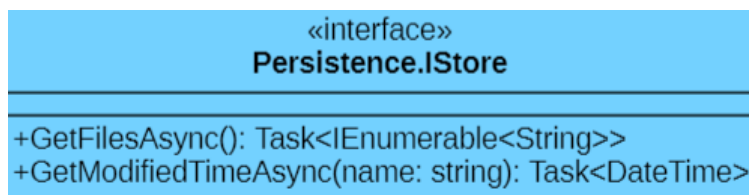


- Az interfészt szöveges fájl alapú adatkezelésre a NMMDataAccess osztály valósítja meg. A fájlkezelés során fellépő hibákat a NMMDDataException kivétel jelzi. A program az adatokat szöveges fájlként tudja eltárolni, amelyeket egy megadott könyvtárban (_basePath) helyez el. Ez majd az alkalmazás platformfüggő saját adatkönyvtára lesz.

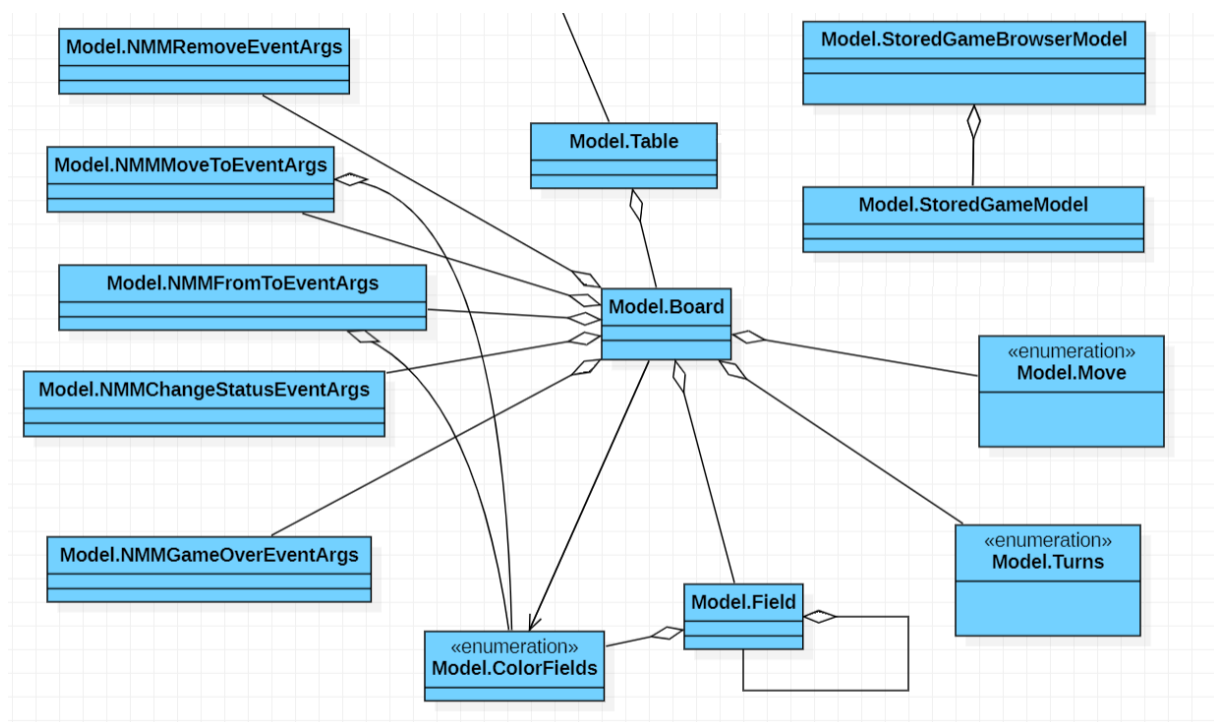


- A program az adatokat szöveges fájlként tudja eltárolni, melyek az txt kiterjesztést kapják. Ezeket az adatokat a programban bármikor be lehet tölteni, illetve ki lehet menteni az aktuális állást.
- A fájl első sora megadja a tábla méretét, hogy hol tartunk az elhelyezés fázisban, mennyi kék bábu van, mennyi zöld bábu van, ki következik, törlés következik-e, egy bábu kiválasztása vagy mozgatása következik-e, és melyik mezőről kell mozgatni. A második sor az összes mező színét jelzi (0-Green, 1-Blue, 2-Transparent).

- IStore interfacet a fájlok mentéséhez és betöltéséhez fogjuk felhasználni.



