# Első beadandó feladat Dokumentáció

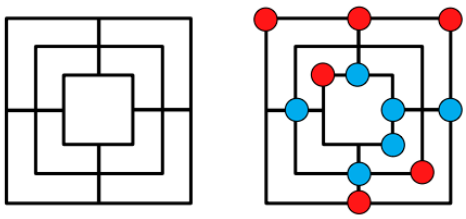
Eseményvezérelt alkalmazások

## Készítette:

Kocsis Márk (npblwo@inf.elte.hu)

## Feladat:

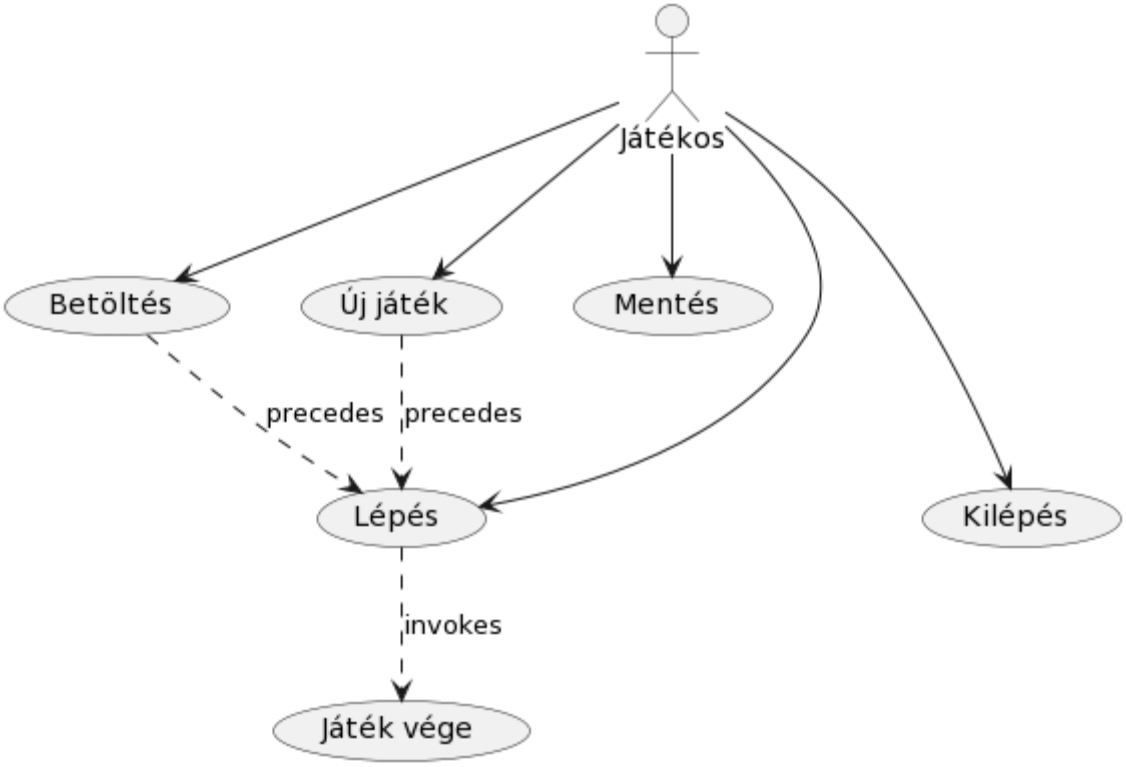
Készítsünk programot, amellyel a következő két személyes játékot játszhatjuk. A malom játékban két játékos egy 24 mezőből álló speciális játéktáblán játszhatja, ahol a mezők három egymásba helyezett négyzetben helyezkednek (mindegyikben 8, a sarkoknál és a felezőpontoknál), melyek a felezőpontok mentén 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.



