# 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, mentésére 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.

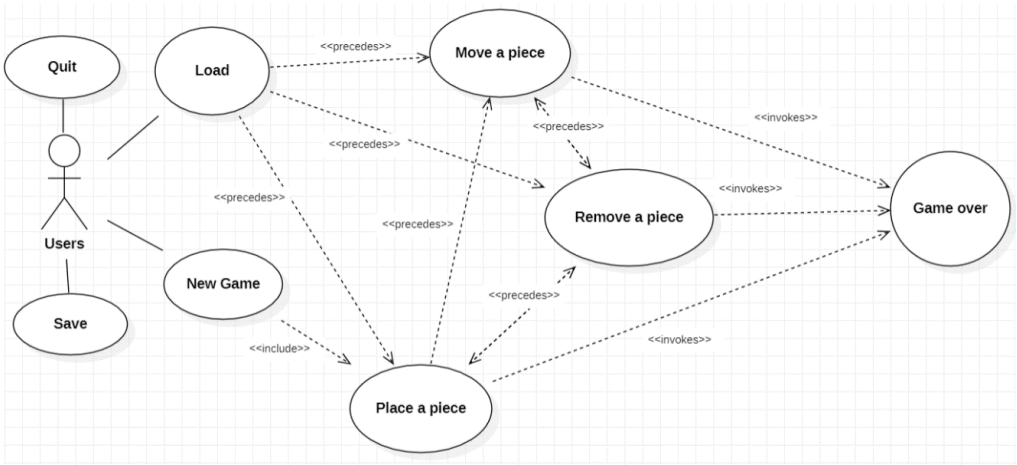
• A feladatot egyablakos asztali alkalmazásként Windows Forms grafikus felülettel valósítjuk meg.

• 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 egérkattintás 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 státuszsorba, 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:

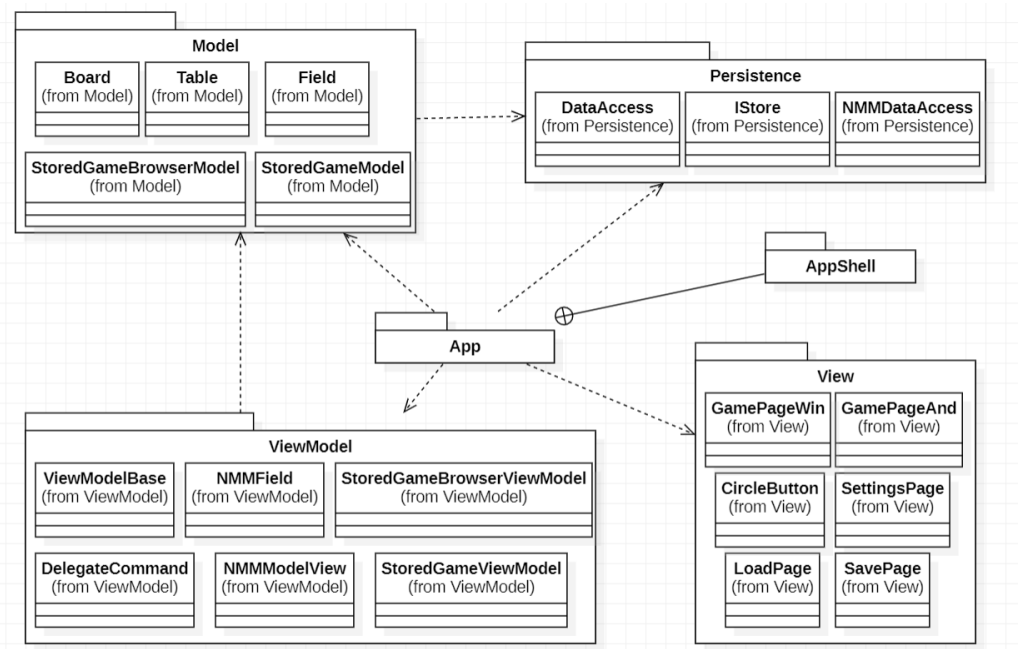
• A programot MVVM architektúrában valósíjuk meg. A megjelenítés a NMM.View, a megjelenítés modellje a NMM.ViewModel, a modell NMMModel.Model, míg a perzisztencia a NMMModel.Persistence névtérben helyezkedik el.

• 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 rendszerre is le tudunk fordítani.

