

# 1. beadandó feladat dokumentáció

## Készítette:

Gasparin Zsombor

[rvdsn3@inf.elte.hu](mailto:rvdsn3@inf.elte.hu)

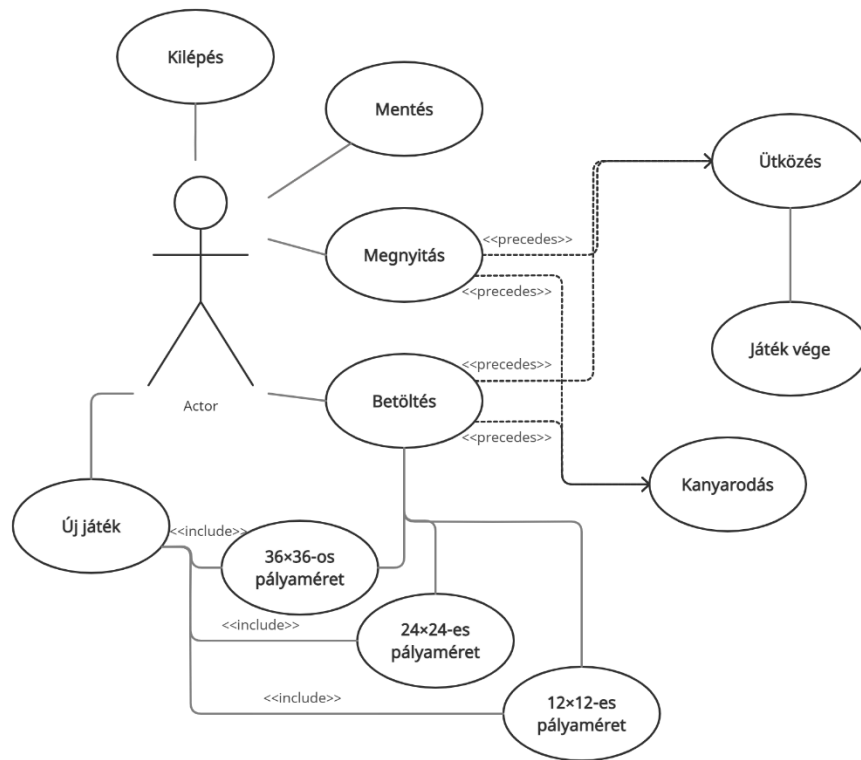
## Feladat

Készítsük programot, amellyel a Tronból ismert fénymotor párbajt játszhatjuk. Adott egy  $n \times n$  elemből álló játékpálya. A két játékos a bal, illetve jobb oldal közepén indul egy-egy fénymotorral, amely egyenesen halad (rögzített időközönként) a legutoljára beállított irányba (függőlegesen, vagy vízszintesen). A motorokkal lehetőség van balra, illetve jobbra fordulni. A fénymotor mozgás közben fénycsíkot húz, ami a játék végéig ott marad. Az a játékos veszít, aki előbb nekiütközik a másik játékos motorjának, bármelyikük fénycsíkjának vagy a pálya szélének.

A program biztosítson lehetőséget új játék kezdésére a pályaméret megadásával ( $12 \times 12$ ,  $24 \times 24$ ,  $36 \times 36$ ), valamint játék szüneteltetésére (ekkor nem telik az idő, és nem mozognak a motorok). Ismerje fel, ha vége a játéknak, és jelenítse meg, melyik játékos győzött. Ezen felül szüneteltetés alatt legyen lehetőség a játék elmentésére, valamint betöltésére.

## Elemzés

- A feladatot Windows Forms grafikus felülettel valósítjuk meg.
- A játékot három különböző méretű pályán lehet játszani:  $12 \times 12$ ,  $24 \times 24$ ,  $36 \times 36$ . Alapértelmezetten a  $24 \times 24$  van kiválasztva
- Az ablakon megjelenítjük a következő menüsort. Fájl (Megnyitás..., Mentés...), Új játék (Pályaméret ( $12 \times 12$ ,  $24 \times 24$ ,  $36 \times 36$ )). Az ablak alján elhelyezett állapotsoron látszik a játék kezdete óta eltelt idő, illetve az aktuális állapotban elérhető akció gomb a játék szüneteltetésére, illetve folytatására.
- A játék táblázatos elrendezést használ, az elemek ebben a táblázatban mozognak
- Ha az egyik játékos nekimegy a másiknak, annak vagy a saját fénycsíkjának, illetve ki akarna menni a pályáról, akkor felugrik egy dialógusablak, hogy a másik játékos nyerte a játékot.

