# 4. Beadandó feladat dokumentáció

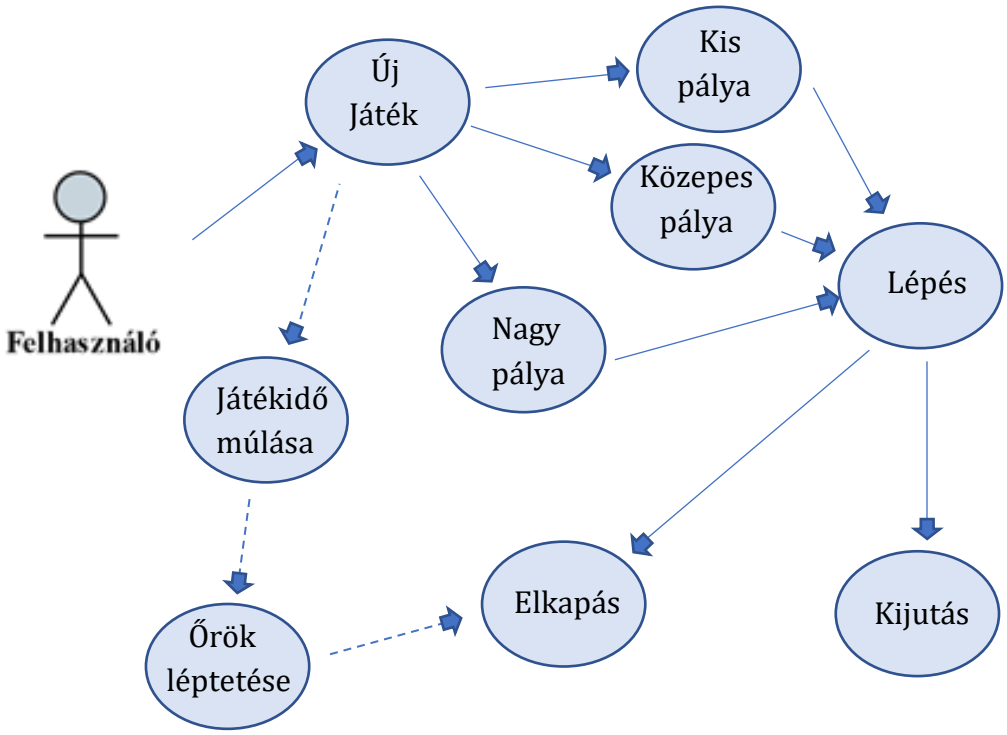
**Készítette:**
Pósa Gergely Tibor
HBZ5H1
gergelyposa@gmail.com

**Feladat:**

Készítsünk programot, amellyel a következő játékot játszhatjuk. Adott egy $n \times n$ elemből álló játékpálya, amely falakból és padlóból áll, valamint őrök járőröznek rajta. A játékos feladata, hogy a kiindulási pontból eljusson a kijáratig úgy, hogy közben az őrök nem látják meg. Természetesen a játékos, illetve az őrök csak a padlón tudnak járni. Az őrök adott időközönként lépnek egy mezőt (vízszintesen, vagy függőlegesen) úgy, hogy folyamatosan előre haladnak egészen addig, amíg falba nem ütköznek. Ekkor véletlenszerűen választanak egy új irányt, és arra haladnak tovább. Az őr járőrözés közben egy 2 sugarú körben lát (azaz egy 5 × 5-ös négyzetet), ám a falon nem képes átlátni. A játékos a pálya előre megadott pontján kezd, és vízszintesen, illetve függőlegesen mozoghat (egyesével) a pályán. A pályák méretét, illetve felépítését (falak és kijárat helyzete, játékos és őrök kezdőpozíciója) tároljuk fájlban. A program legalább 3 különböző méretű pályát tartalmazzon. A program biztosítson lehetőséget új játék kezdésére a pálya kiválasztásával, valamint játék szüneteltetésére (ekkor nem telik az idő, és nem léphet a játékos). Továbbá ismerje fel, ha vége a játéknak. Ekkor jelenítse meg, hogy győzött, vagy veszített-e a játékos.