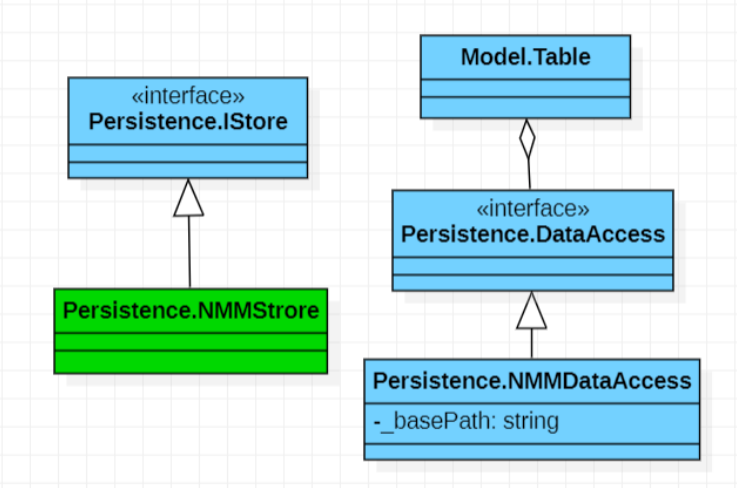
• A megvalósításból külön építjük fel a játék, illetve a betöltés és mentés funkciót, valamennyi rétegben. Utóbbi funkcionalitást újrahasznosítjuk egy korábbi projektből, így nem igényel újabb megvalósítást.

• A program vezérlését az alkalmazás osztály (App) végzi, amely példányosítja a modellt, a nézetmodell és a nézetet, biztosítja a kommunikációt, valamint felügyeli az adatkezelést.

• A program csomagdiagramja az ábrán látható:

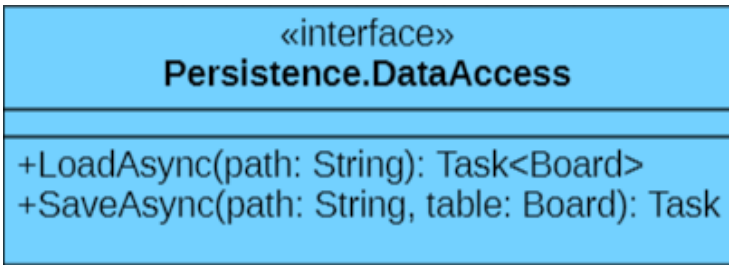


# Perzisztencia:

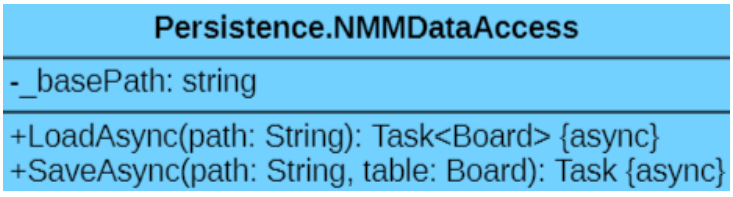



• Az adatkezelés feladata a malom táblával kapcsolatos információk tárolása, valamint a betöltés/mentés biztosítása.

• 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.



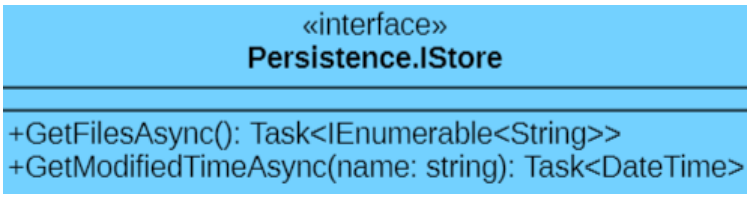

• Az interfészt szöveges fájl alapú adatkezelésre a NMMDataAccess osztály valósítja meg. A fájlkezelés során fellépő hibákat a NMMDataException kivétel jelzi. A program az adatokat szöveges fájlként tudja eltárolni, amelyeket egy megadott könyvtárban (_basePath) helyez el. Ez majd az alkalmazás platformfüggő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



