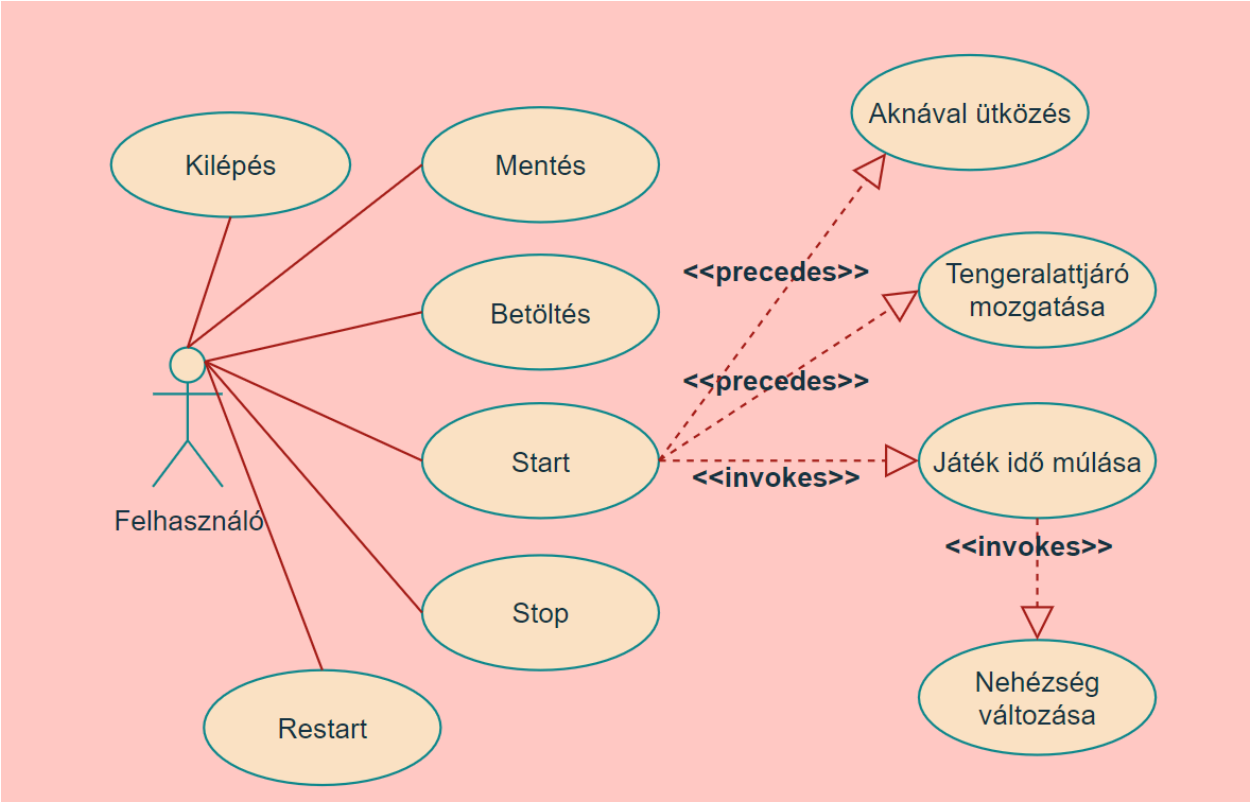
2. Beadandó feladat dokumentáció
Bene Zakariás UAUYQF

Feladat: 10. Aknamező
Készítsünk programot a következő játékra.
A játékban egy tengeralattjárót kell irányítanunk a képernyőn (balra, jobbra, fel, illetve le), amely felett ellenséges hajók köröznek, és folyamatosan aknákat dobnak a tengerbe. Az aknáknak három típusa van (könnyű, közepes, nehéz), amely meghatározza, hogy milyen gyorsan süllyednek a vízben (minél nehezebb,
annál gyorsabban).
Az aknákat véletlenszerűen dobják a tengerbe, ám mivel a hajóskapitányok egyre
türelmetlenebbek, egyre gyorsabban kerül egyre több akna a vízbe. A játékos célja az, hogy minél tovább elkerülje az aknákat. A játék addig tart, ameddig a tengeralattjárót el nem találta egy akna.
A program biztosítson lehetőséget új játék kezdésére, valamint játék szüneteltetésére (ekkor nem telik az idő, és nem mozog semmi a játékban). Ismerje fel, ha vége a játéknak, és jelenítse meg, mennyi volt a játékidő. Ezen felül
szüneteltetés alatt legyen lehetőség a játék elmentésére, valamint betöltésére.

