1. Beadandó feladat dokumentáció

Feladat:

Készítsünk programot, amellyel az alábbi két személyes játékot játszhatjuk.

Adott egy $n \times n$ pontból álló játéktábla, amelyen a játékosok két szomszédos pont között vonalakat húzhatnak (vízszintesen, vagy függőlegesen). A játék célja, hogy a játékosok a húzogatással négyzetet tudjanak rajzolni (azaz ők húzzák be a negyedik vonalat, független attól, hogy az eddigieket melyikük húzta). Ilyen módon egyszerre akár két négyzet is elkészülhet. A játék addig tart, amíg lehet húzni vonalat a táblán.

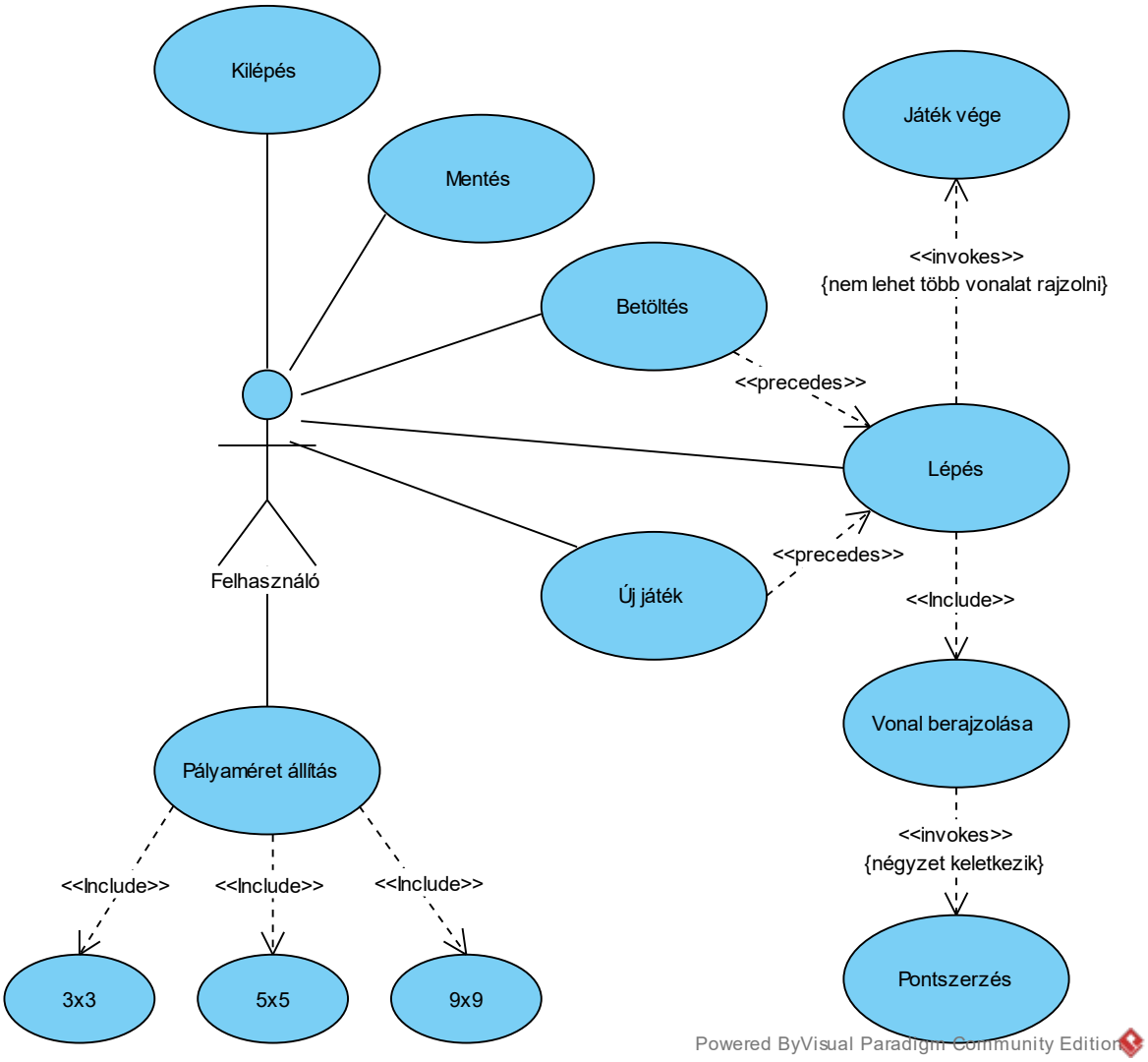
A játékosok felváltva húzhatnak egy-egy vonalat, de ha egy játékos berajzolt egy négyzetet, akkor ismét ő következik.