*1.*

## Elemzés:

- A játékot egy 24 mezős játéktáblán tudjuk játszani, ahogy az az 1-es ábrán látható. Két játékos játssza, akik felváltva lépnek a játékszabályok szerint.
- A feladatot egyablakos asztali alkalmazásként Windows Presentation Foundation
- grafikus felülettel valósítjuk meg.
- Az ablakban elhelyezünk egy menüt a következő menüpontokkal: Fájl (Új játék, Játék betöltése, Játék mentése, Kilépés). Az ablak alján megjelenítünk két státuszsort, amely az egyes játékosok következő lépését, illetve a talonban maradt korongjait mutatja.
- A játéktábla 24 kör alakú gombból áll, és a felrajzolt vonalakból. A gombokra kattintva léphet a játékos, viszont a program csak a játék szabályainak megfelelő, érvényes lépést fogad el.
- A játéknak akkor van vége, amikor egy játékos összes korongja 3 alá csökken (táblán lévők és a talon együtt véve). A játék végét felugró ablakban jelezzük.
- Ha egy játékos a játékszabályokat betartva nem tud lépni, felugró ablakban jelezzük neki, hogy passzolnia kell.
- A játéknak két fő szakasza van, a program ezek szerint határozza meg a játékosok akcióit.
- A felhasználó esetek diagrammja a 2. képen láthatóak

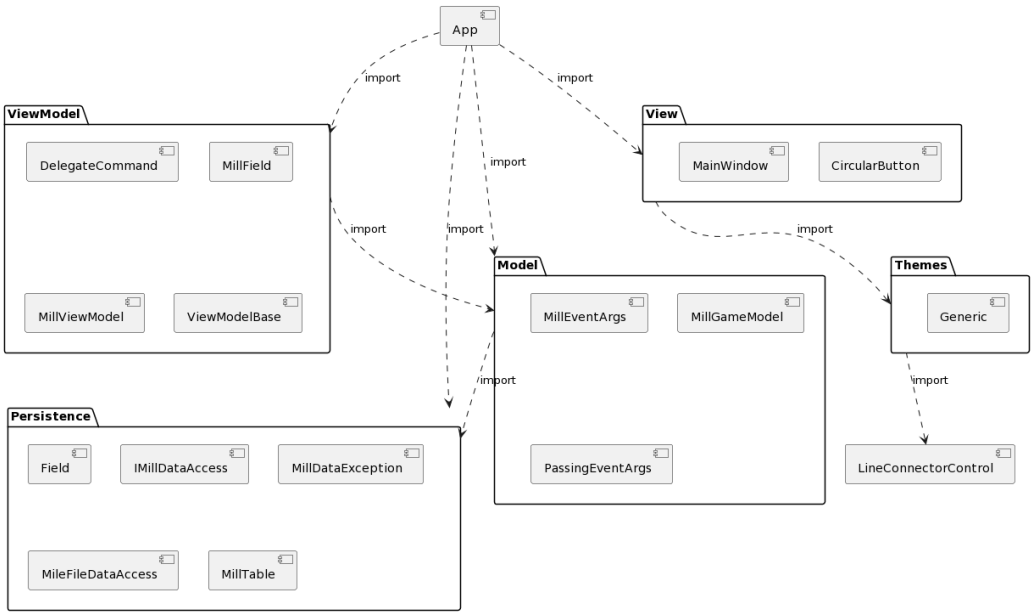



*2.*

# 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 példányosítja a modellt, a nézetmodell és a nézetet, biztosítja a kommunikációt, valamint felügyeli az adatkezelést. A program csomagszerkezete a 3. ábrán látható.




*3.*

## Perzisztencia:

- Az adatkezelés feladata a Malom táblával kapcsolatos információk tárolása, valamint a betöltés/mentés biztosítása.
- A **MillTable** osztály egy érvényes játéktáblát biztosít, amely tudja ellenőrizni, hogy új malom jön-e létre, illetve a mezők szomszédait is eltárolja. A van egy konstruktora, ami létrehoz egy 24 méretű, **Field** elemekből álló tömböt. A tábla tárolja az utolsónak lépett játékost is **_lastPlayer** adattagként a mentések miatt. Csak a mentéskor és betöltéskor használ egy **_currentAction** adattagot is a következő lépés típusát eltárolva. A tábla szintén tárolja a játékosok talonban és a pályán lévő korongjait. A táblában van egy **_mills** nevű, tömbökből álló tömb, amely a 16 különböző malmot reprezentálja. Pár fontos metódusa a **CheckFieldInMill,** ami egy mezőről ellenőrzi, hogy malomban van-e, **GetField**.
- A **Field** osztály játékmezőt reprezentál, egyik adattagja a **_player**, ami 0, 1, 2 értéket vehet fel, attól függően, hogy üres-e a mező, első játékos korongja, illetve a második játékos korongja van rajta. Másik adattagja egy egész típusú tömb, amiben az adott indexnek megfelelő mezőhöz a mező szomszédait tárolja.
- A hosszú távú adattárolás lehetőségeit az **IMillDataAccess** 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 **MillFileDataAccess** osztály valósítja meg. A fájlkezelés során fellépő hibákat a **MillDataException** kivétel jelzi.
- A program az adatokat szöveges fájlként tudja eltárolni, melyek az mtl kiterjesztést kapják. Ezeket az adatokat a programban bármikor be lehet tölteni, illetve ki lehet menteni az aktuális állást.
- A fájl első sora a speciális értékeké, a következő sörrendben: utolsó lépett játékos, első játékos talonja, második játékos talonja, első játékos korongjai a táblán, második játékos korongjai a táblán, illetve a következő lépés típusa (egészként reprezentálva). A második sorban a mezőkből álló tömb **Player** értékei vannak a megfelelő sorrendben.

