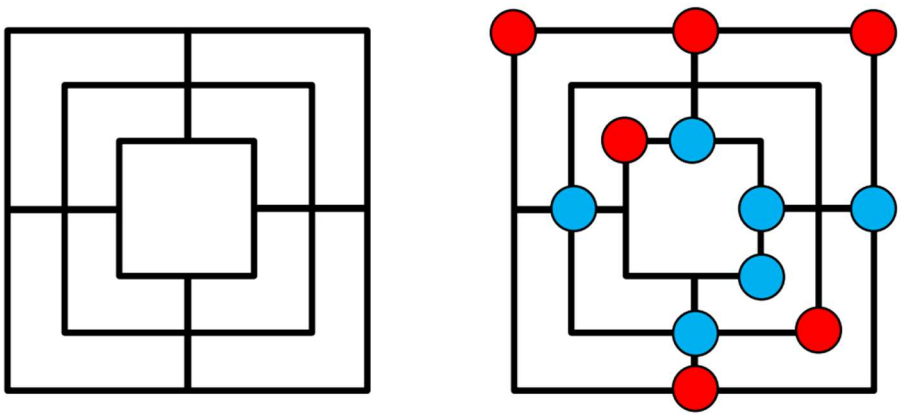
2. Beadandó feladat dokumentáció

Készítette:
Biborka Ágnes
E-mail: h4jd6b@inf.elte.hu

Feladat:

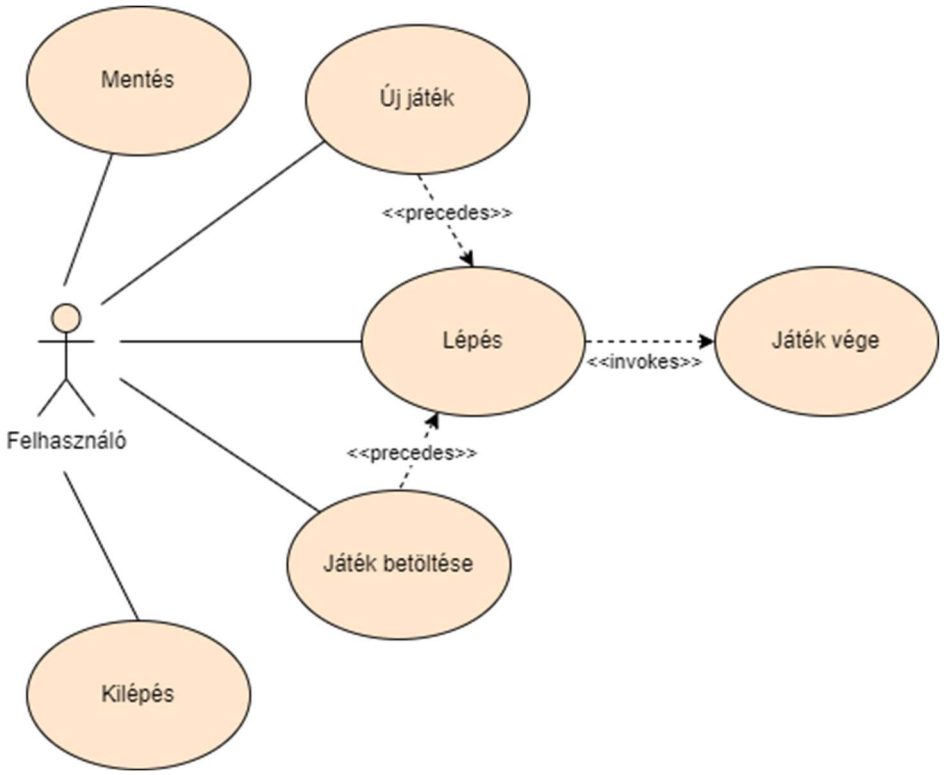
Készítsünk programot, amellyel a következő kétszemélyes játékot játszhatjuk. A malomjátékot két játékos egy 24 mezőből álló speciális játéktáblán játszhatja, ahol a mezők három egymásba helyezett négyzetben helyezkednek (mindegyikben 8, a sarkoknál és a felezőpontoknál), melyek a felezőpontok mentén 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 feladatot egyablakos asztali alkalmazásként Windows Forms grafikus felülettel valósítjuk meg.
- A program indításkor automatikusan új játékot indít.
- Az ablakban elhelyezünk egy menüt a következő menüpontokkal: Menü (Új játék, Játék betöltése, Játék mentése, Kilépés).
- A játéktáblát három darab négyzet reprezentálja, a négyzetek sarkaiban és oldalfelező pontjaiban egy-egy nyomógomb található, összesen 24 darab.