**Elemzés:**

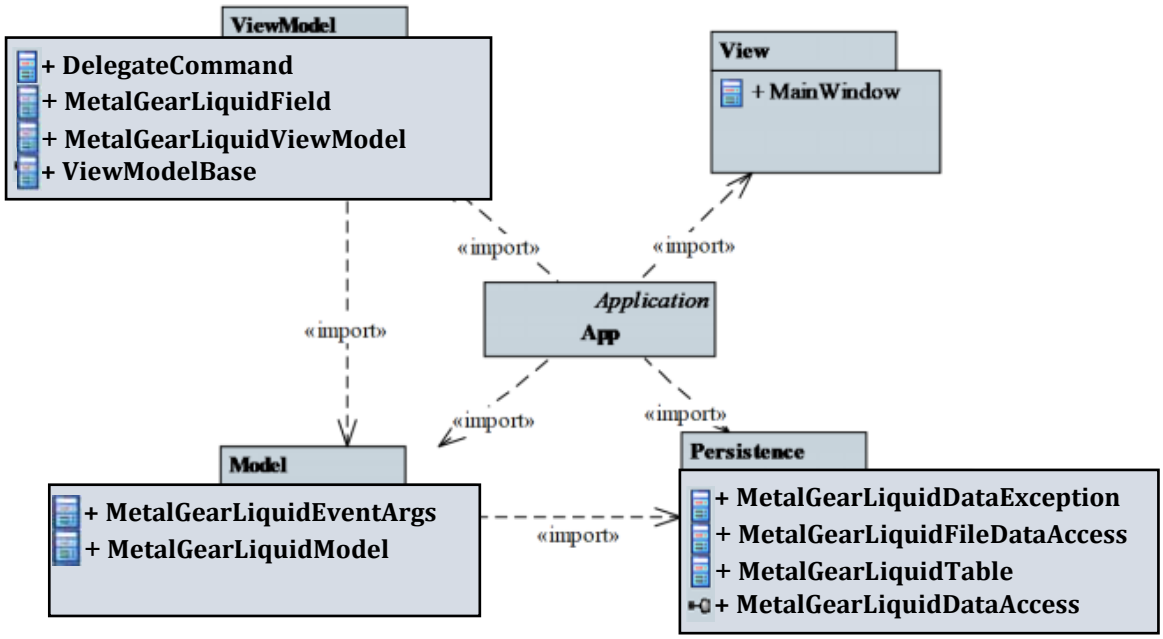
- A játék három pályát tartalmaz, melyek különböző méretűek. 11x15, 12x22 és 15x30. Ezeken különböző elrendezéssel falak, őrök és a játékos.
- A feladatot egyablakos asztali alkalmazásként Windows Presentation Foundation grafikus felülettel valósítjuk meg.
- Az ablakban elhelyezünk egy menüt a következő menüpontokkal: File (Új játék, Játék szüneteltetése, Kilépés), Beállítások (Kis pálya, Közepes pálya, Nagy pálya). Az ablak alján megjelenítünk egy státuszsort, amely az eltelt időt jelzi.
- A játkban 5 fajta mezőt különböztetünk meg: Játékos, Őr, Padló, Fal, Cél
- A játéktáblát egy $n_x$m-es gombokból álló rács reprezentálja. A gomb a w,a,s,d inputok egyikére megváltoztatja a játékos pozícióját, amennyiben nem ütközik falba és nincs megállítva a játék.
- A játék automatikusan feldob egy dialógusablakot, amikor vége a játéknak (kijutottunk vagy elkaptak).
- 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.