## Modell

- A modell lényegi részét a **MillGameModel** osztály valósítja meg. Lehetőséget ad a lépésre a **Step** metódussal, illetve új játék kezdésére a **NewGame** metódussal. Biztosít betöltést és mentést a **Perzisztencia** rétegben lévő komponensek segítségével. Számon tartja a játék aktuális állapotát, és a játékszabályoknak megfelelő lépést engedélyez csak.
- Három eseménnyel tudja tájékoztatni a **View** réteget: **GameOver**, ami a játék végét jelöli, **GameAdvanced**, ami a játék előrehaladásakor valósul meg, illetve a **PlayerHasToPass**, amivel jelezzük, ha egy játékosnak passzolnia kell.
- A modell példányosításkor megkapja az adatkezelés felületét, amelynek segítségével lehetőséget ad betöltésre (**LoadGameAsync**) és mentésre (**SaveGameAsync**).

## Nézetmodell

- A nézetmodell megvalósításához felhasználunk egy általános utasítás (**DelegateCommand**), valamint egy ős változásjelző (**ViewModelBase**) osztályt.
- A nézetmodell feladatait a **MillViewModel** 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 **(_model**), de csupán információkat kér le tőle. Direkt nem avatkozik a játék futtatásába.

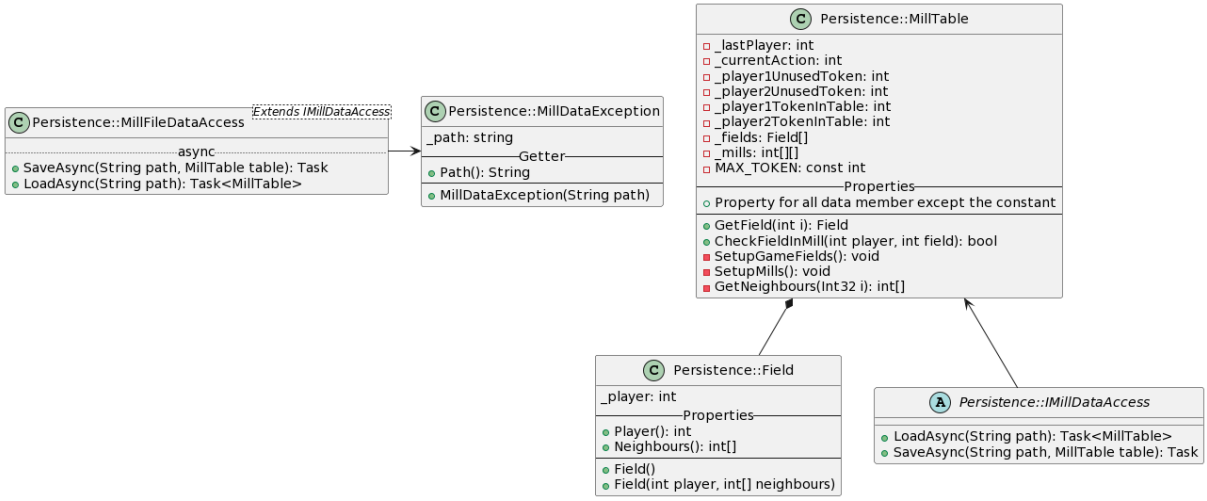
- ékmező számára egy külön mezőt biztosítunk (**MillField**), amely eltárolja a pozíciót, illetve a mező értékét, ami ha nincs rajta korong 0, ha van rajta kék, akkor 1, ha piros, akkor 2-t vesz fel. A lépés parancsát (**StepCommand**) is eltároljuk. A mezőket egy felügyelt gyűjteménybe helyezzük a nézetmodellbe (**Fields**).