A program biztosítson lehetőséget új játék kezdésére a pályaméret megadásával (3×3, 5×5, 9×9), játék mentésére és betöltésére. Ismerje fel, ha vége a játéknak, és jelenítse meg, melyik játékos győzött (ha nem döntetlen). Játék közben a vonalakat, illetve a négyzeteket színezze a játékos színére.

Elemzés:

- A játékot három pályamérettel játszhatjuk: 3x3-mas, 5x5-ös, 9x9-es. A program indításkor 5x5-ös pályamérettel töltődik be.
- A feladatok Xamarin Forms alkalmazásként, Android platformon valósítjuk meg, amely két lapból fog állni. Az alkalmazás portré tájolást támogat.
- A játék négy képernyőn fog megjelenni:
    - Az első képernyő (Játék) tartalmazza a játéktáblát, a játék állását (pontszámok, aktuális játékos) a lap alján, az új játék, valamint a beállítások gombjait a lap tetején.
    - A második képernyőn van lehetőség betöltésre, illetve mentésre, valamint a pályaméret állítására (három rádiógombbal).
    - A további két képernyő a betöltésnél, illetve mentésnél megjelenő lista, ahol a játékok elnevezése mellett a mentés dátuma is látható. Mentés esetén ezen felül lehetőség van új név megadására is.
- A játéktáblát egy $n \times n$ db kör alakú gombból álló rács reprezentálja. A felület segíti a felhasználót, hogy az aktuális állapotban mely pontból (amelyiknek van olyan szomszédja, amely között még nincs vonal), s mely pontba húzható vonal (szomszédos és még nincs másik vonal). Egy vonal behúzásakor az aktuális játékos színére színeződik az, amennyiben keletkezik négyzet az szintén.

- A játék automatikusan feldob egy felugró ablakot, amikor vége a játéknak (nem lehet több vonalat behúzni).
- A felhasználói esetek az 1. ábrán láthatók.

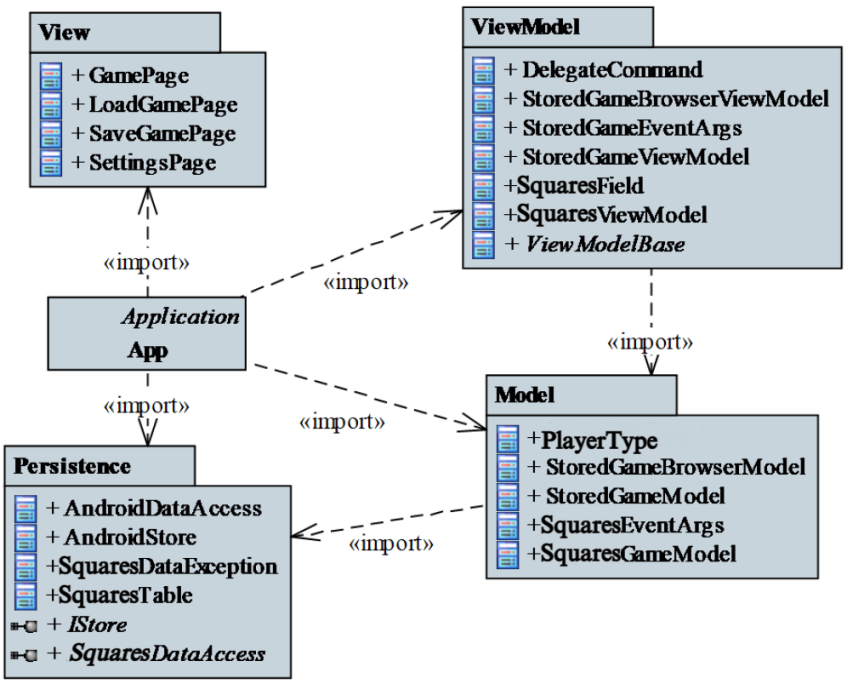


1. ábra: Felhasználói esetek diagramja

Tervezés:

❖ Programszerkezet

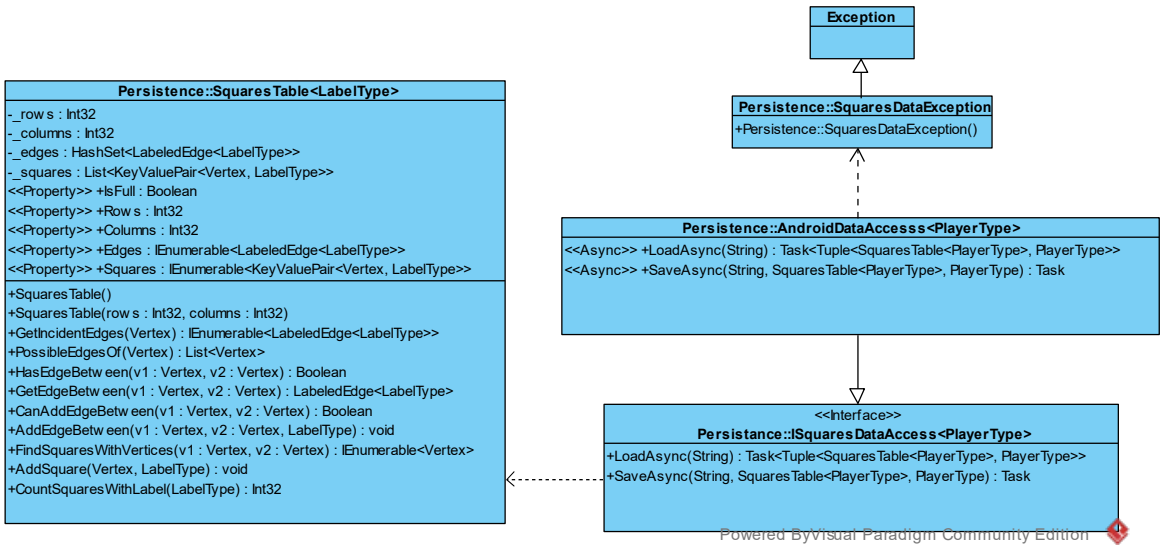
- A szoftvert két projektből építjük fel, a Xamarin Forms megvalósítást tartalmazó osztálykönyvtárból (.NET Standard Class Library), valamint az Android platform projektből. Utóbbi csupán a perzisztencia Android specifikus megvalósítását tartalmazza, minden további programegységet az osztálykönyvtárban helyezünk el.
- 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.