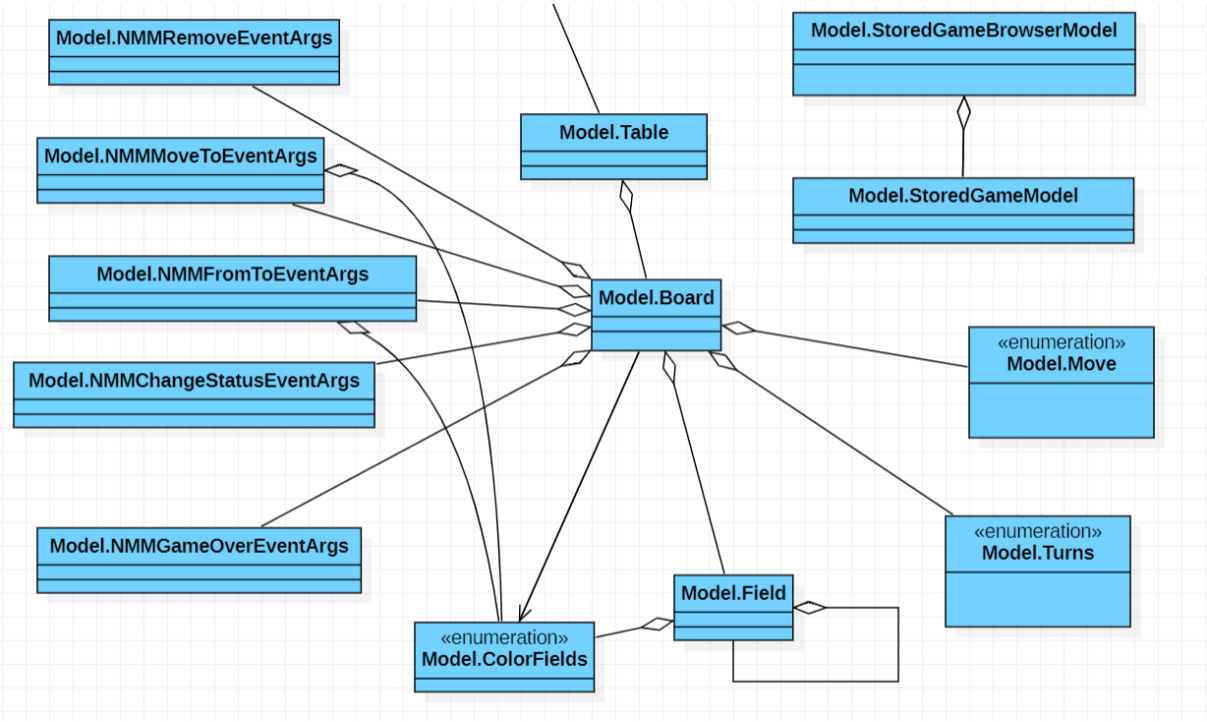

• A program az adatokat szöveges fájlként tudja eltárolni, melyek az txt kiterjesztést kapják. Ezeket az adatokat a programban bármikor be lehet tölteni, illetve ki lehet menteni az aktuális állást.

• 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 bábut, 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(int: indFrom, int: indTo, player: Turns): bool
-canMove(int ind): bool
-canPlayerMove(): bool
-removeOne(ind: int, player: Turns): bool
-threeNextToEachOther(ind: int): bool
-allNextToEachOther(player: Turns): bool
-ChangeStatus(status: string): void
-Remove(ind: int): void
-GameOver(winner: string): void
-MoveTo(color: ColorFields, ind: int): void
-FromTo(color: ColorFields, from: int, to: int): void

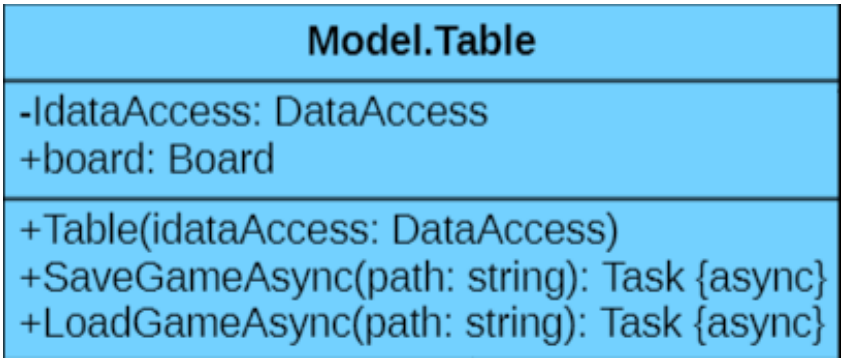
• Field osztály egy mezőt jelez, ami rámutat a szomszédjaira, és a saját színét tárolja.

**Model.Field**

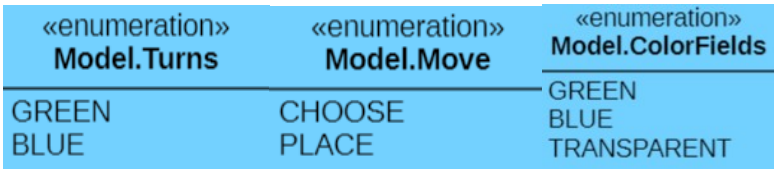
+player: Model.ColorFields
-up_: Field
-down_: Field
-right_: Field
-left_: Field
+Up: Field {property}
+Down: Field {property}
+Left: Field {property}
+Right: Field {property}

+Field()