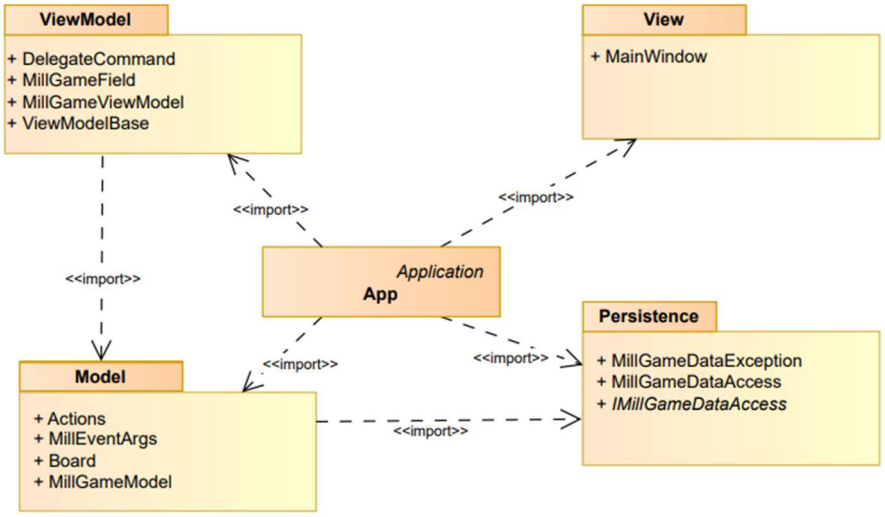
- Amennyiben a játkosok még nem helyezték el összes bábujukat a mezőn, a soron következő játékos egy üres mezőre kattintva *elhelyezheti ott a bábuját*. (Rakodás fázis.)
- Amennyiben már a játékosok az összes bábujukat a táblára helyezték és még mindkét játékosnak több, mint 2 bábuja van a táblán, a soron következő játékos először a mozgatni kívánt bábura, majd egy szomszédos üres nyomógombra kattintva *mozgathatja a bábuját*. (Mozgatás fázis.)
- Amennyiben egy játékos malmot alakít ki, akár a rakodás, akár a mozgatás fázis alatt, az *ellenfél egy bábujára kattintva leveheti azt* a pályáról (kivéve, ha az a bábu egy malom része).
- A játék automatikusan feldob egy dialógusablakot, amikor vége a játéknak (az egyik játékos bábuinak a száma 3 alá csökken).
- Szintén dialógusablakokkal végezzük el a mentést, illetve betöltést, a fájlneveket a felhasználó adja meg.
- A felhasználói esetek az 1. ábrán láthatóak.



1. ábra: Felhasználói esetek diagramja

Tervezés:

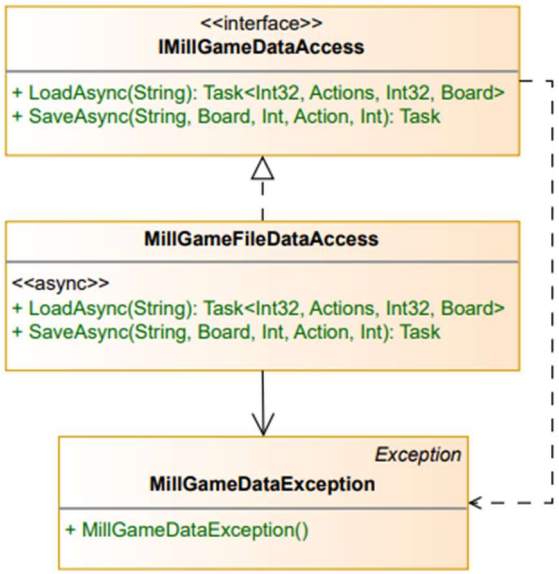
- Programszerkezet:
  - A programot *MVVM architektúrában* valósítjuk meg, ennek megfelelően **`View`**, **`Model`**, **`ViewModel`** és **`Persistence`** névtereket valósítunk meg az alkalmazáson belül. A program környezetét az alkalmazás osztály (App) végzi, amely példányosítja a modellt, a nézetmodell és a nézetet, biztosítja a kommunikációt, valamint felügyeli az adatkezelést. A program csomagszerkezete a 2. ábrán látható.
  - A program szerkezetét két projektre osztjuk implementációs megfontolásból: a **`Persistence`** és **`Model`** csomagok a program felületfüggetlen projektjében, míg a **`ViewModel`** és **`View`** csomagok a WPF függő projektjében kapnak helyet.