Modell:



- A Board osztály egy malom táblát biztosít, ami 24 mezőből áll, és a mezők a szomszédjaikra mutatnak. Az osztályban a játékosok lépései és az adott játékfázis szerint változnak a mező színei. Az osztály számolja a bábuk számát, és azt is, hogy a játéknak mikor van vége. Egy mezőre kattintáskor mindig a move függvényt kell meghívni. Új játékkezdés esetén meg a newGame-t. moveTo és moveFromTo függvények mozgatják a bábukat. threeNextToEachOther ellenőrzi, ha malom van, allNextToEachOther pedig, ha egy játékos összes bábuja malomban van. canMove, hogy tud-e a bábu mozogni, canPlayerMove, hogy tud-e a játékos összes bábuja mozogni. removeOne törli a bábút, ha tudja. end véget vet a játéknak.

Model.Board
-green0: int -blue1: int -turns: int -player_: Turns -remove: bool -moves: Move -moveFrom: int -end_: bool -fields: Field<> +green0Value: int {property} +blue1Value: int {property} +turnsValue: int {property} +player_Value: Turns {property} +removeValue: bool {property} +movesValue: Move {property} +moveFromValue: int {property} +FieldsColor: ColorFields<> {property} +MoveTo_: EventHandler<NMMMoveToEventArgs> {event} +FromTo_: EventHandler<NMMFromToEventArgs> {event} +Remove_: EventHandler<NMMRemoveEventArgs> {event} +End_: EventHandler<NMMGameOverEventArgs> {event} +ChangeStatus_: EventHandler<NMMChangeStatusEventArgs> {event}
+Board() +Board(colors: ColorFields<>, green0: int, blue1: int, turns: int, remove: bool, player_: Turns, moves: Move, moveFrom: int) +move(ind: int): void +newGame(): void -toString(p: Turns): string -switchPlayer(p: Turns): Turns -equals(a: Model.ColorFields, b: Turns): bool -toColorFields(a: Turns): Model.ColorFields -moveTo(ind: int, player: Turns): bool -moveFromTo(ind: indFrom, int: indTo, player: Turns): bool -canMove(ind: ind): bool -canPlayerMove(): bool -removeOne(ind: int, player: Turns): bool -threeNextToEachOther(ind: ind): bool -allNextToEachOther(player: Turns): bool -ChangeStatus(status: string): void -Remove(ind: ind): void -GameOver(winner: string): void -MoveTo(color: ColorFields, ind: ind): void -FromTo(color: ColorFields, from: ind, to: ind): void

- Field osztály egy mezőt jelez, ami rámutat a szomszédjaira, és a saját színét tárolja.

Model.Field
+player: Model.ColorFields -up_: Field -down_: Field -right_: Field -left_: Field +Up: Field {property} +Down: Field {property} +Left: Field {property} +Right: Field {property}
+Field()

- A Table osztály tartalmaz egy Board osztályt, így betöltés (LoadGameAsync) és mentés (SaveGameAsync) esetén le tudja cserélni a malomtablát.

Model.Table
-IdataAccess: DataAccess +board: Board
+Table(idataAccess: DataAccess) +SaveGameAsync(path: string): Task {async} +LoadGameAsync(path: string): Task {async}

- A modellben vannak felsorolók is, a színek beállításához és a mozgathoz.

«enumeration» Model.Turns	«enumeration» Model.Move	«enumeration» Model.ColorFields
GREEN BLUE	CHOOSE PLACE	GREEN BLUE TRANSPARENT

- A Board()-nak 5 eseménye van: MoveTo_, FromTo_, Remove_, ChangeStatus_ és End_. Ezekre tud a nézet feliratkozni.

Model.NMMFromToEventArgs
- _color: ColorFields - _transplnd: int - _ind: int
+NMMFromToEventArgs(color: ClolorFiledcs, ind: int, transplnd: int) +color(): ColorFields {property} +transplnd(): int {property} +ind(): int {property}

Model.NMMGameOverEventArgs
- _winner: string
+NMMGameOverEventArgs(winner: string) +winner(): string {property}

Model.NMMChangeStatusEventArgs
- _status: string
+NMMChangeSatusEventArgs(status: string) +status(): string {property}

Model.NMMRemoveEventArgs

- _transplnd: int

+NMMRemoveEventArgs(transplnd: int)

+transplnd(): int {property}

Model.NMMMoveToEventArgs

- _color: ColorFields

- _ind: int

+NMMMoveToEventArgs(color: ColorFields, ind: int)

+color(): ColorFields {property}

+ind(): int {property}

- Ide tartozik még a StoredGameBrowserModel, amiből a tárolt játékok keresőjének modellje és a StoredGameModel, ami egy tárolt játék modellje.