2. ábra: Az alkalmazás csomagdiagramja

❖ **Perzisztencia *(3. ábra)***

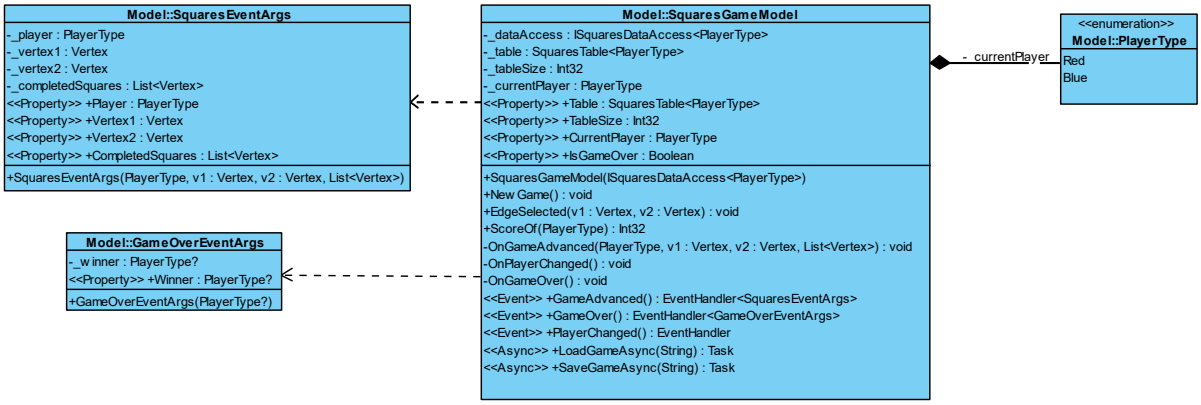
- A perzisztencia feladata a játéktáblával (gráfszerű reprezentáció) kapcsolatos információk tárolása, valamint a betöltés/mentés biztosítása.
- A **`SquaresTable`** osztály egy érvényes játéktáblát biztosít (ellenőrzi a tárolt értékeket, kapott paramétereket). Tárolja a már berajzolt vonalakat (**`_edges`**), négyzeteket (**`_squares`**), és az aktuális játékost (**`_currentPlayer`**). A tábla alapértelmezés szerint 5×5-ös, de ez a konstruktorban paraméterezhető. Lehetőség van egy pontra illeszkedő vonalak lekérdezésére (**`GetIncidentEdges`**); egy pontból kiinduló lehetséges vonalak lekérdezésére (**`PossibleEdgesOf`**); két pont közötti vonal lekérdezésére (**`GetEdgeBetween`**); adott vonalat (két pont által meghatározott) tartalmazó négyzet(ek) lekérdezésére (**`FindSquaresWithVertices`**).
- A pontok (**`Vertex`**) és a vonalak (**`LabeledEdge`**) egy **`Utilities`** nevű osztályban vannak megvalósítva. A vonalak címkézve vannak (**`LabelType`** típusparaméterezhető címkével), és a tartalmazott pontok sorrendje nem számít egyenlőségvizsgálatkor, azaz irányítatlan.
- A tábla lehetőséget az állapotok lekérdezésére (**`IsFull`**, **`CurrentPlayer`**, **`Edges`**, **`Squares`**).
- A hosszú távú adattárolás lehetőségeit az **`ISquaresDataAccess`** 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 Android adatkezelésre a **`AndroidDataAccess`** osztály valósítja meg.
- A program az adatokat szöveges fájlként tudja eltárolni, melyek a felhasználó személyes könyvtárában helyezünk el.
- A fájl első sora megadja a tábla méretét szóközzel elválasztva. A második sora megadja az aktuális játékost. A harmadik a címkézett éleket (**`címke:x1,y1-x2,y2;`** formában). A nagyedik a címkézett négyzeteket (**`címke:x,y;`** formában, ahol x és y a bal felső sarok koordinátái).