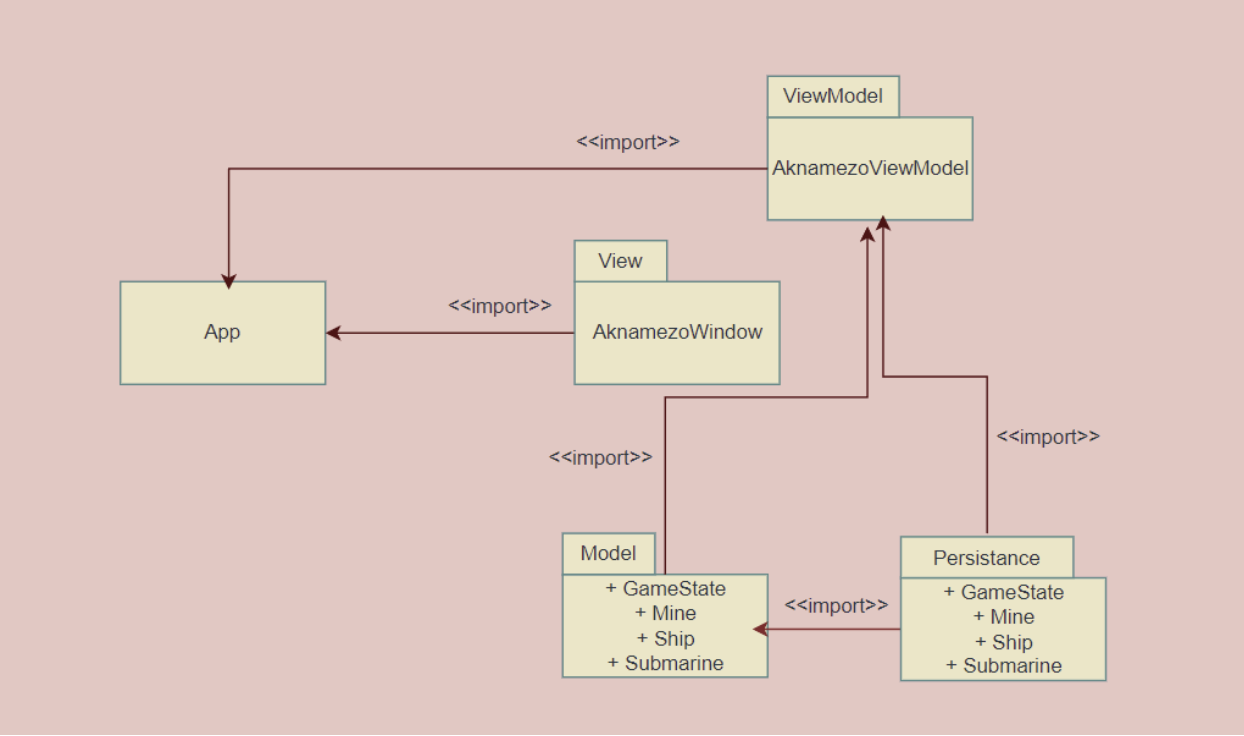
Elemzés:

- A játék az alap (Easy) nehézségen indul és ahogy megy az idő nehezedik. Összesen 4 nehézségi szint van: Easy, Normal, Hard, Death. Minél nehezebb a játék annál gyorsabban dobálják a hajók az aknákat a tengerbe.
- A feladat egy ablakos WPF applikáció formájában van megvalósítva.
- A játék az ablak közepén található Canvas-on játszódik. A W,A,S,D gombok nyomására mozgatható a játékos, ha már a játék elindult (START gomb meglett nyomva). A játéknak vége, ha a játékos ütközik egy aknával. Ekkor egy MessageBox kiírja a játék időt, majd a játék leáll. Új játékot a RESTART gomb megnyomásával lehet indítani.
- A Canvas alatt találhatóak a menü gombok. START(elindít egy játékot), STOP(megállítja a játékot), RESTART (újra indítja a játékot), SAVE (kinyit egy SaveFileDialogue-t a játék elmentésére), LOAD (kinyit egy OpenFileDialogue-t egy régebben elmentett játék beolvasására).
- A felhasználói esetek az alábbi ábrán találhatók:



