Eva Beadandó 2

Hartyányi Kevin
hartyanyi.kevin@gmail.com
C0S0RJ

November 2019

Contents

1 Feladat

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.

2 Elemzés

- A feladatot egyablakos asztali alkalmazásként Windows Presentation Foundation grafikus felülettel valósítjuk meg.
- Az ablakban elhelyezünk két menüt a következő menüpontokkal: File, Idő. Figyelni kell, hogy a felhasználó csak akkor tudjon menteni, ha előtte a játékot megállította a Stop Game megnyomásával.
- A játéktáblát egy 6x6 nyomógombokból álló rács reprezentálja. A nyomógombokal a w,a,s,d gombok használatával tudunk kommunikálni (mozgatni a játékost).
- A játék állapota adatbázisba menthető az Entity FrameworkORM keretrendszer használatával. A betöltést és a mentést egyedi dialógusablakokkal végezzük, a mentések egyedi nevéta felhasználó adja meg.
- A játék automatikusan elkezdődik indításnál. Menteni és betölteni a játék folyamán bármikor lehet, a fájlneveket a felhasználó adja meg.
- A felhasználói esetek a figure 1 ábrán láthatóak.

3 Tervezés

- Programszerkezet:
  - A programot MVVM architektúrában valósítjuk meg, ennek megfelelően View, Model,ViewModelés Persistence névtereket valósítunk meg az alkalmazáson belül.A program környezetét az alkalmazás osztály (App)

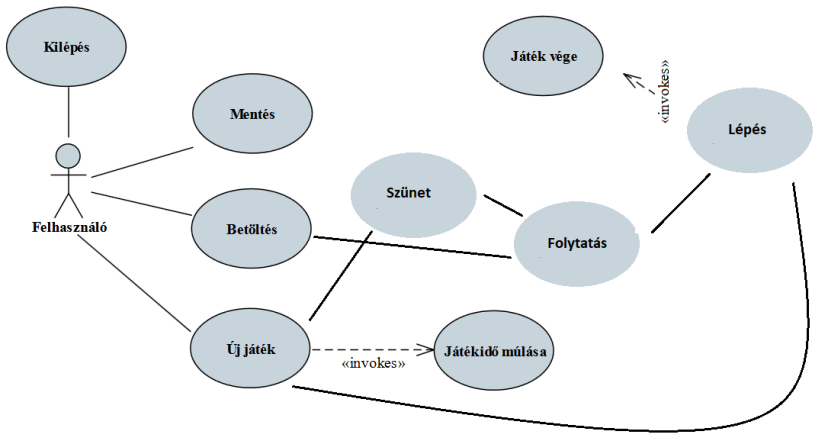

Figure 1: Felhasználói esetek

végzi, amely példányosítja a modellt, a nézetmodell és a nézetet, biztosítja a kommunikációt, valamint felügyeli az adatkezelést.A program csomagszerkezete a figure 2 ábrán látható.

- Perzisztencia
  - Az adatkezelés feladata a Model-el kapcsolatos információk, betöltés/mentés.
  - Az adatokat egy speciális osztály segítségével adjuk át a perzisztencia és a Model között, betöltés esetén.
  - A hosszú távú adattárolás lehetőségeit az IDataGame interfész adja meg, amely lehetőséget ad a játék betöltésére (Load), valamint mentésére (Save).
  - Az interfészt adatbázis alapú adatkezelésre a GameDbDataAccess osztály valósítja meg.Az adatbáziskezelés az Entity Framework használatával a Field és Game entitás típusokkal és a GameContext adatbázis kontextussal történik

- Modell
  - A modell lényegi részét a GameControlModell osztály valósítja meg, amely szabályozza a tábla tevékenységeit, valamint a játék egyéb paramétereit, úgy mint az idő (gameTime) és a nehézség (difficultyTime). A típus lehetőséget ad új játék kezdésére (NewGame), szüneteltetésére(StopGame), továbbá folytatására (StartGame). Új