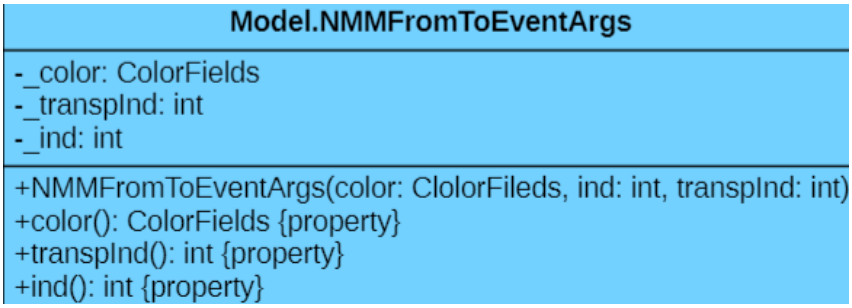
• A Table osztály tartalmaz egy Board osztályt, így betöltés (LoadGameAsync) és mentés (SaveGameAsync) esetén le tudja cserélni a malomtáblát.

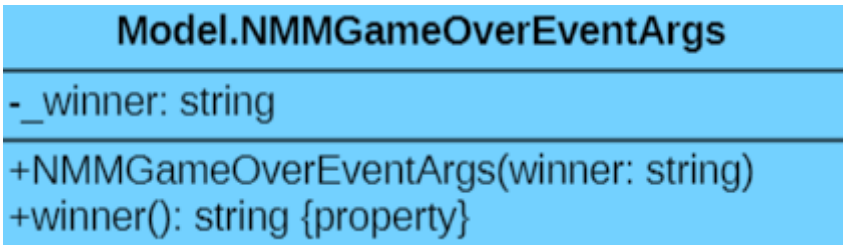


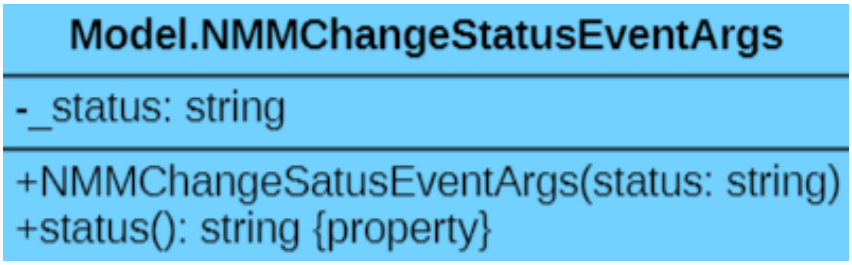

• A modellben vannak felsorolók is, a színek beállításához és a mozgatáshoz.

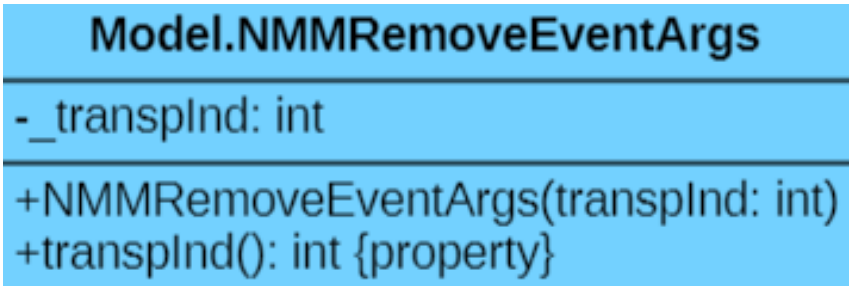


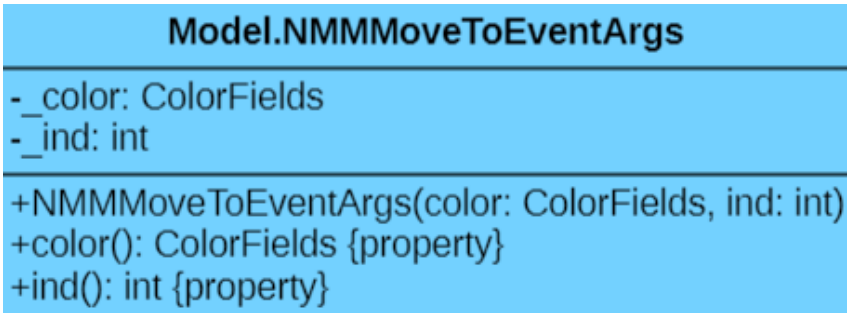

• A Board()-nak 5 eseménye van: MoveTo_, FromTo_, Remove_, ChangeStatus_ és End_. Ezekre tud a nézet feliratkozni.