- Perzisztencia:
- Az adatkezelés feladata a _table táblával kapcsolatos információk tárolása, valamint a pálya betöltés biztosítása.
- A MetalGearLiquidTable osztály egy érvényes páylát biztosít (azaz mindig ellenőrzi a beállított értékek), ahol minden mezőre ismert az értéke (**`_fieldValues`**). Ezt játék kezdetekor egy filból kiolvasva kapjuk meg, a megfelelő pálya kiválasztásával. A pálya lehetőséget az állapotok lekérdezésére:

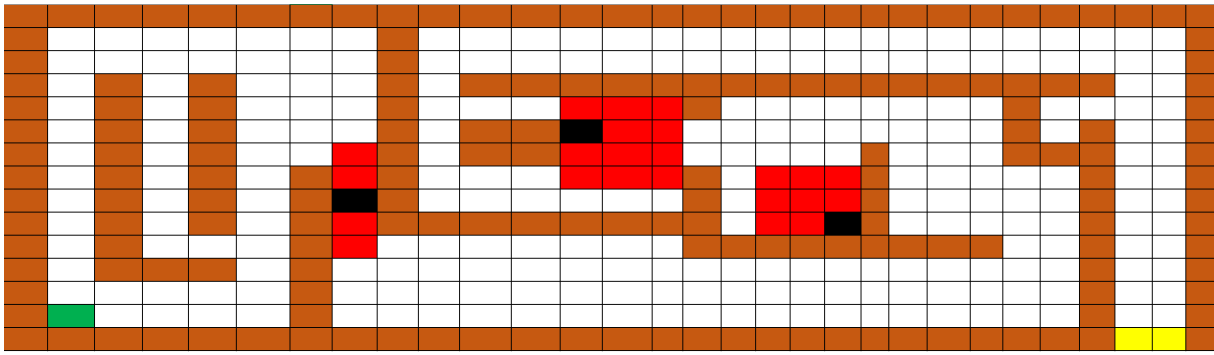
  Snek – Játékos pozíciója, Guards – Az őrök pozíciói, SpotPoints – Az őrök látóterének pozíciói, FieldValues – az egyes pálya beli pozíciók értéke.
- A hosszú távú adattárolás lehetőségeit az **`MetalGearLiquidDataAccess`** interfész adja meg, amely lehetőséget ad a tábla betöltésére (LoadAsync). A műveleteket hatékonysági okokból aszinkron módon valósítjuk meg.
- Az interfészt szöveges fájl alapú adatkezelésre a **`MetalGearLiquidFileDataAccess`** osztály valósítja meg. A fájlkezelés során fellépő hibákat a

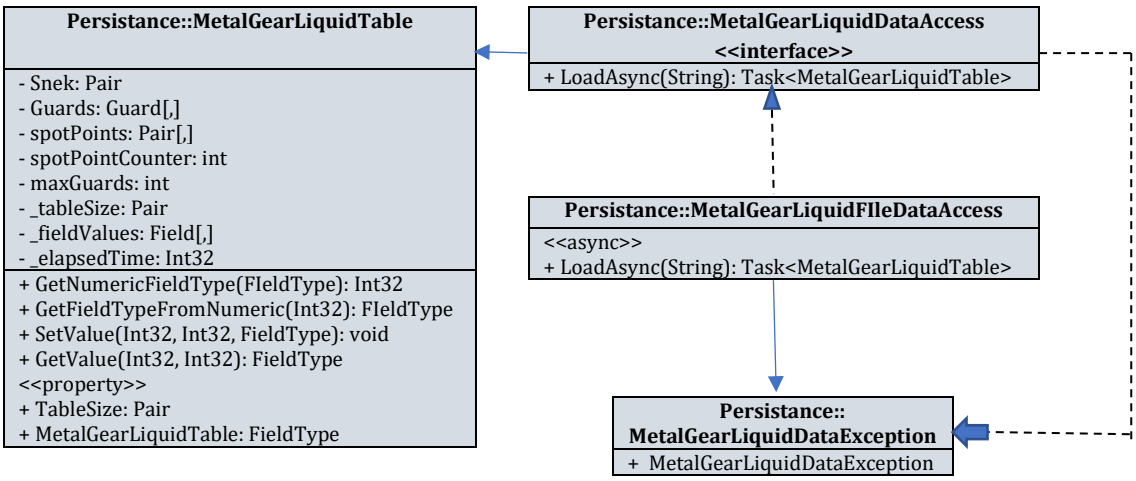
  **`MetalGearLiquidDataException`** kivétel jelzi.
- A program az adatokat szöveges fájlként tudja eltárolni, melyek az **`stl`** kiterjesztést kapják. Ezeket az adatokat a programban bármikor be lehet tölteni.
- A fájl első sora megadja a tábla méretét, az óra állását, és hogy hány őr van jelen. A fájl többi része izomorf leképezése a játéktáblának, azaz összesen n sor következik, és minden sor m számot tartalmaz szóközökkel választva. A számok 0-4 közöttiek lehetnek, ahol a számok jelentése:

  0 – Játékos, 1 – Őr, 2 – Fal, 3 – Padló, 4 – Kijárat

  Ehhez vizuális reprezentáció a 2. ábra


Guard's vision
Floor
Player
Wall
Exit
Guard




Persistance::MetalGearLiquidTable
- Snek: Pair
- Guards: Guard[,]
- spotPoints: Pair[,]
- spotPointCounter: int
- maxGuards: int
- _tableSize: Pair
- _fieldValues: Field[,]
- _elapsedTime: Int32
+ GetNumericFieldType(FIeldType): Int32
+ GetFieldTypeFromNumeric(Int32): FIeldType
+ SetValue(Int32, Int32, FieldType): void
+ GetValue(Int32, Int32): FieldType
<<property>>
+ TableSize: Pair
+ MetalGearLiquidTable: FieldType
Persistance::MetalGearLiquidDataAccess
<<interface>>
+ LoadAsync(String): Task<MetalGearLiquidTable>
Persistance::MetalGearLiquidFIleDataAccess
<<async>>
+ LoadAsync(String): Task<MetalGearLiquidTable>
Persistance::
MetalGearLiquidDataException
+ MetalGearLiquidDataException

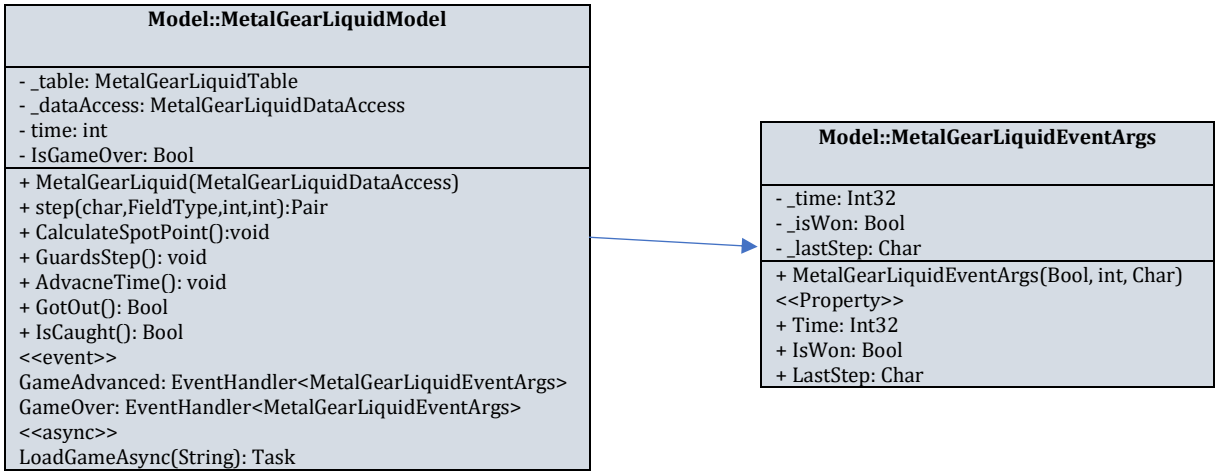
Modell:

A modell lényegi részét a **`MetalGearLiquidModel`** osztály valósítja meg, amely szabályozza a pálya tevékenységeit, valamint a játék egyéb paramétereit, úgymint az idő (**`time`**). A típus lehetőséget ad új pálya betöltésére (LoadGame) valamint a lépésre (step). Az idő előreléptetését időbeli lépések végzéséve (**`AdvanceTime`**) tehetjük meg. Ebben 2 függvény is szerepel, az egyik (GuardStep) az őröket lépteti abba az irányba amerre néznek épp, illetve egy másikat (CalculateSpotPointss), amely az őrök új pozíciói alapján kiszámolja a látóterüket.

A játékállapot változásáról a **`GameAdvanced`** esemény, míg a játék végéről a **`GameOver`** esemény tájékoztat. Az események argumentuma

**`MetalGearLiquidEventArgs`**) tárolja a győzelem állapotát, valamint a játékidőt.

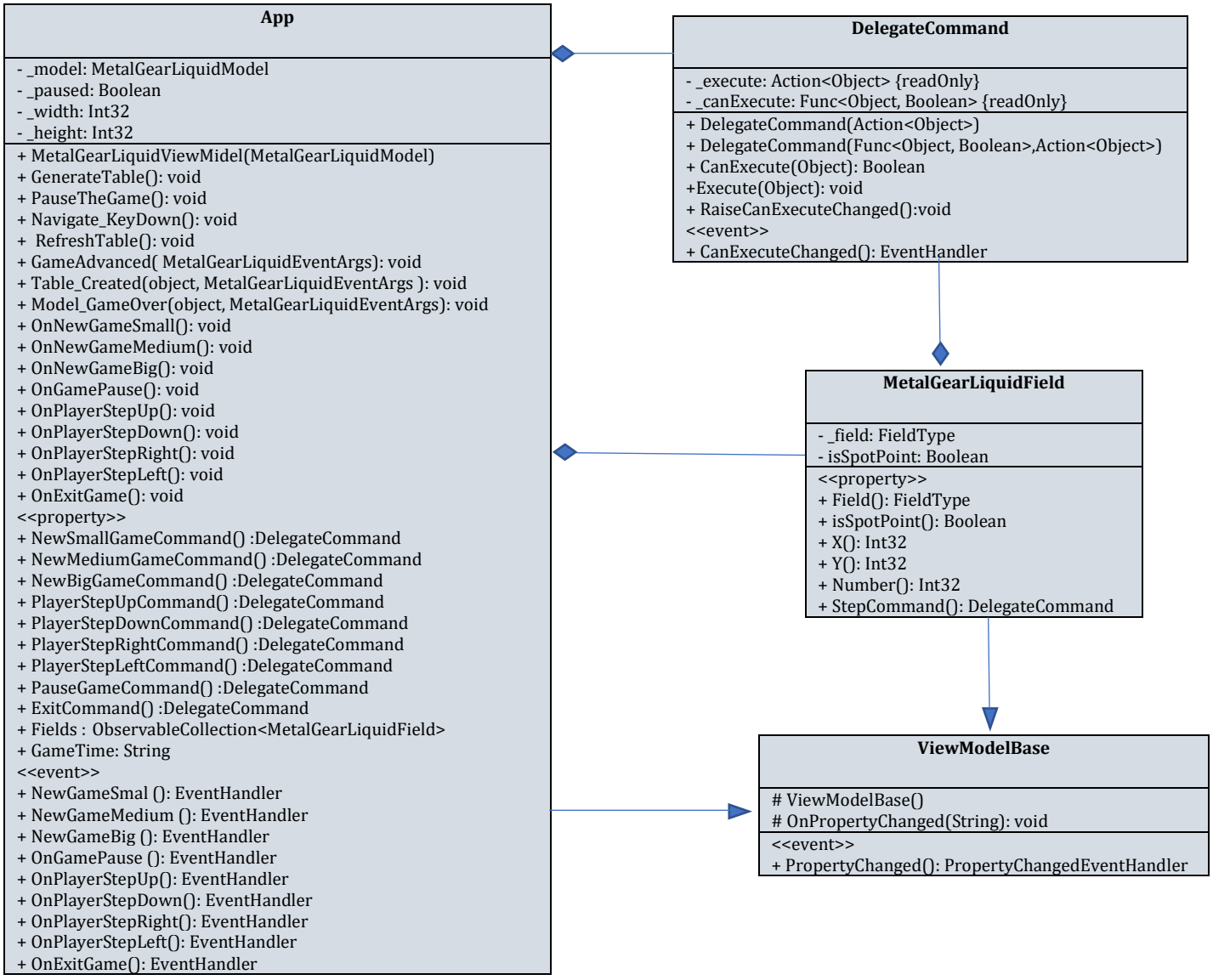
A modell példányosításkor megkapja az adatkezelés felületét, amelynek segítségével lehetőséget ad betöltésre (**`LoadGameAsync`**)