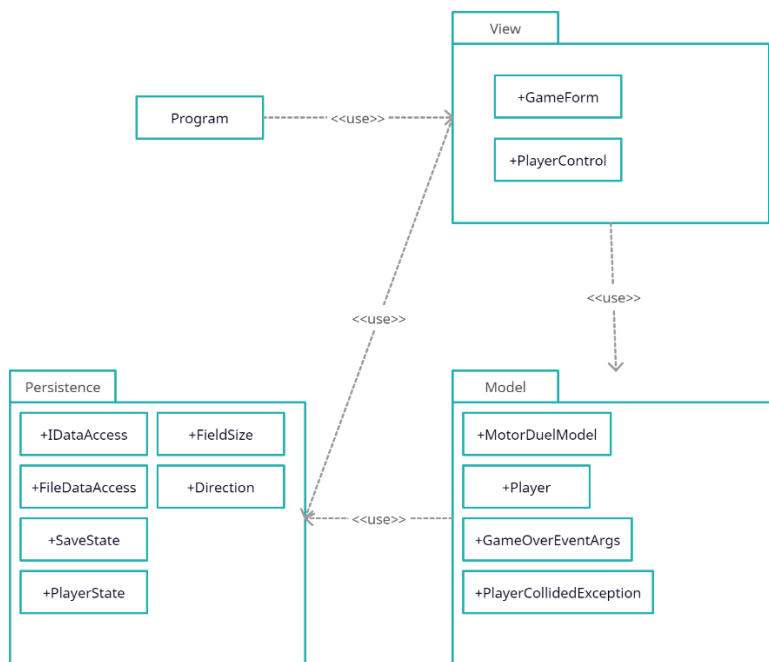


1. ábra: Felhasználói esetek diagramja

## Tervezés

### Szerkezet

- A programot 3 rétegű szerkezetben valósítjuk meg. A **model** felelős a logikáért, a **view** a megjelenítésért, a **persistence** pedig az adatmentésért és olvasásért.



2. ábra: Az alkalmazás csomagdiagramja

## Perzisztencia

- A perzisztenciában vannak a különböző felsorolás típusok (**FieldSize**, **Direction**, **Turn**), illetve struktúrák az adat tárolására.
  - Az **IDataAccess** interfész a műveletek általános sémáját tartalmazza, a modell csak ezt kapja majd meg. A konkrét implementáció a **FileDataAccess** osztályban van.
  - A **PlayerState** struktúra a **Player** osztály adatait tartalmazza. A **ToFileString()** osztály átalakítja ezeket az adatokat szöveggé, hogy le lehessen menteni őket egy fájlba. A statikus **Parse()** metódus visszaalakítja az adatokat szöveges formából. A **SaveState** kettő **PlayerState** mezőt tartalmaz, illetve a pálya méretét és az eltelt időt.