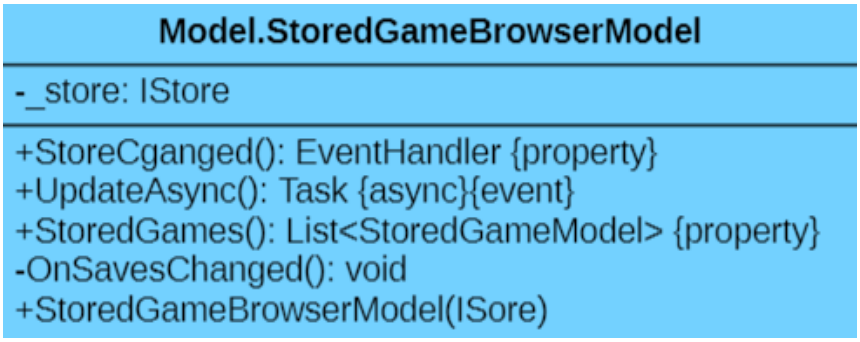



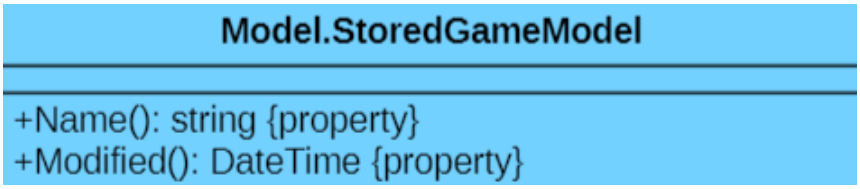





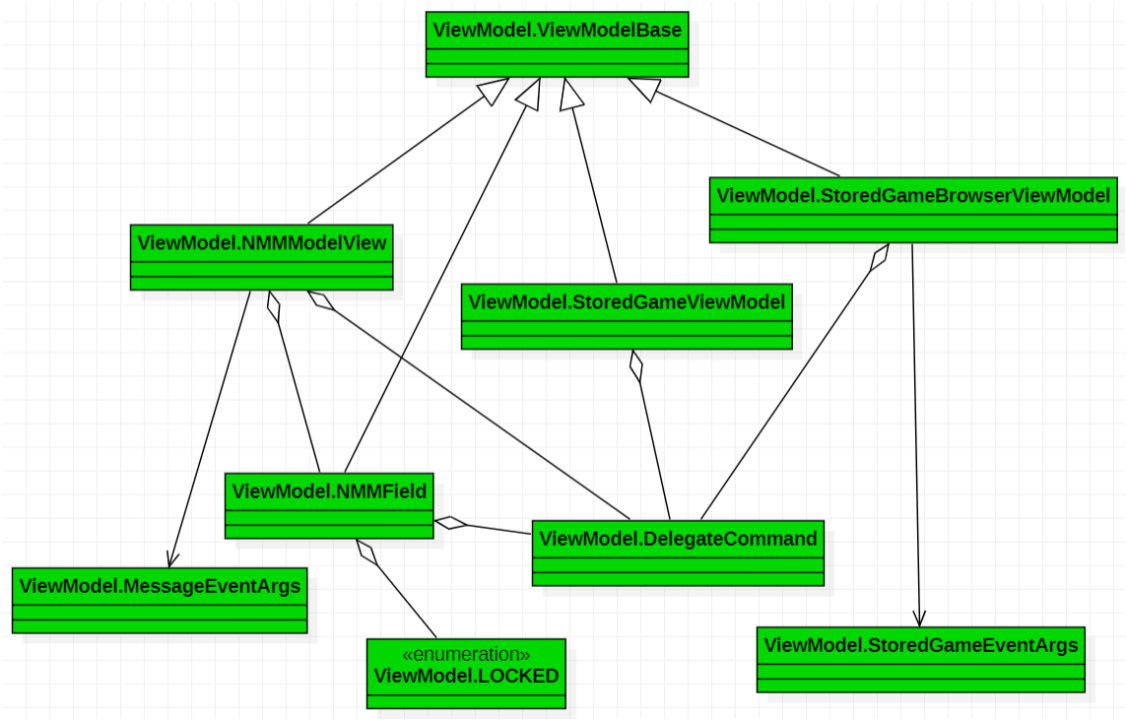




• Ide tartozik még a StroredGameBrowserModel, amiből a tárolt játékok keresőjének modellje és a StoredGameModel, ami egy tárolt játék modellje.