*3. ábra: A **`Persistence`** csomag osztálydiagramja*

❖ Modell *(4. ábra)*

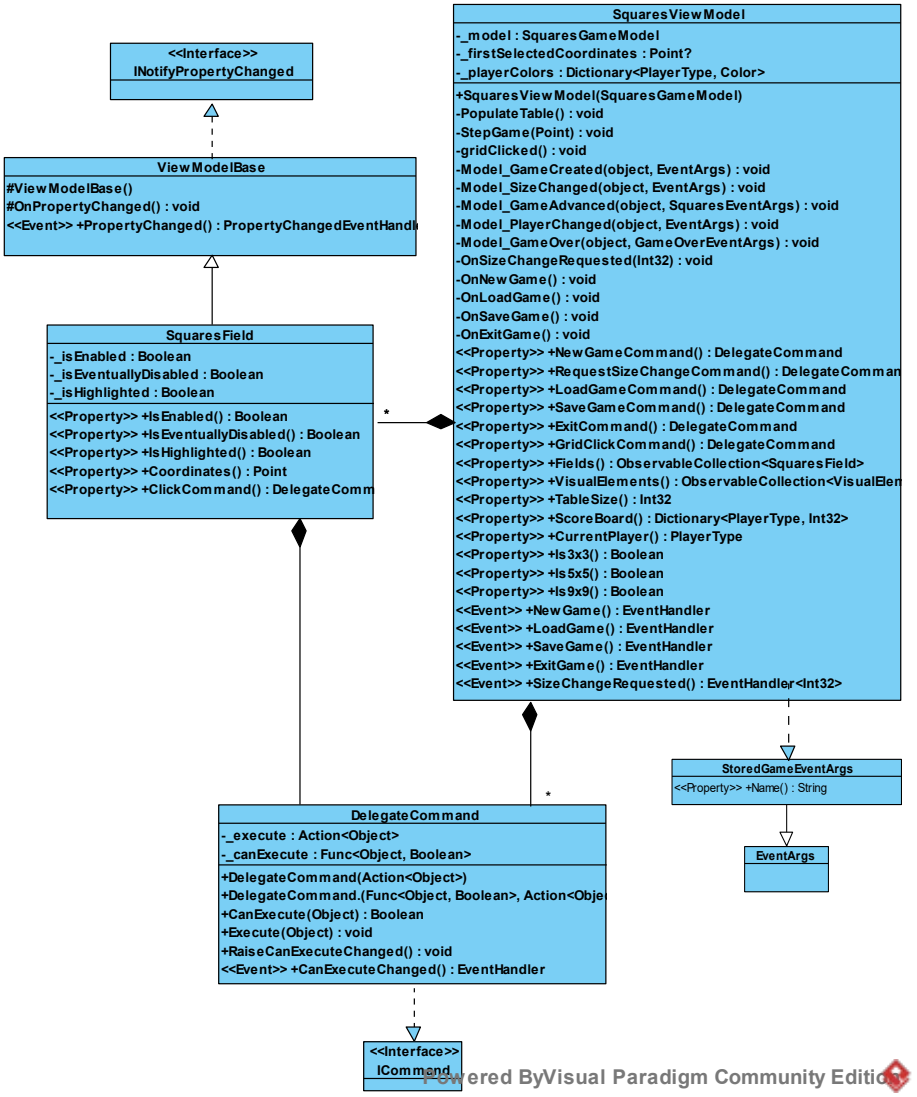
- A modell lényegi részét a **`SquaresGameModel`** osztály valósítja meg, amely szabályozza a tábla tevékenységeit. A típus lehetőséget ad új játék kezdésére (**`NewGame`**), lépésre (**`EdgeSelected`**), valamint egy játékos pontszámának lekérdezésére (**`ScoreOf`**).
- A játékállapot változásáról a **`GameAdvanced`** esemény, a játék végéről a **`GameOver`** esemény, míg a játékos változásáról a **`PlayerChanged`** esemény tájékoztat. A **`GameAdvanced`** esemény argumentuma (**`SquaresEventArgs`**) tárolja az újonnan behúzott vonal végpontjait, valamint a vonal behúzásával keletkező négyzet(ek)et. A **`GameOver`** esemény argumentuma (**`GameOverEventArgs`**) tárolja a nyertest (ha van egyértelmű).
- A modell példányosításkor megkapja az adatkezelés felületét, amelynek segítségével lehetőséget ad betöltésre (**`LoadGameAsync`**) és mentésre (**`SaveGameAsync`**).
- A játékosok típusát a **`PlayerType`** felsorolási típus reprezentálja. A modell fel van készítve ennek bővítésére.



*4. ábra: A **`Model`** csomag osztálydiagramja*

❖ Nézetmodell *(5. ábra)*

- A nézetmodell megvalósításához felhasználunk egy általános utasítás (**`DelegateCommand`**), valamint egy ős változásjelző (**`ViewModelBase`**) osztályt.
- A nézetmodell feladatait a **`SquaresViewModel`** 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, illetve a pálya méretet szabályozza (vezérlőn keresztüli megerősítéssel). Direkt nem avatkozik a játék futtatásába, a lépés végrehajtásán felül.
- A játékmező számára egy külön mezőt biztosítunk (SquaresField), amely eltárolja a pozíciót, engedélyezettséget, kiemeltséget, valamint a kattintás parancsát (**`ClickCommand`**). A mezőket egy felügyelt gyűjteménybe helyezzük a nézetmodellbe (**`Fields`**).
- Ugyancsak felügyelt gyűjteménybe helyezzük a táblán található, kirajzolandó (vonal és négyzet) grafikus elemeket (**`VisualElements`**).



