3. Beadandó feladat dokumentáció

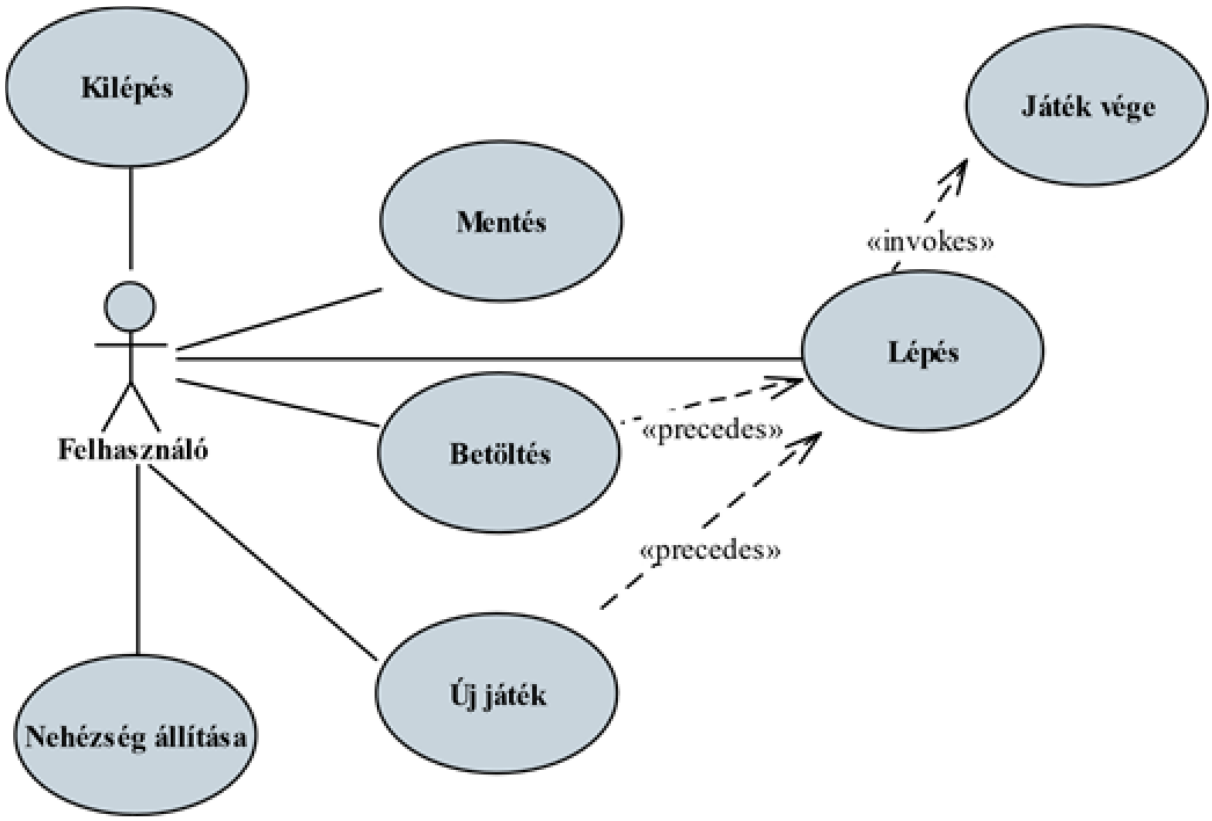
Készítette:
Máté Zsolt Sándor
E-mail: s0y2on@inf.elte.hu

Feladat:

Készítsünk programot, amellyel a következő két személyes játékot játszhatjuk. Adott egy $n \times n$ mezőből álló tábla, amelyre a játékosok 2 × 1-es méretű téglalapokat helyezhetnek el (vízszintesen, vagy függőlegesen). A játékosok felváltva léphetnek. A játék célja, hogy a téglalapokkal elhatároljuk a tábla egy részét (teljesen körbevéve téglalapokkal), amelyben így minden mező a játékosé lesz (beleértve az ellenfél által korábban elfoglalt mezőket is). A program külön jelölje meg a lehelyezett téglalapokat, illetve az elfoglalt területeket, és játék közben folyamatosan jelenítse meg az elfoglalt terület méretét játékosonként. Eseményvezérelt alkalmazások 2020/2021 őszi félév 8 A program biztosítson lehetőséget új játék kezdésére a pályaméret megadásával (6 × 6, 8 × 8, 10 × 10), valamint játék mentésére és betöltésére. Ismerje fel, ha vége a játéknak, és jelenítse meg, melyik játékos győzött.