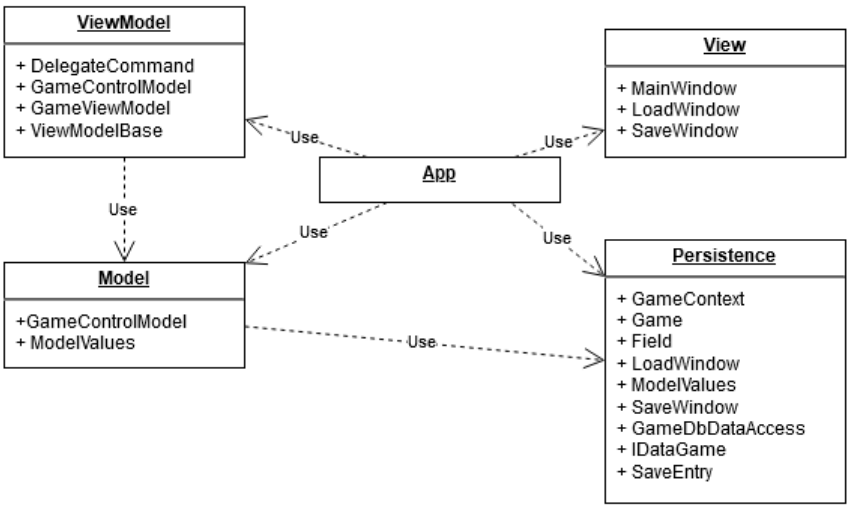

Figure 2: Az alkalmazás csomagdiagramja

játéknál megadható a kiinduló játéktábla is. Az idő előreléptetését Timer segítségével valósítjuk meg.

- A játék állapot változásáról különböző események tájékoztatják a nézetett (ShipMoveEvent, BombMoveEvent, BombCreateEvent, ...) a játék végéről a GameOverEvent esemény tájékoztat, ami az aktuális játékídőt is továbbadja.
- A modell példányosításkor megkapja az adatkezelés felületét, amelynek segítségével lehetőséget ad betöltésre (LoadGame) és mentésre (SaveGame) valamint a létező mentések lekérdezésére (ListGames).
- A játék nehézsége folyamatosan növekszik az idő elörehaladtával, ameddig a játékos meg nem hal.

- NézetModell
  - A nézetmodell megvalósításához felhasználunk egy általános utasítás (DelegateCommand), valamint egy ős változásjelző (ViewModelBase)osztályt.
  - A nézetmodell feladatait a GameViewModel 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.
  - A játékmező számára egy külön mezőt biztosítunk (GameField), amely eltárolja a pozíciót és színt. A mezőket egy felügyelt gyűjteménybe helyezzük a nézetmodellbe (Fields).