2. ábra: Az alkalmazás csomagdiagramja

- Perzisztencia:
  - A réteg felel egy malomjátékkal kapcsolatos információk hosszútávú tárolásáért, valamint a betöltéséért és mentéséért.
  - A *hosszú távú adattárolás* lehetőségeit az **`IMillGameDataAccess`** interfész biztosítja, 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 **`MillGameFileDataAccess`** osztály valósítja meg. A fájlkezelés során fellépő *hibákat* a **`MillGameDataException`** kivétel jelzi.

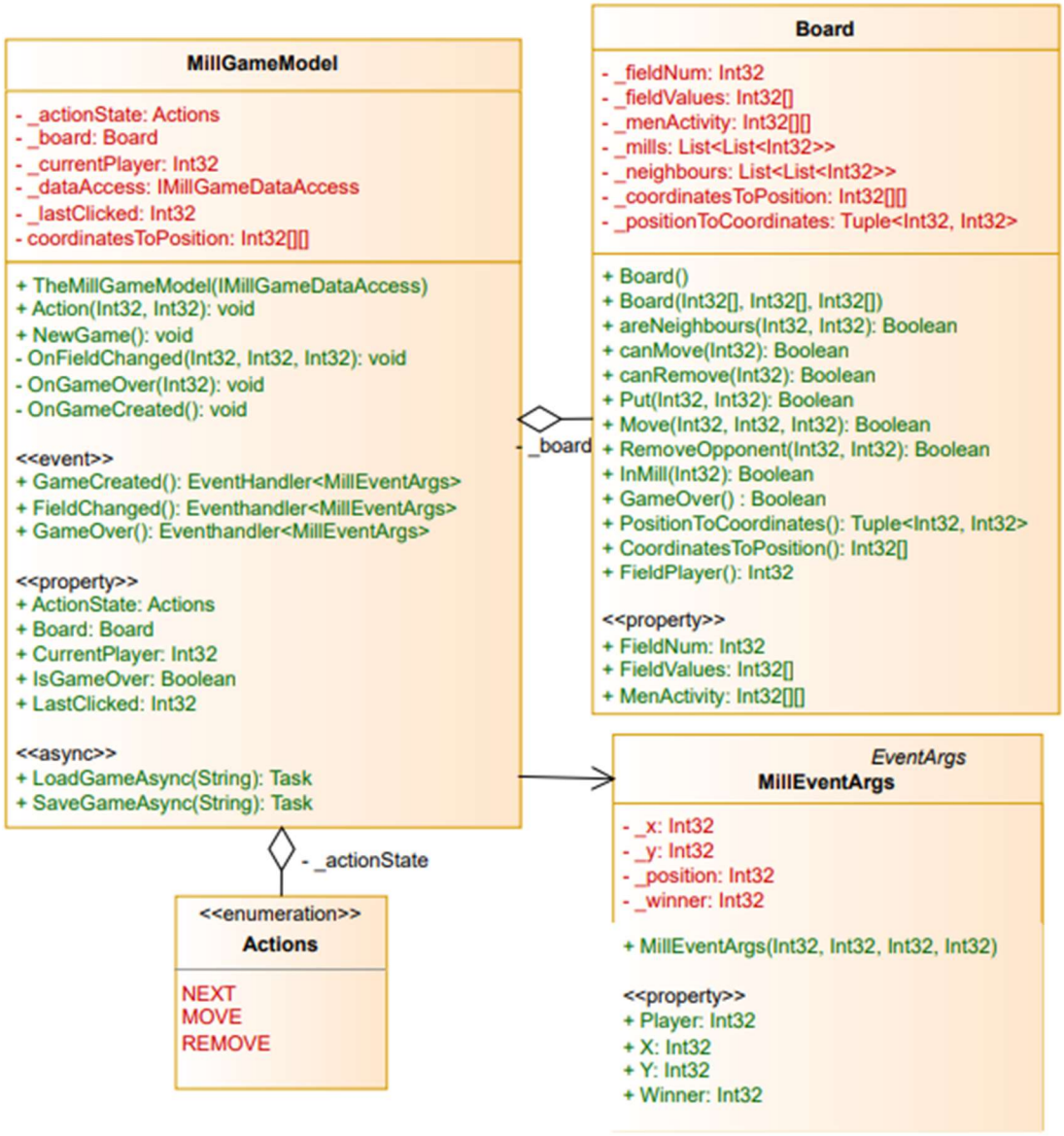
- A program az adatokat szöveges fájlként tudja eltárolni, melyek az **`mtl`** kiterjesztést kapják. Ezeket az adatokat a programban bármikor be lehet tölteni, illetve ki lehet menteni az aktuális állást.
- A fájl első sora szóközökkel elválasztva rendre megadja az aktuális játékost (0, ha piros, 1, ha kék következik), a következő lépés fajtát (következő lépés, mozgatás, bábu levétel), és az utoljára kijelölt mezőt azonosító pozíciót. A fájl második sora tárolja a piros játékos bábuinak státuszát, rendre az inaktív, az aktív és a már kiesett játékosok számát, szóközzel elválasztva. A harmadik sorban a kék játékos bábuinak státusza következik hasonlóan. Az utolsó sor a tábla mezőinek értéke, pozíció szerint növekvő sorrendben, szóközzel elválasztva. Egy mező értéke a rajta álló játékos száma (0, ha piros, 1, ha kék játékos foglalja el), vagy -1, ha üres.