Model.StoredGameBrowserModel

- _store: IStore

+StoreCganged(): EventHandler {property}

+UpdateAsync(): Task {async}{event}

+StoredGames(): List<StoredGameModel> {property}

-OnSavesChanged(): void

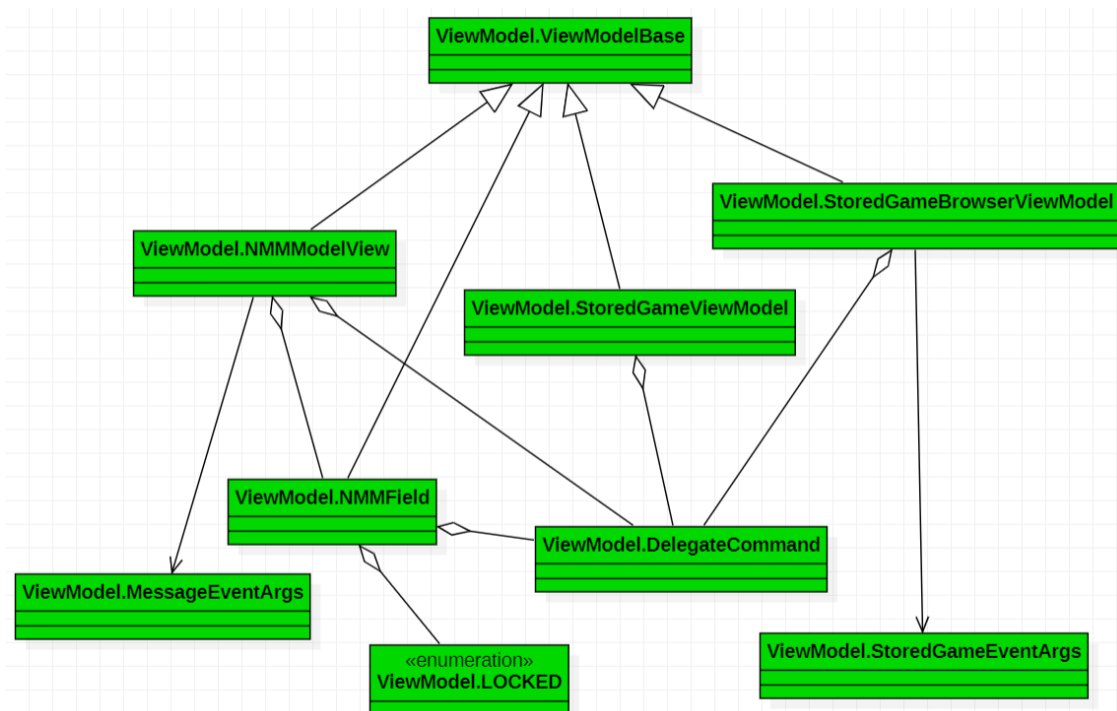
+StoredGameBrowserModel(IStore)

Model.StoredGameModel

+Name(): string {property}

+Modified(): DateTime {property}

Nézetmodell:



- A nézetmodell megvalósításához felhasználtunk egy általános utasítás (MyCommand), valamint egy ős változásjelző (ViewModelBase) osztályt.

```
ViewModel.DelegateCommand
- _execute: Action<Object> {readOnly}
- _canExecute: Func<Object; Boolean> {readOnly}
+DelegateCommand(execute: Action<Object>)
+DelegateCommand(canExecute: Func<Object; Boolean>, execute: Action<Object>)
+CanExecute(Object): Boolean
+Execute(Object): void
+RaiseCanExecuteChanged(): void
+CanExecuteChanged(): EventHandler
```

```
ViewModel.ViewModelBase
#ViewModelBase()
#OnPropertyChanged(String)
+PropertyChanged(): PropertyChangedEventHandler
```

- A nézetmodell feladatait a NMMModelView 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 (table_), de csupán információkat kér le tőle. Direkt nem avatkozik a játék futtatásába.