## Nézet

- A nézet csak egy képernyőt tartalmaz, a **MainWindow** osztályt. A nézet egy rácsban tárolja a játékmezőt, a menüt és a státuszsort. A játékmező két **Grid** vezérlő alkotja, ahol az első Grid a külső csomagoló egység, a belső **Grid** definiál egy 7x7-es rácsot, amelyben felvesszük a kör alakú gombokat a megfelelő helyekre. A csomagoló **Gridben** elhelyezzük a mezőket összekötő vonalakat, amik egy saját magunk írt WPF vezérlőből állnak (**LineConnectorControl**). Minden adatot adatkötéssel kapcsolunk a felülethez, továbbá azon keresztül szabályozzuk a gombok színét is. A csomagdiagrammon látható egység, a **Themes** nem alkot önálló névteret, a speciális vezérlőnket definiálja a **MainWindow** számára.
- A fájlnév bekérését betöltéskor és mentéskor, valamint a figyelmeztető üzenetek megjelenését beépített dialógusablakok segítségével végezzük.

## Környezet:

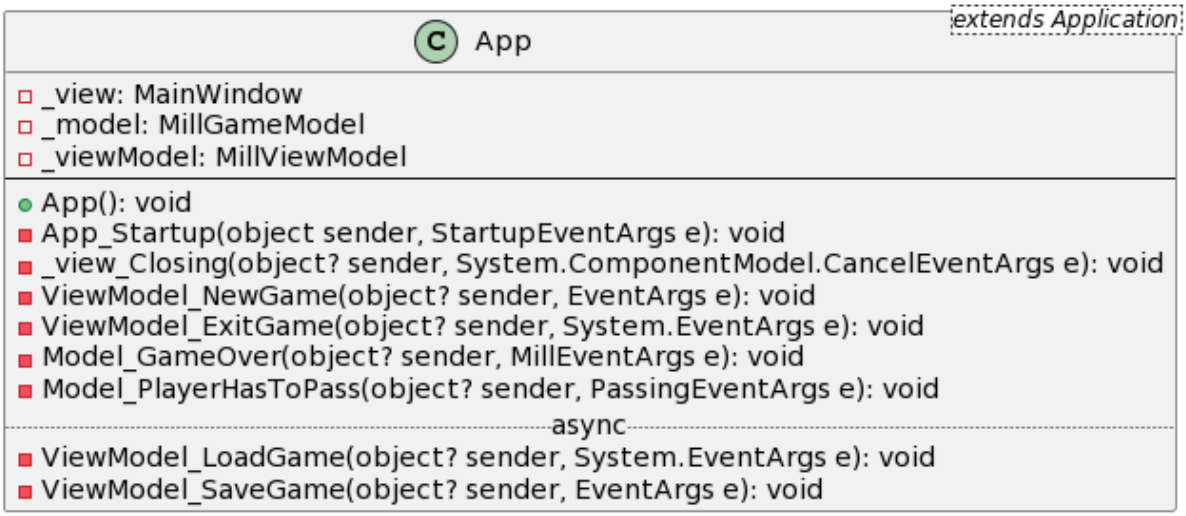
- Az **App** osztály feladata az egyes rétegek példányosítása (**App_Startup**), összekötése, a nézetmodell, valamint a modell eseményeinek lekezelése, és ezáltal a játék, az adatkezelés, valamint a nézetek szabályozása.




*Perzisztencia*


*Model*


*ViewModel*




*App*

## Tesztelés:

- A modell funkcionalitása egységtesztek segítségével lett ellenőrizve a **MillGameModelTest** osztályban.
- A következő testesetek futottak le: **MillGameModelLoadTest** (betöltés), **MillGameModelNewGameTest** (új játék), **MillGameModelStepTest** (lépés), **MillGameModelMillTest** (malom keletkezik), **MillGameModelWin** (valaki nyer).