Elemzés:

- A játékot három különböző méretű pályán átszhatjuk: 6 x 6, 8 x 8 és 10x10. A program indításkor a legkisebb pályamérete állít be, és automatikusan új játékot indít.
- A feladatot egyablakos asztali alkalmazásként Windows Forms grafikus felülettel valósítjuk meg.
- Az ablakban elhelyezünk egy menüt a következő menüpontokkal: Új játék (6x6, 8x8, 10x10), Méntés, Betöltés. Az ablak alján megjelenítünk egy státuszsort, amely a pontok számát és aktuális játékost jelzni.
- A játéktáblát egy 6x6(8x8,10x10) nyomógombokból álló rács reprezentálja. A nyomógomb egérkattintás hatására megváltoztatja zölddel jelzett gombokat. A már beszinezett, vagy szabálytalan elhelyezést nem engedjük.
- A játék automatikusan feldob egy dialógusablakot, amikor vége a játéknak (jelzi hogy ki nyert és hány ponttal). 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 háromrétegű architektúrában valósíjuk meg. A megjelenítés a **`View`**, a modell a **`Model`**, míg a perzisztencia a **`Persistence`** névtérben helyezkedik el. A program csomagszerkezete a 2. ábrán látható.

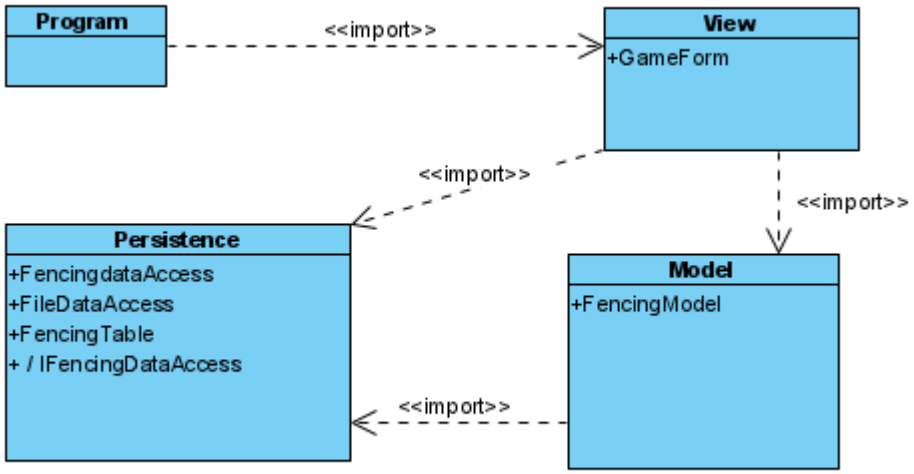
Perzisztencia:

- Az adatkezelés feladata a Fencing táblával kapcsolatos információk tárolása, valamint a betöltés/mentés biztosítása.
- A **`FencingTable`** osztály egy érvényes Fencing táblát biztosít (azaz mindig ellenőrzi a beállított értékek), ahol minden mezőre ismert az értéke (**`_field`**) és az aktuális játékost.. A tábla alapértelmezés szerint *6*×*6*-os, de ez a konstruktorban paraméterezhető. A tábla lehetőséget az állapotok lekérdezésére.

  (**`IsFilled`**, **`GameSize`**, , **`GetFiledType`**,), valamint szabályos léptetések ellenőrzésére és végrehajtására (**`CanPlaceBlock`**, **`PlaceBlock`**) , illetve direkt beállítás (**`SetFieldType`**) elvégzésére.

- A hosszú távú adattárolás lehetőségeit az **`ISudokuDataAccess`** 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 **`SudokuFileDataAccess`** osztály valósítja meg.
- A program az adatokat szöveges fájlként tudja eltárolni, melyek az **`sav`** kiterjesztést kapják. Ezeket az adatokat a programban bármikor be lehet tölteni, illetve ki lehet menteni az aktuális állást.
- A fájl egy FencingTable class ír le json állományként.

2. ábra: Az alkalmazás csomagdiagramja



Modell:

- A modell lényegi részét a **`FencingModel`** osztály valósítja meg, amely szabályozza a tábla tevékenységeit, valamint a játék egyéb paramétereit, úgymint a lépések (**`_BluePoints,RedPoints`**). A típus lehetőséget ad új játék kezdésére (NewGame), valamint lépésre (**`TryStepGame`**). Új játéknál megadható a kiinduló játéktábla is, különben automatikusan generálódnak kezdő mezők.
- A játékállapot változásáról a **`FieldChanged`** esemény, míg a játék végéről a **`GameOver`** esemény tájékoztat. Az események argumentuma objectumokat adunk át.
- A modell példányosításkor megkapja az adatkezelés felületét, amelynek segítségével lehetőséget ad betöltésre (**`LoadGameAsync`**) és mentésre

(**`SaveGameAsync`**)