ViewModel.NMMModelView
+Fields: ObservableCollection<NMMField> +table_: Table -status: string
+ResumeCommand(): DelegateCommand {property} +NewGameCommand(): DelegateCommand {property} +LoadGameCommand(): DelegateCommand {property} +SaveGameCommand(): DelegateCommand {property} +ExitGameCommand(): DelegateCommand {property} +LoadGame(): EventHandler {property} +SaveGame(): EventHandler {property} +ExitGame(): EventHandler {property} +ResumeGame(): EventHandler {property} +ShowMessage(): EventHandler<MessageEventArgs> {property} +Fields(): ObservableCollection<NMMField> {property} +Status(): string {property} -OnLoadGame(): void -OnSaveGame(): void -OnExitGame(): void -OnResume(): void -OnPushButton(object): void -Model_FromTo(object, NMMFromToEventArgs): void -Model_MoveTo(object, NMMMoveToEventArgs): void -Model_Remove(object, NMMRemoveEventArgs): void -Model_ChangeStatus(object, NMMChangeStatusEventArgs): void -addFields(): void +OnNewGame(): void +Refresh(): void +NMMModelView(Table)

- A játékmező számára egy külön mezőt biztosítunk (NMMField), amely eltárolja a pozíciót, színét, méretét, indexét és parancsát (PushCircleButton). A mezőket egy felügyelt gyűjteménybe helyezzük a nézetmodellbe (Fields).

ViewModel.NMMField
-color: ColorFields -ind: int -locked: LOCKED
+Color(): ColorFields {property} +Ind(): int {property} +Locked(): LOCKED {property} +PushCircleButton(): DelegateCommand {property} +convert(int): int +convertBack(int): int

- Üzenetek küldésére használhatja a nézetmodell a MessegeEventArgs-ot.

ViewModel.MessageEventArgs
-str: string
+MessageEventArgs(string)
+_str(): string {property}

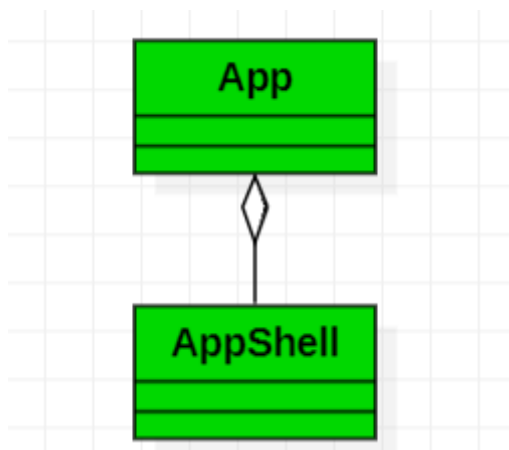
- Ide tartozik még a StoredGameBrowserViewModel, amiből a tárolt játékokat lehet kikeresni és a StoredGameViewModel, ami egy tárolt játék.

ViewModel.StoredGameBrowserViewModel
-_model: StoredGameBrowserModel
+StoredGameBrowserViewModel(StoredGameBrowserModel)
+GameLoading(): EventHandler<StoredGameEventArgs> {event}
+GameSaving(): EventHandler<StoredGameEventArgs> {event}
+NewSaveCommand(): DelegateCommand {property}
+StoredGames(): ObservableCollection<StoredGameViewModel> {property}
-UpdateStoredGames(): void
-Model_StoreChanged(object, EventArgs): void
-OnGameLoading(string): void
-OnGameSaving(string): void

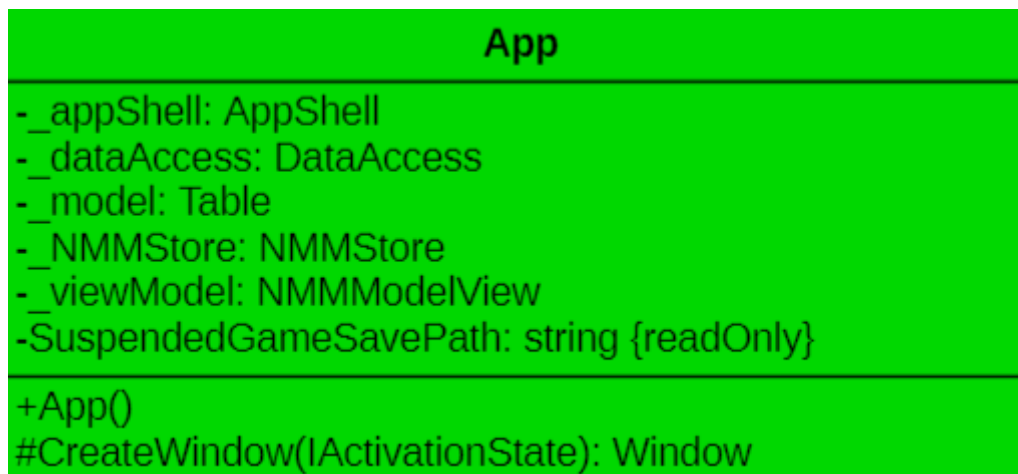
ViewModel.StoredGameViewModel
-_name: string
-_modified: DateTime
+Name(): string {property}
+Modified(): DateTime {property}
+LoadGameCommand(): DelegateCommand {property}
+SaveGameCommand(): DelegateCommand {property}