3. ábra: A Persistence csomag osztálydiagramja

- Modell:
  - A *modell lényegi részét* a **`MillGameModel`** osztály valósítja meg, amely szabályozza a tábla tevékenységeit, valamint a fájl menü műveleteit. A típus lehetőséget ad új játék kezdésére (**`NewGame`**), valamint lépésre (**`Action`**).
  - A mezők állapotváltozásáról a **`FieldChanged`** esemény, míg a játék végéről a **`GameOver`** esemény tájékoztat. Az események argumentuma (**`MillEventArgs`**) tárolja a győztes számát (0, ha a piros játékos győzött, 1, ha a kék, -1, ha még egyik sem), a megváltoztatandó mező pozícióját és hogy melyik játékos áll a megváltozott mezőn.

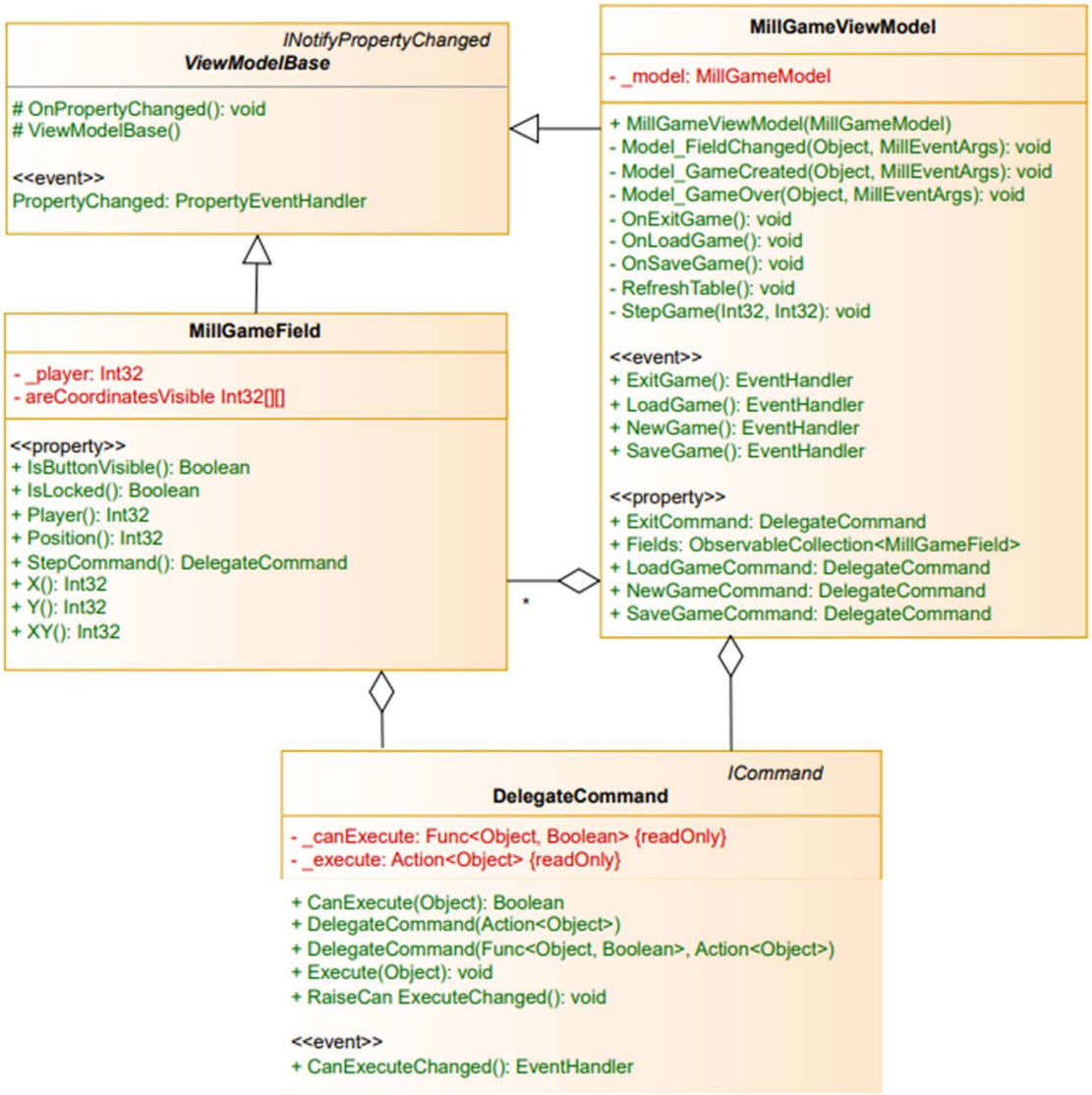
- A modell példányosításkor megkapja az adatkezelés felületét, amelynek segítségével lehetőséget ad betöltésre (**`LoadGameAsync`**) és mentésre (**`SaveGameAsync`**).
- A **`Board`** osztály egy érvényes malom táblát biztosít és lehetőséget a **`MillgameModel Action`** metódusa által meghívott műveletek (**`Put`**, **`Move`**, **`RemoveOpponent`**) megvalósítására az egyes mezők (**`_fieldValues`** tömb) és a játékosok bábuinak aktivitás (**`_menActivity`** tömb) megváltoztatásával.
- A **`Put`** művelet egy üres helyre helyez bábut, a **`Move`** művelet egy magadott helyről megadott helyre mozgat egy bábut ellenőrizve, hogy a megadott mezők szomszédosok-e és a cél mező üres-e. A **`RemoveOpponent`** művelet egy megadott bábut távolít el, amennyiben az nem a soron következő játékoshoz tartozik és nem áll malomban.



