Dokumentáció

Bodor Zoltán, Hack János, Nagy Márton, Pető Bence

2013. december 4.

Tartalomjegyzék

1. fejezet

Követelmény Feltárás

a fajok részletes leíását tartalamző txt tartalam valahova sztem ide kell de nem tudom pontossan hova ....

1.1. Célkitűzés, projektindító dokumentum

A szoftver egy egy számítógépnél játszható körökre osztott startégiai játék lesz. A megrendelő elvárja a játéktól, hogy több ember, legalább kettő, képes legyen egymás ellen játszani. A felhasználó szeretne a játékban a saját választott nevével játszani. A játék egy pályán játszódik, ahol a játékosok különböző egységekkel rendelkeznek. A játék célja a másik játékos egységeinek legyőzése, vagy a pályán megtalálható pénzbeviteli források teljes uralma. A megrendelő további az alap játékon túli feature-öket is szívessen látna az ídő és erőforrás mennyiségétől függően. Ezek a következőek lennének:

- Saját pályák készítésének lehetősége.
- A megkezdett játékok elmentése, valamint visszatöltése.
- Játékos ranglétra, amin a legjobb 10 játékos látszik elért pont alapján.
- Csata közbeni súgó, amin meg tudja nézni agy egységek részletes leírását és a részletes játék szabályokat.
- Játék gépi játékos ellen, valamint 2 vs 2, hogy több ismerősével is játszhasson.

1.2. Szakterületi fogalomjegyzék

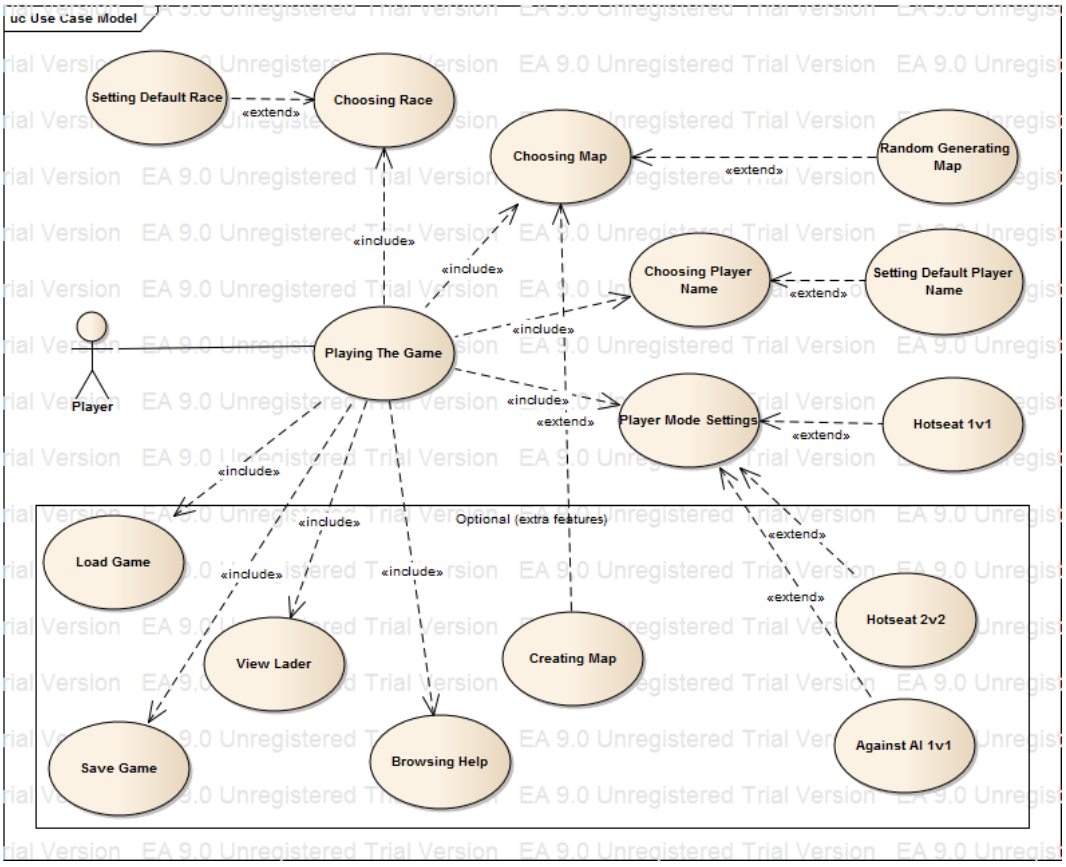
- **Hotseat**: Egy számítógép előtti, egy billenytűzettel és egérrel játszható, többjátékos mód.
- **Race**: A játékban használható fajok.
- **Lader**: A játékos ranglétra, itt szerepelnek a legjobb játékosok.
- **Körökre osztott stratégia**: A játék körökből áll, egy kör egy játékos összes lépését jelenti a tovább adásig a másik játékosnak.