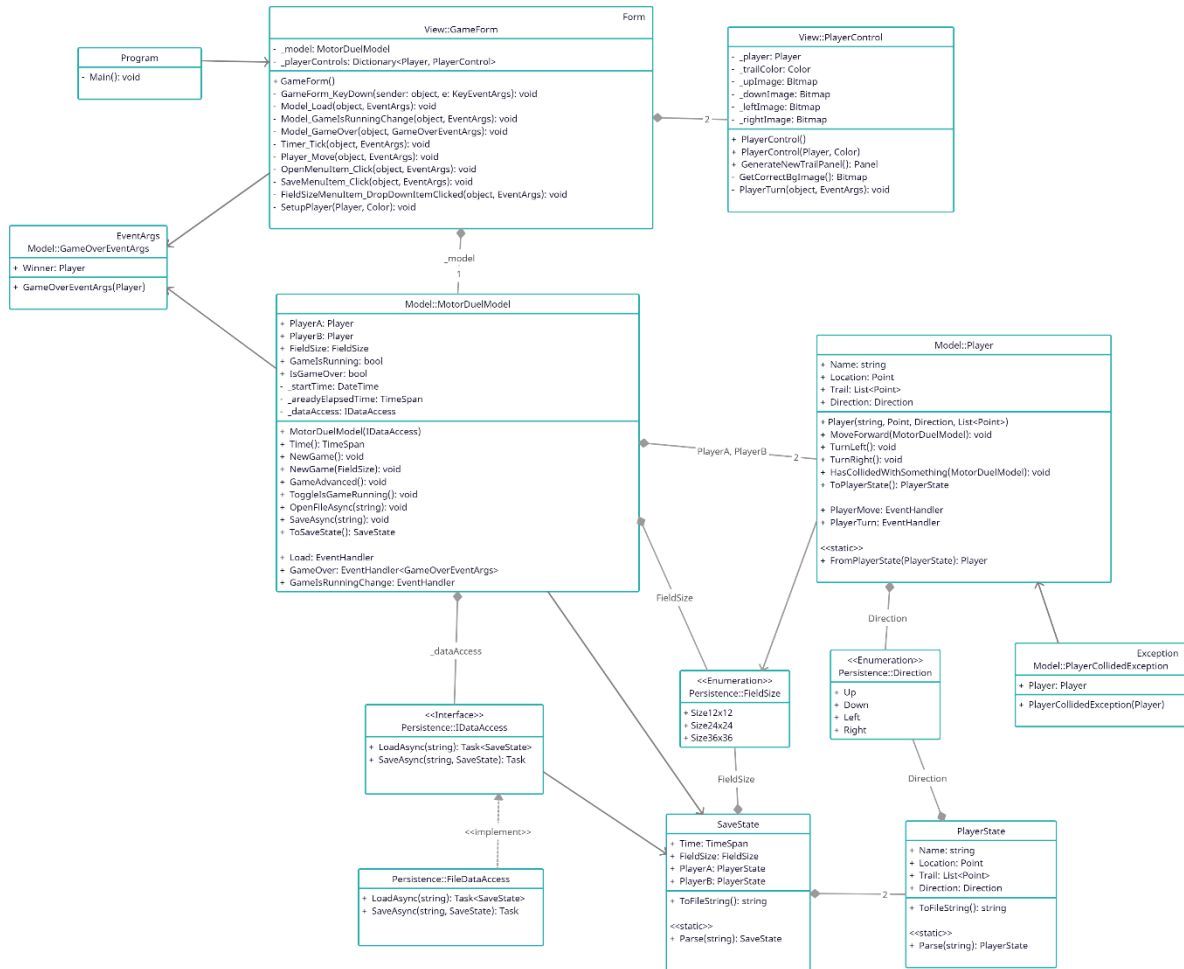
## Modell

- A modell központi eleme a **MotorDuelModel** osztály. Ez tárolja a játékos példányokat és a játék működési logikáját.
  - A **NewGame()** metódus felkészíti a modellt a játékra, illetve a **Load** eseménnyel jelzi a nézetnek, hogy újra kell rajzolni a pályát.
  - A **GameIsRunning** tulajdonság tárolja, hogy a játék éppen megy, vagy meg van-e állítva, amit a **ToggleGameIsRunning()** metódussal lehet változtatni.
  - Az **IsGameOver** tulajdonság jelzi, hogy a játéknak vége van-e már.
  - A **Time** metódus jelzi a játék kezdete óta eltelt időt. A játék megállításakor az időt nem számolja.

- A **GameAdvanced()** metódus kezeli a játék folyását, lépteti a játékosokat és figyeli, hogy vége van-e a játéknak
- Az **OpenFile()** és **Save()** metódusok elküldik, illetve megkapják a perzisztenciától az adatokat
- A játékosok **Player** példányok. Van pozíciójuk (**Location**), irányuk (**Direction**), tárolják a fénycsík helyét (**Trail**).
  - A **MoveForward()** metódus hatására megvizsgálják, hogy a pályán vannak-e még, és ha igen, akkor meghívják a **PlayerMove** eseményüket, aminek hatására a nézet átrajzolja őket a megfelelő helyre. Ha mozgás során a játékos nekiment a másik játékosnak, annak a fénycsíkjának, vagy a pálya szélének, akkor dobunk egy **PlayerCollidedException**-t.
  - A **TurnLeft()** és **TurnRight()** metódusok fordítanak a játékos irányán, és ezt **PlayerTurn** eseménnyel jelzik a nézetnek.

## Nézet

- Az ablakért a **GameForm** osztály felelős, ebben létrehozuk a **MotorDuelModel** egy példányát és hozzákapcsoljuk az eseményeit.
  - Az osztályban főleg eseménykezelők vannak. Rajtuk kívül van egy **\_playerControls** szótár, ahol a kulcsok **Player** példányok, az értékek pedig **PlayerControl**lok.
  - A **SetupPlayer()** metódus kap egy játékost és egy szint és ezekkel létrehozza a megfelelő **PlayerControl** példányt
  - A pálya egy **TableLayoutControl** példány, amit a **Model\_Load** eseménykezelő meghívásakor a megfelelő méretre hozunk és elhelyezzük benne a **PlayerControl**lokat.
  - A **timer** időzítő **Tick** eseményére frissítjük az adatokat az ablakon, illetve meghívjuk a **MotorDuelModel GameAdvanced()** metódusát, ami lépteti a játékosokat.
- A **PlayerControl** osztály felelős a játékosok megjelenítéséért. Betöltik a motorok képét és a **Player** osztály **PlayerTurn** eseményére a megfelelő irányban álló képet teszik be háttérképként. A **GenerateNewTrailPanel()** metódus csinál egy új **Panel** példányt, amit a **GameForm** beilleszt a megfelelő helyre a játékos mozgásakor.



## Tesztelés

A modell egységtesztekkel lett tesztelve a `MotorDuelModelTests` osztályban. Ehhez az MSTest tesztelő könyvtárat használtuk.

- A **PlayerCollidesWithSide()**, **PlayerCollidesWithOwnTrail()**, **PlayerCollidesWithOthersTrail()** metódusokban teszteljük, hogy az ütközések típusainál kiváltódig-e a **PlayerCollidedException** kivétel.
- A **PlayerMovesWell()** metódusban teszteljük, hogy a játékos megfelelően mozog és kanyarodik a pályán.
- A **DataLoadsWellFromPersistence()** metódus teszteli, hogy a perzisztenciától kapott adatokat megfelelően használja-e fel a modell.