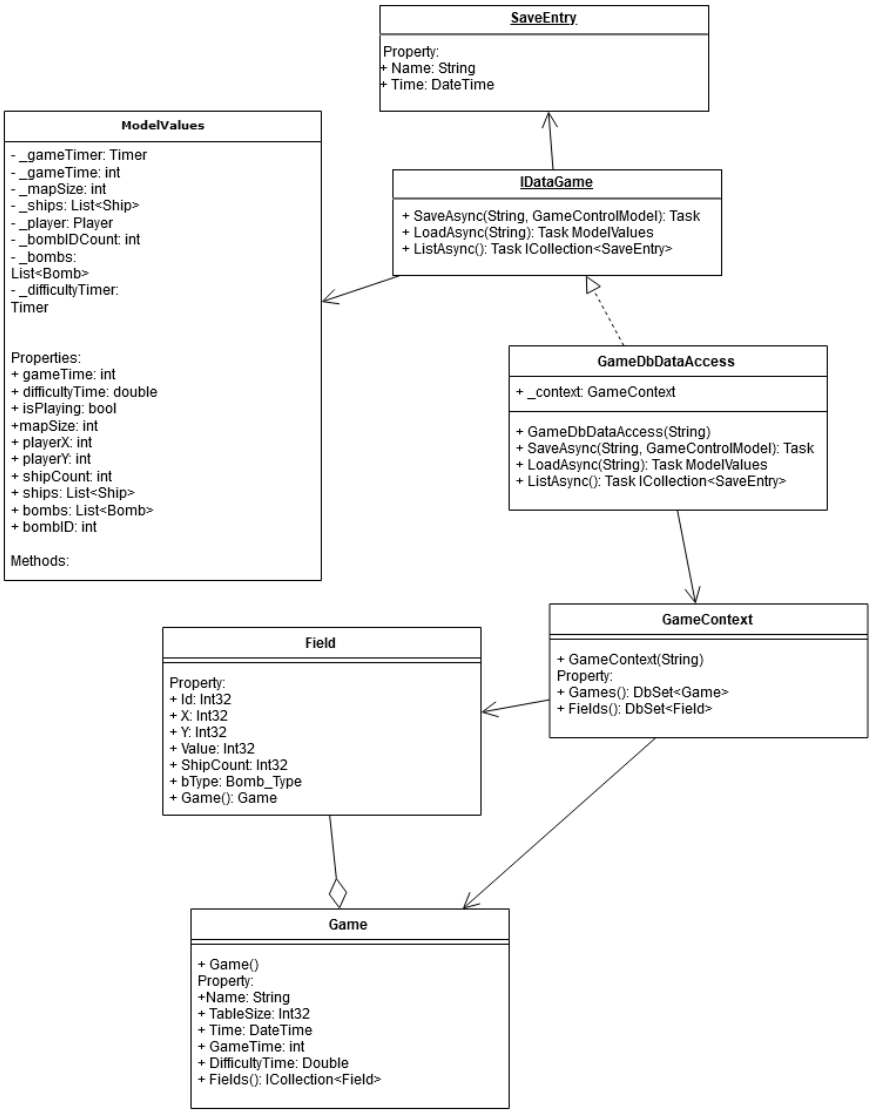

Figure 3: Perzisztencia

- Nézet
  - A nézet fő képernyőjét a MainWindowosztálytartalmazzaA nézet egy rácsban tárolja a játékmezőt, a menüt és a státuszsort. A játékmező egy ItemsControlvezérlő, ahol dinamikusan felépítünk egy rácsot (UniformGrid), amely gombokból áll. Minden adatot adatkötéssel kapc-

solunk a felülethez, továbbá azon keresztül szabályozzuk agombok színét is.

  - A betöltendő játékállapot bekérésért a LoadWindow osztály felel, amely dialógusablakként került megjelenítésre. A nézet egy ListBoxvezérlőben listázza ki az elérhető játékállapotokat.
  - Új mentésének nevét a SaveWindow osztály által megjelenített felület kéri be. A nézeten egy szövegdobozban (TextBox) megadható az új mentés neve, valamint a LoadWindow ablakhoz hasonlóan megjeleníti a létező mentések nevét (felülírás céljából).
  - A figyelmeztető és információs üzenetek megjelenését beépített dialógusablakok segítségével végezzük.

- Az osztálydiagram a figure **??** ábrán látható.

- Környezet

  - Az Apposztály 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.

4 Tesztelés

- A modell funkcionalitása egységtesztek segítségével lett ellenőrizve a Test osztályban.

- Az alábbi tesztesetek kerültek megvalósításra:

  - LoadTest: Teszteljük a játék betöltését
  - SaveTest: Teszteljük a játék mentését, és, hogy az adatok nem változnak a modelben ennek hatására
  - NewGame: Az új játék indítását teszteljük, megfelelően töltödik-e fel a model értékekkel.
  - GameOverTest: A játék végét teszteljük
  - BombCreateTest: Teszteljük, hogy megfelelően létrejönnek-e a bombák a modelben és erről eseményt küld e a form-nak.
  - PlayerMove: Teszteljük a játékos mozgatását mind a 4 irányba.