Tervezés:

- Programszerkezet:
  - A programot MVVM architektúrában valósíjuk meg. A megjelenítés a View, a modell a Model, a perzisztencia a Persistence névtérben helyezkedik el és a ViewModel a ViewModel névtérben helyezkedik el. A program csomagszerkezete a 2. ábrán látható.
  - A projekt szerkezete 2 részre van osztva. Egy Class Library, ami a Modellt és a Perzisztenciát tartalmazza és egy WPF applikáció, ami a Viewt és ViewModelt tartalmazza.
- Perzisztencia:
  - A Persistence névtérben a **IFileManager** interface és **JsonFileManager** osztály található.
  - Az **IFileManager** interface-nek 2 metódusát kell implementálni: Save és Load.
  - A **JsonFileManager** osztály implementálja a 2 metódust. A Save metódus lehetővé teszi egy **GameState** példány JSON formátumban való elmentését. A Load metódus lehetővé teszi egy JSON fájlból való **GameState** betöltését.

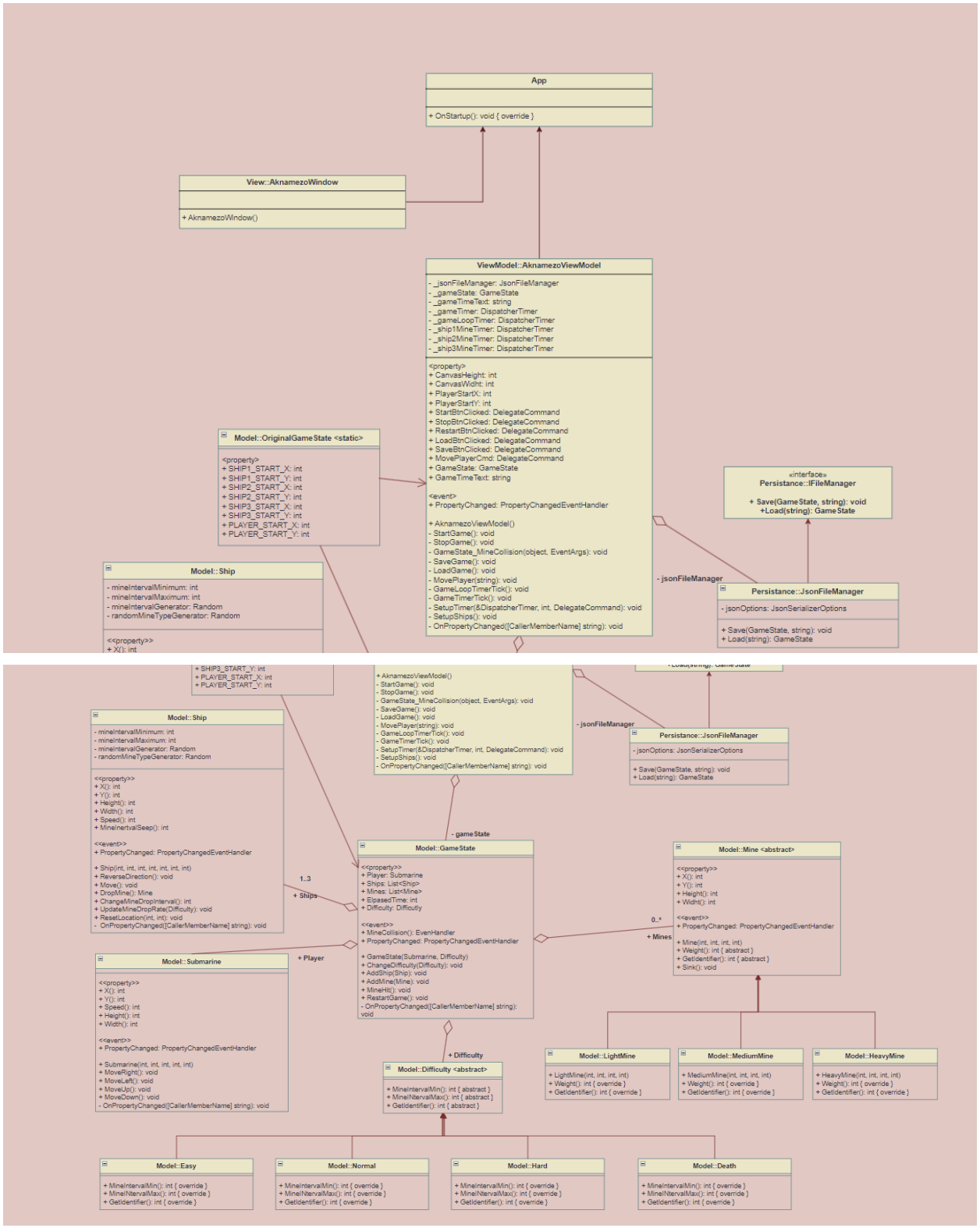


- Modell:
  - A modellt több osztály valósítja meg. A **GameState** osztály tárolja a játék releváns adatait (ElapsedTime, Player, Ships, Mines, Difficulty).
  - A **Submarine** osztály tárolja a játékos adatait (pozíció a pályán, sebesség). Ebben az osztályban vannak implementálva a játékosnak szükséges metódusai, mint például a játékos mozgatása (MoveRight, MoveUp…).
  - A **Ship** osztály tárolja az ellenséges hajó adatait (hogy milyen gyorsan dobjon aknákat, pozíció, sebesség). Itt vannak megvalósítva a szükséges metódusok a hajókhoz (DropMine - ledob egy új aknát és átállítja a MineIntervalSpeedet a nehézséghez megfelelően, Move ü mozgatja a hajót).
  - A **Mine** osztályban tároljuk az egyes aknák adatait és műveleteit. 3 leszármazottja van: LightMine, MediumMine, HeavyMine. Ezek határozzák meg, hogy milyen sebességgel süllyednek az aknák.
  - A **Difficulty** osztálynak 4 leszármazottja van. Easy, Normal, Hard, Death. Ezektől függ, hogy milyen intervallumon választják a hajók a következő aknájuk ledobását. Ezeket az adatokat a MineIntervalMin és MineIntervalMax metódusok adják meg nekünk.