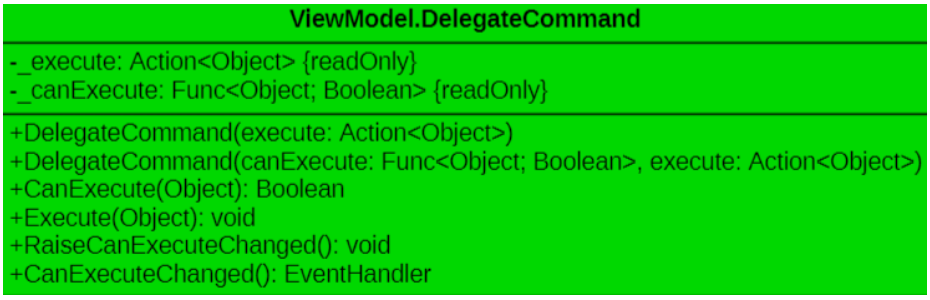


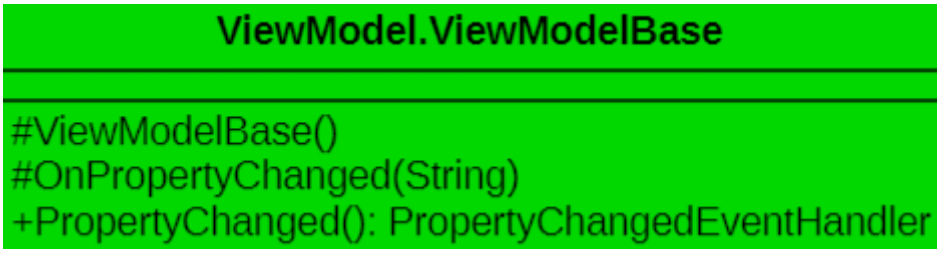



# Nézetmodell:



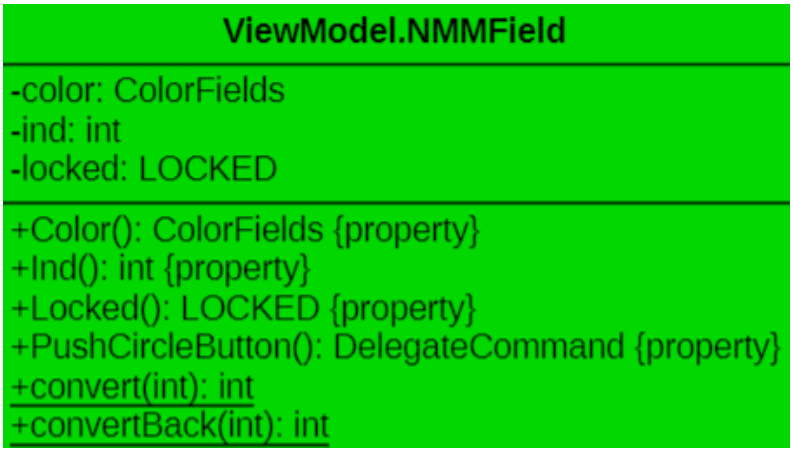

• A nézetmodell megvalósításához felhasználunk egy általános utasítás (MyCommand), valamint egy ős változásjelző (ViewModelBase) osztályt.



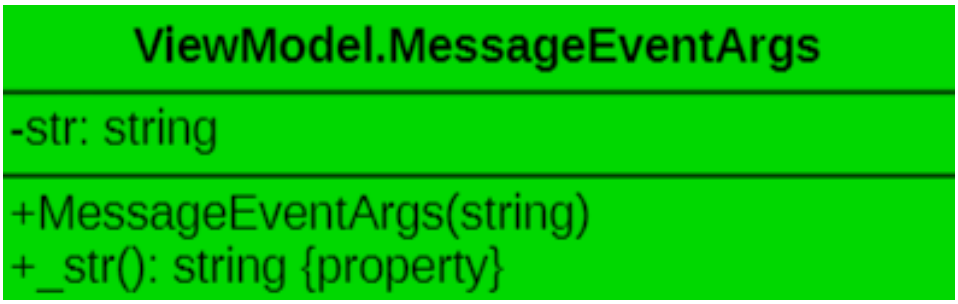




• A nézetmodell feladatait a NMMModelView osztály látja el, amely parancsokat biztosít az új játék kezdéséhez, játék betöltéséhez, mentéséhez, valamint a kilépéshez. A parancsokhoz eseményeket kötünk, amelyek a parancs lefutását jelzik a vezérlőnek. A nézetmodell tárolja a modell egy hivatkozását (table_), de csupán információkat kér le tőle. Direkt nem avatkozik a játék futtatásába.