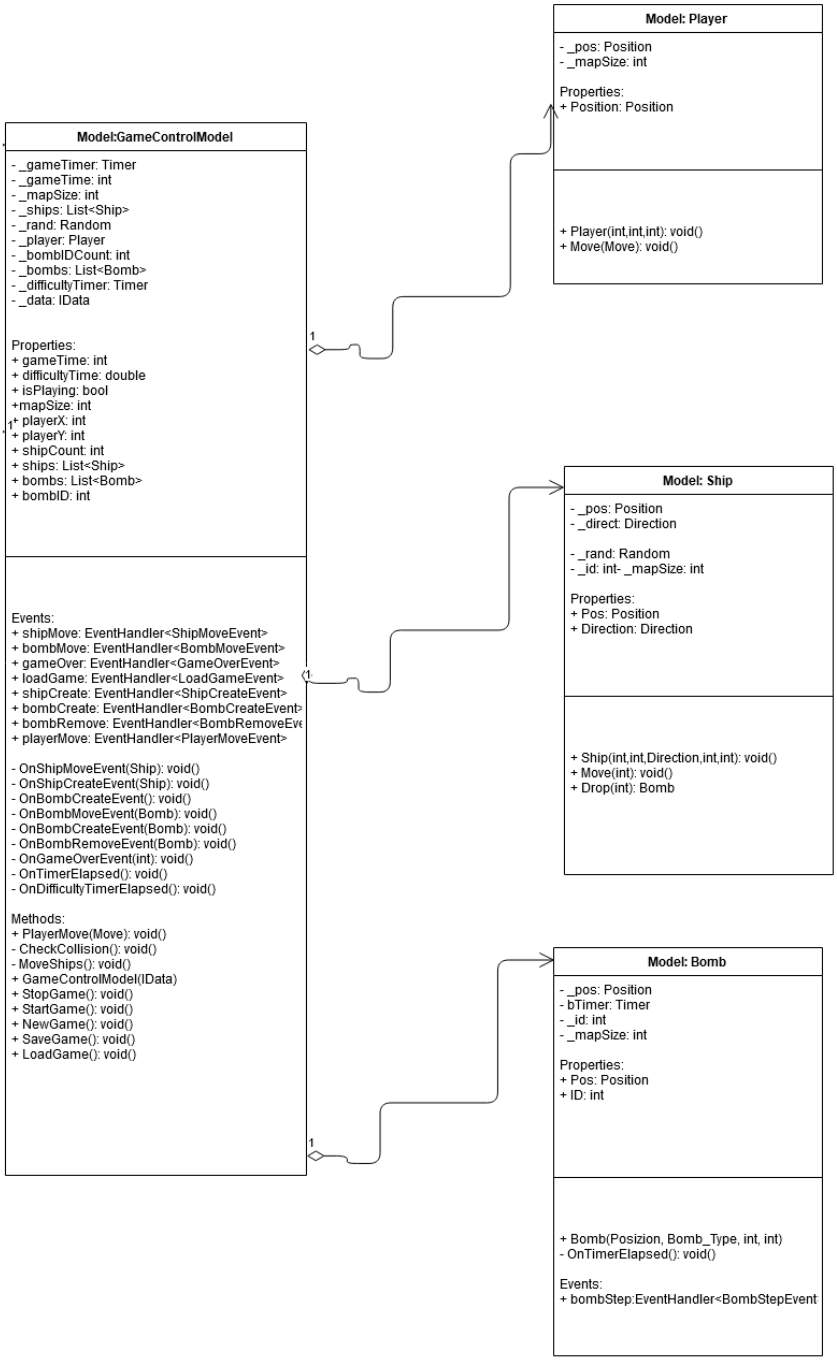

Figure 4: Model

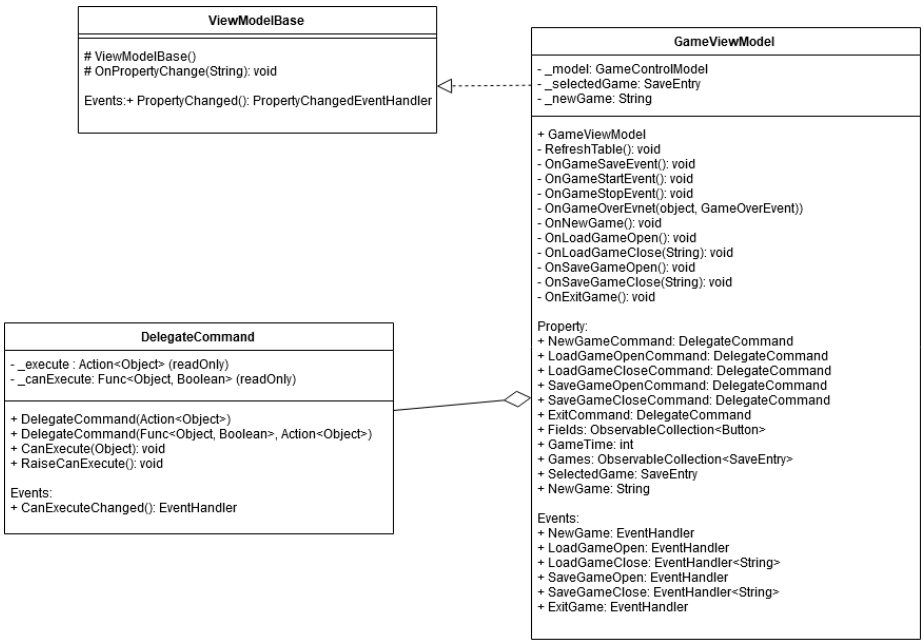

Figure 5: NézetModel

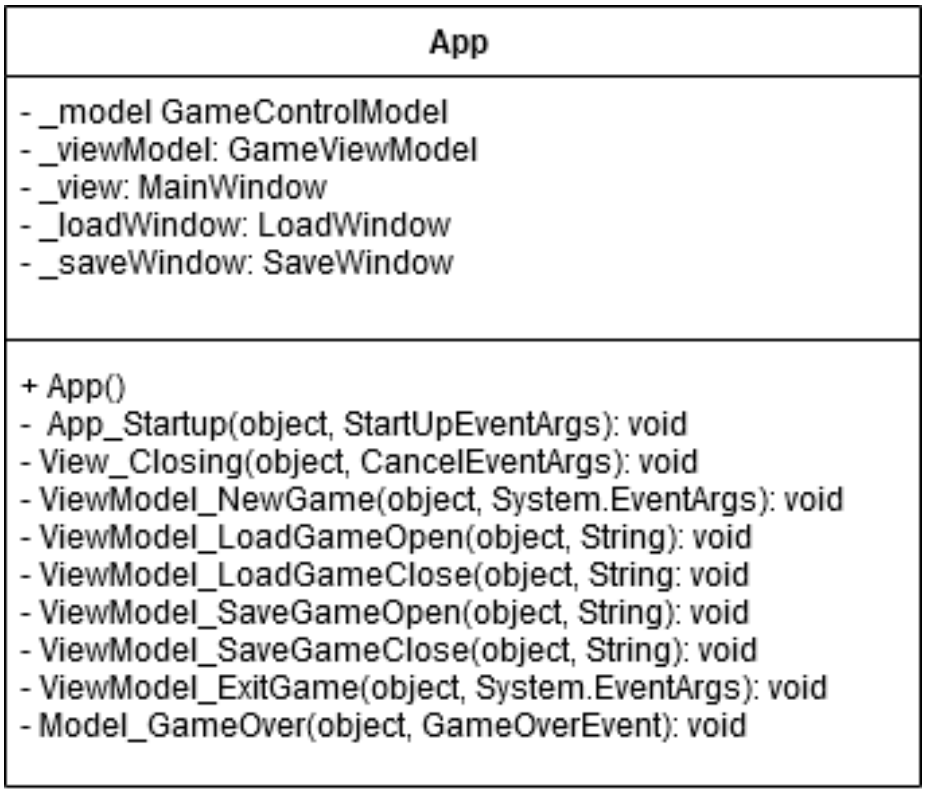
App
- _model GameControlModel
- _viewModel: GameViewModel
- _view: MainWindow
- _loadWindow: LoadWindow
- _saveWindow: SaveWindow
+ App()
- App_Startup(object, StartUpEventArgs): void
- View_Closing(object, CancelEventArgs): void
- ViewModel_NewGame(object, System.EventArgs): void
- ViewModel_LoadGameOpen(object, String): void
- ViewModel_LoadGameClose(object, String: void
- ViewModel_SaveGameOpen(object, String): void
- ViewModel_SaveGameClose(object, String): void
- ViewModel_ExitGame(object, System.EventArgs): void
- Model_GameOver(object, GameOverEvent): void

Figure 6: App