  - A **MineCollision** event. Ezt az eseményt a MineHit metódus váltja ki. Ezt a metódust a GameState osztályban találhatjuk és a AknamezoViewMode hívja meg minden _gameLoopTimerTick esetén. Ha

a játékos ütközött egy aknával, akkor Invoke-olja ezt az eseményt, ami a játék véget jelenti.

- Nézet:
    - A nézetet a **AknamezőViewModel** osztály kezeli, ami tartalmazza a **GameState** egy példányát. Az App.xaml.cs fájlban van a viewmodel a view-hoz kötve DataContext ként. Maga a view az AknamezoWindow-ban van megvalósítva.
    - A játék fájlba való elmentését és betöltését is a ViewModel kezeli a SaveBtnClicked, LoadBtnClicked **DelegateCommandok** és a **JsonFileManager** osztály használatával.
    - A **DelegateCommand** osztály implementálja az **ICommand** interfészt, ami által készithetünk parancsokat az egyes view-ban lévő objektumokhoz, amiket tudunk majd bindolni.
    - Maga a játék egy Canvas-ban játszódik. A **_gameLoopTimer** minden 16 tized másodpercben (kb. 60 FPS így a játék) tick-el és, ekkor frissítjük a játékot, azaz lerajzoljuk a hajókat, a játékost és az aknákat, így a játék nem akadozik és minden természetesnek tűnik.
    - A **_gameTimer** minden 1 másodpercben tick-el. Ekkor frissítjük a GameState-ben az Elapsed Time-ot és itt ellenőrizzük, hogy mikor kell nehézséget váltani.

Az alábbi kép a teljes osztálydiagram:



Tesztelés (MSTest Project):

- A **MineHitTest** ellenőrzi, hogy az aknák és a játékos ütközése helyesen működik és, hogy az ütközés esetén kiváltódik e a MineCollision esemény.
- A **DifficultyChangeTest** ellenőrzi, hogy a nehézség váltás esetén a hajók helyesen váltják át a aknadobási intervallumokat.
- A **GameMovementTest** ellenőrzi, hogy az egyes mozgatható osztályok helyesen mozognak.
- A **RestartGameTest** ellenőrzi, hogy miután a játékos újraindítja a játékot minden a helyes kezdetleges pozícióban lesz és, hogy minden akna törlődik e.
- A **SaveAndLoadGameTest** ellenőrzi, hogy a JsonFileManager osztály metódusai helyesen vannak e implementálva.