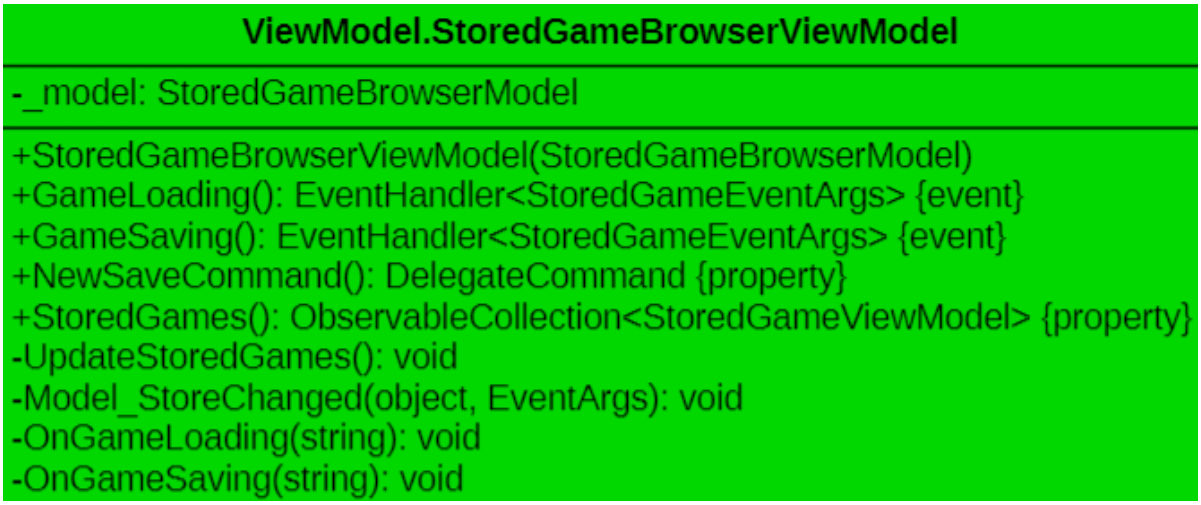

• A játékmező számára egy külön mezőt biztosítunk (NMMField), amely eltárolja a pozíciót, színét, méretét, indexét és parancsát (PushCircleButton). A mezőket egy felügyelt gyűjteménybe helyezzük a nézetmodellbe (Fields).

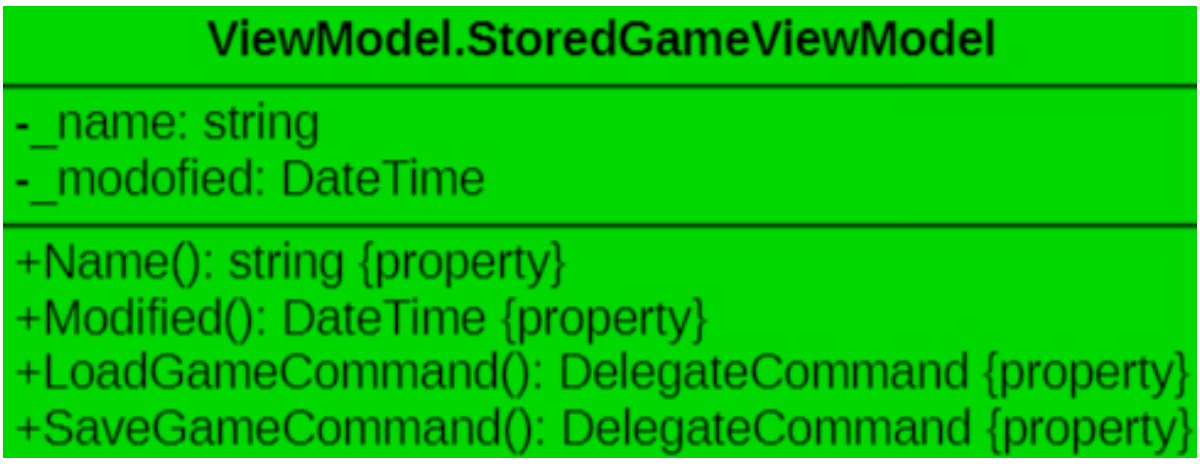


• Üzenetek küldésére használhatja a nézetmodell a MessegeEventArgs-ot.

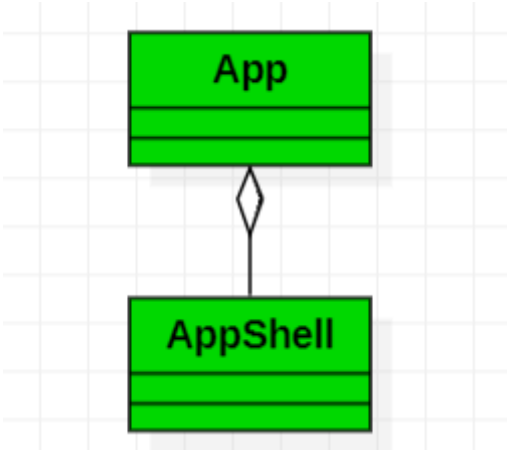



• Ide tartozik még a StroredGameBrowserViewModel, amiből a tárolt játékokat lehet kikeresni és a StoredGameViewModel, ami egy tárolt játék.