ViewModel.StoredGameEventArgs
+Name(): string {property}

Nézet:



- A nézetet navigációs lapok segítségével építjük fel.
- A GamePageAnd és GamePageWin osztályok tartalmazzák a játéktáblát, amelyet egy Grid segítségével valósítunk meg, amelyben Button elemeket helyezünk el.
- A SettingsPage osztály tartalmazza a betöltés, mentés, új játék, és folytatás gombjait.
- A LoadPage és a SavePage szolgál egy létező játékállapot betöltésére, illetve egy új mentésére.
- Az App osztály feladata az alkalmazás vezérlése, a rétegek példányosítása és az események feldolgozása.



- 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.

AppShell

```
- _model: Table
- _viewModel: NMMModelView
- _dataAccess: DataAccess
- _gamePage: ContentPage
- _settingsPage: SettingsPage
- _store: IStore
- _storedGameBrowserModel: StoredGameBrowserModel
- _storedGameBrowserViewModel: StoredGameBrowserViewModel

+AppShell(IStore, DataAccess, Table, NMMModelView)
-Model_GameOver(object, NMMGameOverEventArgs): void {async}
-ViewModel_ShowMessage(object, MessageEventArgs): void {async}
-ViewModel_ExitGame(object, System.EventArgs): void {async}
-ViewModel_ResumeGame(object, EventArgs): void {async}
-ViewModel_LoadGame(object, EventArgs): void {async}
-ViewModel_SaveGame(object, EventArgs): void {async}
-StoredGameBrowserViewModel_GameLoading(object, StoredGameEventArgs): void {async}
-StoredGameBrowserViewModel_GameSaving(object, StoredGameEventArgs): void {async}
```

Tesztelés:

A modell funkcionalitása egységtesztek segítségével lett ellenőrizve a TestNMM osztályban.

- Az alábbi tesztesetek kerültek megvalósításra és következőket ellenőrzik le:
 1. TestConstruct(): Létrejön-e a tábla?
 2. TestPlaceOne(): Le tudunk-e helyezni egy bábut?
 3. TestTurns(): Az első fázis lépései csökkennek-e?
 4. TestSwitchPlayer(): Váltunk-e a két játékos között?
 5. TestCannotPlace(): Nem tudunk lehelyezni bábut?
 6. TestRemovePhase(): Törlés következik-e?
 7. TestRemove(): Kitöröltük-e a bábut?
 8. TestRemoveDecrease(): Csökkent-e a báruk száma?
 9. TestRemovingIfAllNextToEachOther(): Töröl-e, akkor is, ha minden bábu 3-ast alkot?
 10. TestSecondPhaseChoosing(): A második fázisban választottunk-e bábut?
 11. TestSecondPhaseMoving(): A második fázisban mozgattunk-e bábut?
 12. TestSecondPhaseCannotMoveToTransparent(): Nem tudunk mozgatni, mert messze van.
 13. TestSecondPhaseCannotMoveToOwnField(): Nem tudunk mozgatni saját mezőre.
 14. TestSecondPhaseCannotMoveToOtherPlayersField(): Nem tudunk mozgatni másik játékos mezőjére.
 15. TestHaveToPass(): Passzolnia kell-e a játékosnak?
 16. TestThirdPhase(): Lehet-e ugrálni a 3. fázisban?
 17. TestEndGame(): Vége van-e a játéknak?
 18. TestNewGame(): Tudunk-e új játékot kezdeni?
 19. TestLoad(): Meghívódik-e a Load?
 20. TestLoadWrongFileFormat(): Rossz fájlformátum esetén kapunk-e exceptiont?
 21. TestBoardIndex(): Fel tudunk-e helyezni a táblára korongokat?

22. TestBoardOutOfIndexing(): Ha nagyobb számot adunk meg, mint a tábla mezőinek száma, akkor kapunk-e exceptiont?
23. TestBoardOutOfIndexing2(): Ha kisebb számot adunk meg, mint a tábla mezőinek száma, akkor kapunk-e exceptiont?