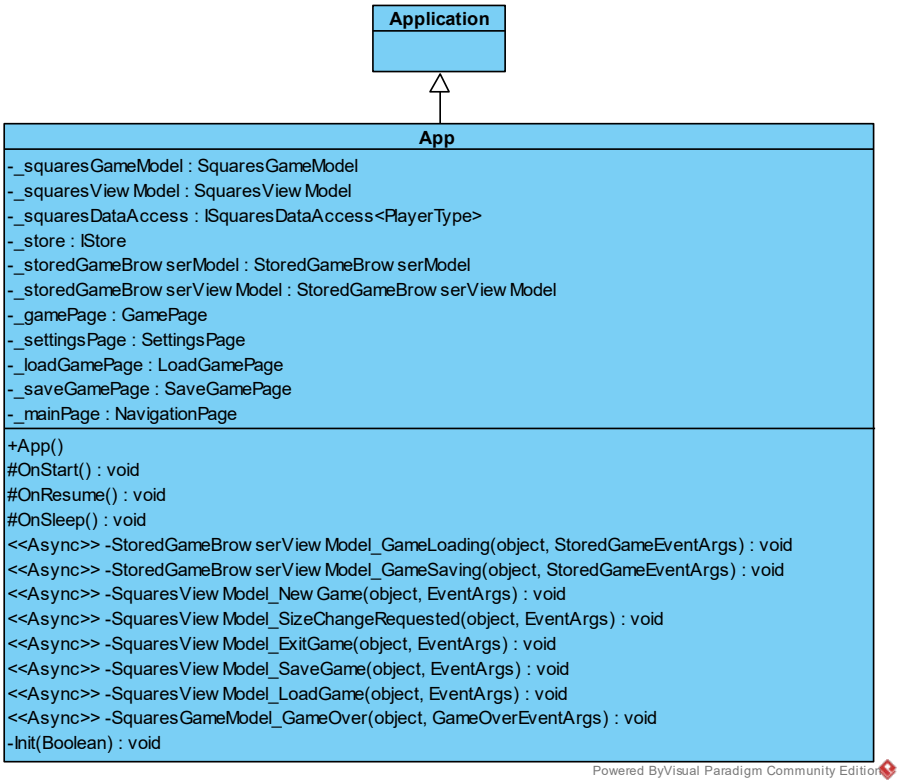
*5. ábra: A **`ViewModel`** osztálydiagramja*

❖ Nézet

- A nézetet navigációs lapok segítségével építjük fel.
- A **`GamePage`** osztály tartalmazza a játéktáblát, amelyet egy **`CustomGrid`** segítségével valósítunk meg (ez egy külső komponens), amelyben Button elemeket helyezünk el.
- A **`SettingsPage`** osztály tartalmazza a betöltés, mentés gombjait, illetve **`RadioButton`** példányokat a nehézség állítására.

❖ Környezet *(6. ábra)*

- Az **`App`** osztály feladata az alkalmazás vezérlése, a rétegek példányosítása és az események feldolgozása.
- Kezeljük az alkalmazás életciklust, így felfüggesztéskor (**`OnSleep`**) elmentjük az aktuális játékállást (**`SuspendedGame`**), folytatáskor (**`OnResume`**) és újraindításkor (**`OnStart`**) pedig folytatjuk, amennyiben történt mentés.



6. ábra: A vezérlés osztálydiagramja

Tesztelés:

- A modell funkcionalitása egységtesztek segítségével lett ellenőrizve a **`SquaresTest`** osztályban.
- Az alábbi tesztesetek kerültek megvalósításra:
  - **`SquaresGameModelNewGameTest:`** új játék indításának tesztelése. Megfelelő a tábla mérete, behúzott vonalak, rajzolt négyzetek száma.
  - **`SquaresGameModelEdgeSelectedTest:`** él kiválasztásának tesztelése, hogy bekerül-e a táblába.
  - **`SquaresGameModelEdgeSelectedNewSquareTest:`** új négyzet keletkezésének tesztelése, megfelelőek a keletkező koordináták és megfelelő játékoshoz rendelődik-e. Eseménykezelő (**`GameAdvanced`**) tesztelése.
  - **`SquaresGameModelPlayerSteppingTest:`** következő játékosra lépés tesztelése és négyzet rajzolása esetén ugyanaz a játékos következik-e.
  - **`SquaresGameModelScoreOfTest:`** négyzet rajzolása esetén csak a megfelelő játékos kap pontot.
  - **`SudokuGameModelLoadTest:`** a játék modell betöltésének tesztelése mockolt perzisztencia réteggel.