# Nézet:



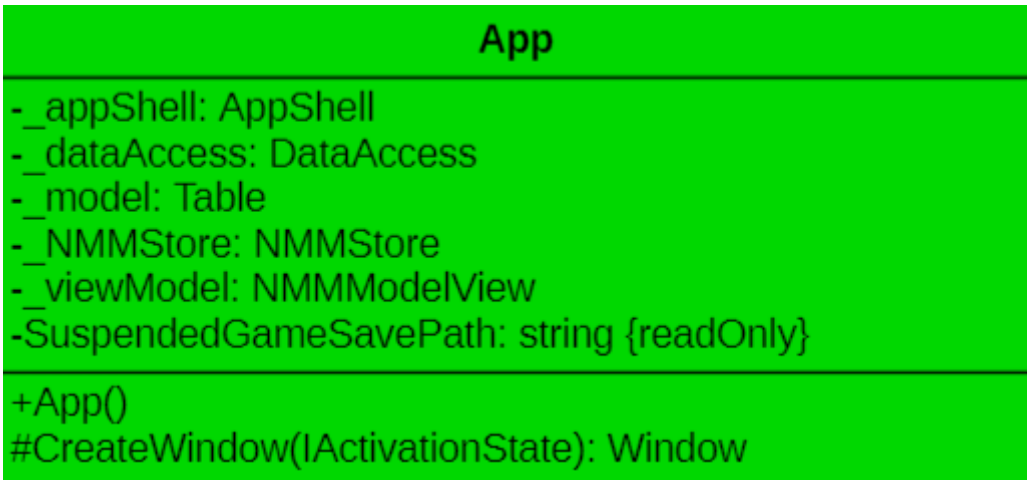

• A nézetet navigációs lapok segítségével építjük fel.

• 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.

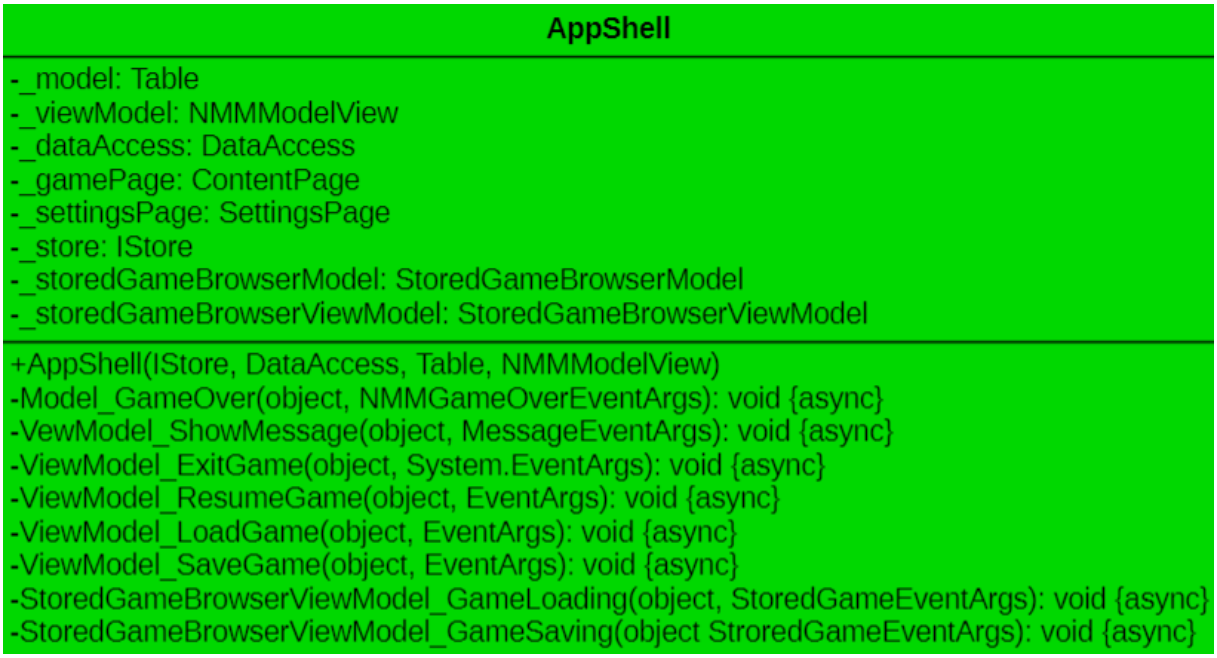
• A LoadPage és a SavePage szolgál egy létező játékállapot betöltésére, illetve egy új mentésére.

• 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.




# 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 csökkenek-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 bábuk 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?