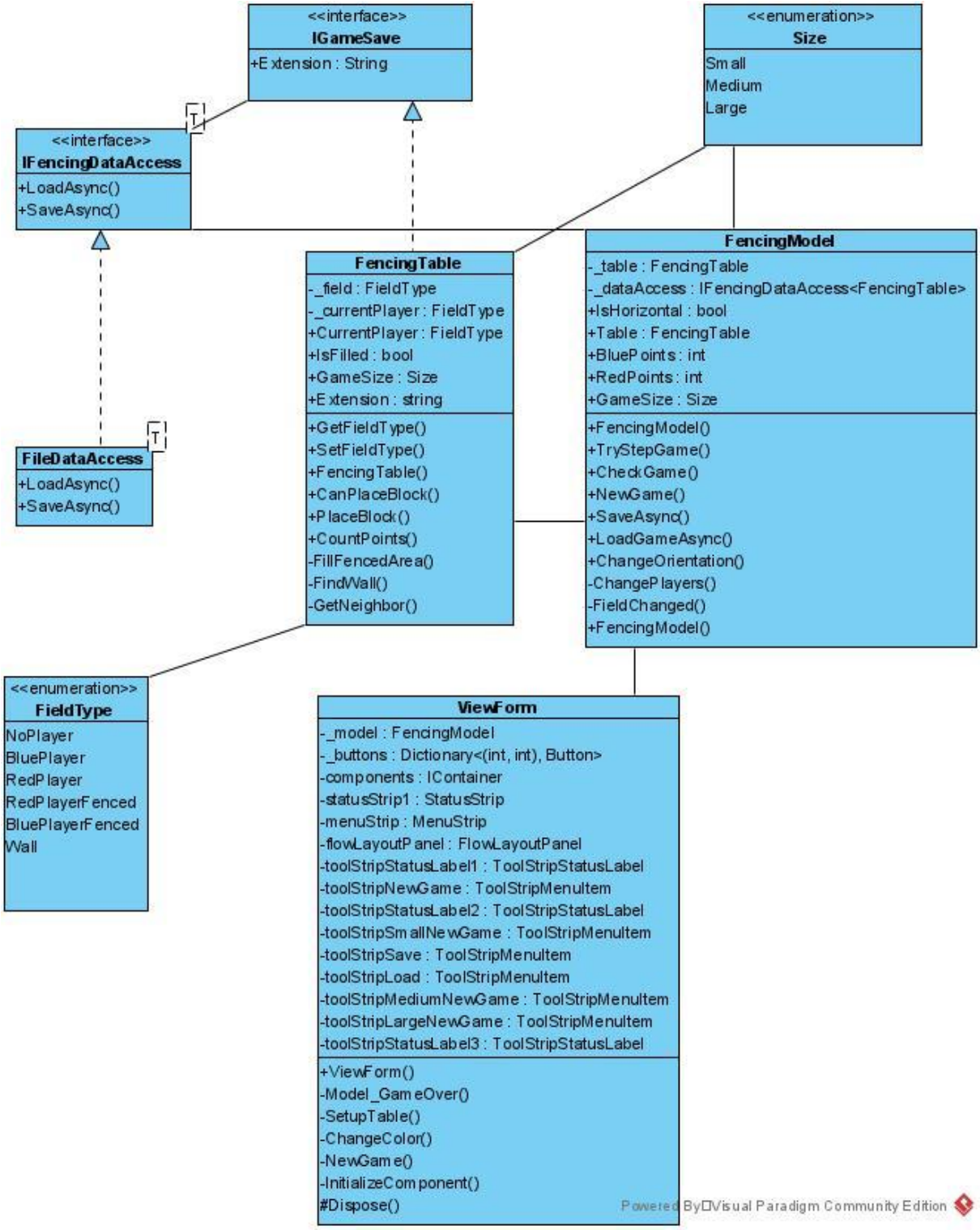
- A játék mértetét a **`Size`** felsorolási típuson át kezeljük, és külön osztályban konstansok segítségével tároljuk az egyes méretek paramétereit.
- Hasonlóan FieldType a pályan elhezhető blokkot típusait tároljuk külön osztáltban.

Nézet:

- A nézetet a **`ViewForm`** osztály biztosítja, amely tárolja a modell egy példányát (**`_model`**).

- A játéktáblát egy dinamikusan létrehozott gombmező (**`_buttonGrid`**) reprezentálja. A felületen létrehozzuk a megfelelő menüpontokat, illetve státuszsort, valamint dialógusablakokat, és a hozzájuk tartozó eseménykezelőket. A játéktábla generálását és az értékek beállítását (**`SetupTable`**) metódusvégzi.

- A program teljes statikus szerkezete a 3. ábrán látható.



3. ábra: Az alkalmazás osztálydiagramja

Tesztelés:

- A modell funkcionalitása egységtesztek segítségével lett ellenőrizve a **`FencingModelTest`** osztályban.
- Az alábbi tesztesetek kerültek megvalósításra:
- **`SudokuGameModelNewGame`**: Új játék indítása, a mezők kitöltése
- **`FencingPlacingCanPlaceBlock`**: Játékbeli lépés ellenőrzése, játék. Több lépés végrehajtása azonos játékmezőn.
- **`FencingChangePlayers`**: A játékosok váltakozásának ellenőrzése.
- **`FencingAreaFenced`**: Annak ellenőrzése hogy az elkerített részek megfelelően átszineződnek.
- **`FencingLoadTest`**: Betöltés megfelelő működése, mezők ellenőrzés mérettel és aktuális játékossal.
- **`FencingGameOver`**: Játék vége eseménynek ellenőrzése.