4. ábra: A Model csomag osztálydiagramja

- Nézetmodell:
  - A nézetmodell feladatait a **`MillGameViewModel`** 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.
  - A nézetmodell megvalósításához felhasználunk egy általános utasítás osztályt (**`DelegateCommand`**), valamint egy változásjelző ősosztályt (**`ViewModelBase`**).

- A játékmezők számára egy külön osztályt biztosítunk (**`MillGameField`**), ami a **`ViewModelBase`** változásjelző osztály leszármazottja, és eltárolja a mező láthatóságát, a mezőn tartózkodó játékost (0, ha piros, 1, ha kék játékos áll a mezőn), a mező X, Y koordinátáit, valamint a lépés parancsát (**`StepCommand`**, ami a **`DelegateCommand`** osztály egy példánya). A mezőket egy felügyelt gyűjteménybe (**`ObservableCollection`**) helyezzük a nézetmodellbe (**`Fields`**).

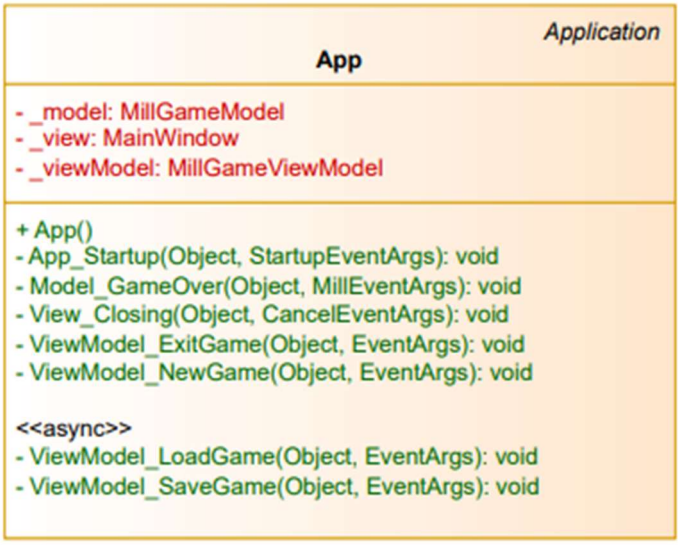


5. ábra: A ViewModel csomag osztálydiagramja

- Nézet:
  - A nézet *egy ablakot tartalmaz*, a **`MainWindow`** osztályt. A nézet egy rácsban tárolja a menüsort és a játékmezőt. A játékmező egy **`ItemsControl`** vezérlő, ahol dinamikusan felépítünk egy rácsot

(**`UniformGrid`**), amely gombokból áll. Minden adatot adatkötéssel kapcsolunk a felülethez, továbbá azon keresztül szabályozzuk a gombok színét is.
    - 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.