Nézetmodell:

- A nézetmodell megvalósításához felhasználunk egy általános utasítás (DelegateCommand), valamint egy ős változásjelző (ViewModelBase) osztályt.
- A nézetmodell feladatait a SudokuViewModel 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 játéknehézséget szabályozza. Direkt nem avatkozik a játék futtatásába.

- A játékmező számára egy külön mezőt biztosítunk (SudokuField), amely eltárolja a pozíciót, szöveget, engedélyezettséget, valamint a lépés parancsát (StepCommand). 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 egy ItemsControl vezérlő, ahol dinamikusan felépítünk egy rácsot (UniformGrid), amely címkékből áll. Minden adatot adatkötéssel kapcsolunk a felülethez, továbbá azon keresztül szabályozzuk a címkéken megjelenő grafikákat is.

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.

- A játék léptetéséhez tárol egy időzítőt is (_timer), amelynek állítását is szabályozza az egyes funkciók hatására.

| **App** |
| --- |
| - _model: MetalGearLiquidModel<br>- _viewModel: MetalGearLiqudViewModel<br>- _view: MainWindow<br>- _timer: DispatcherTimer<br>- timerStarted: Boolean<br>- isGameOver: Boolean |
| + App()<br>+ AppStartup(object, StartupEventArgs): void<br>+ TimerTick(object, EventArgs): void<br>+ View_Closing(object, CancelEventArgs): void<br>+ ViewModel_NewGameSmall(Object, EventArgs): void<br>+ ViewModel_NewGameMedium(Object, EventArgs): void<br>+ ViewModel_NewGameBig(Object, EventArgs): void<br>+ VieModel_PauseGame(Object, EventArgs): void<br>+ ViewModel_PlayerStepUp(Object, EventArgs): void<br>+ ViewModel_PlayerStepDown(Object, EventArgs): void<br>+ ViewModel_PlayerStepRight(Object, EventArgs): void<br>+ ViewModel_PlayerStepLeft(Object, EventArgs): void<br>+ ViewModel_ExitGame (Object, KeyEventArgs): void<br>+ GameOver(Object, MetalGearLiquidEventArgs):void |

## Tesztelés:

- A modell funkcionalitása egységtesztek segítségével lett ellenőrizve a **`MetalGearLiquidGameModelTest`** osztályban.
- Az alábbi tesztesetek kerültek megvalósításra:
- **`SmallMapLoadTest`**,

  **`MediumMapLoadTest`**,

  **`LargeMapLoadTest`**: Új játék indítása, a pályák betöltése, a program értékeinek helyes be/feltöltése.
- **`GameModelStepTest`**: A játékos lépése által történt változások ellenőrzése

- **`GameTimeAdvancedTest`**: A játékbeli idő múlásának vizsglálata, az őrök mozgásának megvalósulása, az őrök látóterének újrakalkulálása.
- **`GameOverTest`**: Az exit-re való lépéssel való gameover, az elkapással való gameover