6. ábra: A vezérlés osztálydiagramja

Tesztelés:

- A modell funkcionalitása *egységtesztek* segítségével lett ellenőrizve a **`MillGameModelTest`** és a **`MillGameBoardTest`** osztályban.
- Az alábbi tesztesetek kerültek megvalósításra:
    - **`MillGameModelLoadTest`**: A játék modell betöltésének tesztelése mockolt perzisztencia réteggel. Ugyanebben a teszt metódusban kerül tesztelésre a **`_model.GameOver`** event is. A mockolt modellben a kék játékos egy lépésre áll a győzelemtől, a következő játék lépés szimulálásával kiváltódik a játék vége esemény.
    - **`BoardPutTest`**: Előre megadott játékállás folytatásaként annak tesztelése, hogy egy mezőre le tudunk-e helyezni bábut, vagy sem. Ide tartoznak a következő esetek:

- A kijelölt mezőn az ellenfél bábuja áll.
- A kijelölt mezőn saját bábu áll.
- A kijelölt mező még üres.

- **`CanMoveTest`**: Előre megadott játékállás folytatásaként annak tesztelése, hogy egy játékos tudja-e mozgatni valamelyik bábuját, vagy sem. Ide tartoznak a következő esetek:
  - A játékosnak minden bábuja be van kerítve, nem tud mozogni.
  - A játékosnak van olyan bábuja, aminek van üres szomszédja, tud mozogni.
- **`BoardMoveTest`**: Előre megadott játékállás folytatásaként annak tesztelése, hogy egy megadott játékos tud-e egy kijelölt mezőre lépni. Ide tartoznak a következő esetek:
  - A kijelölt mező szomszédos a megadott játékoséval, de a mezőn egy saját bábu áll.
  - A kijelölt mező szomszédos a megadott játékoséval, de a mezőn az ellenfél bábuja áll.
  - A kijelölt mező nem szomszédos a megadott játékoséval és foglalt.
  - A kijelölt mező üres, de nem szomszédos a megadott játékoséval.
  - A kijelölt mező szomszédos a megadott játékoséval és üres.
- **`BoardInMillTest`**: Előre megadott játékállás folytatásaként annak tesztelése, hogy egy megadott játékos épp malomban áll-e vagy sem. Ide tartoznak a következő esetek:
  - A kijelölt mező szomszédos a megadott játékoséval, de a mezőn egy saját bábu áll.
  - A kijelölt mező szomszédos a megadott játékoséval, de a mezőn az ellenfél bábuja áll.
  - A kijelölt mező nem szomszédos a megadott játékoséval és foglalt.
  - A kijelölt mező üres, de nem szomszédos a megadott játékoséval.
  - A kijelölt mező szomszédos a megadott játékoséval és üres.
- **`CanRemoveTest`**: Előre megadott játékállások folytatásaként annak tesztelése, hogy az ellenfélnek van-e olyan bábuja, amit az aktuális játékos el tud távolítani. Ide tartoznak a következő esetek:
  - Az ellenfél összes bábuja malomban áll, az aktuális játékos egyet sem tud levenni.
  - Az ellenfélnek van olyan bábuja, ami nem áll malomban, az aktuális játékos tud bábut levenni az ellenfél bábujai közül.
- **`BoardRemoveOpponentTest`**: Előre megadott játék állás folytatásaként annak tesztelése, hogy egy kijelölt mező bábuja eltávolítható-e a pályáról.

        - A kijelölt bábu a játékos saját bábuja.
        - A kijelölt bábu az ellenfél bábuja, de malomban áll.
        - A kijelölt mező üres, nem is áll rajta bábu.
        - A kijelölt mező bábuja az ellenfél játékosa és nem áll malomban.
    - **`BoardGameOverTest`**: Előre megadott játék állásban annak ellenőrzése, hogy a **`GameOver`** metódus helyesen jelzi-e a játék végét.