1.3. Használatieset-modell, funkcionális követelmények

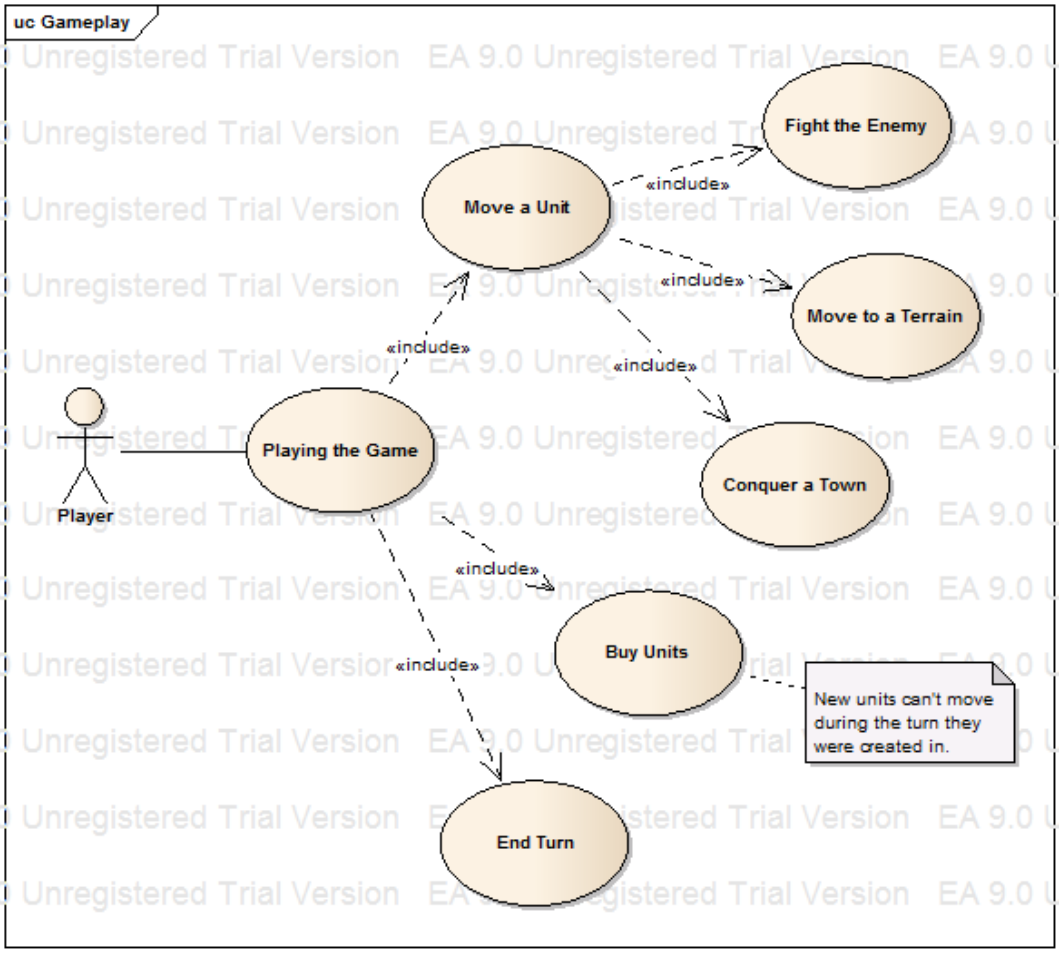
A Szoftver alapvető célja a játék. A Játék alapértelmezett eseben egy számítógép előtt ülő két emberek között zajlik. A játékosok választhatnak maguknak nevet a játék megkezdése előtt, ha nem választanának a játék az alapértelmezett neveket használja majd (Player1, Player2), amivel később a játékban hivatkozunk rájuk. A játékosok a nevük megválasztása után választhatnak az előre elkészített pályák közül, valamint random is generáltathatnak pályát, amin játszani fognak. A pálya kiválasztása után minden játékosnak lehetősége nyilik a játszani kívánt faj kiválasztására. Ezután megkezdődhet a játék.

A felhasználó továbbá extra featureként szívessen látná a következő dolgokat:

- Saját pályák készítésének lehetősége.
- A megkezdett játékok elmentése, valamint visszatöltése.
- Játékos ranglétra, amin a legjobb 10 játékos látszik elért pont alapján.
- Csata közbeni súgó, amin meg tudja nézni agy egységek részletes leírását és a részletes játék szabályokat.
- Játék gépi játékos ellen, valamint 2 vs 2, hogy több ismerősével is játszhasson.



1.1. ábra. Használatieset-modell



1.2. ábra. Játékmenet használatieset-modell

1.4. Szakterületi követelmények

1.5. Nem funkcionális követelmények

- A játék legyen könnyen átlátható, és használata a gyorsan tanulható.

2. fejezet

Tervezés

2.1. Program architecturája

A program három fő komponensből tevődik össze melyek az MVVM (Model-View-ViewModel) minta alap elemei. A kapcsolatok az egyes komponensek között a tervmintának megfelelően a következőek:

- **Model**: Az üzleti logika – a játék eseményeinek kezelése, számítások stb. – ebben a komponesben található. Ide kerül minden a játék logikai működéséért felelős osztály.
- **View**: Minden a megjelenítéssel kapcsolatos osztály, a különböző megjelenítendő sémák osztályai.
- **ViewModel**: A View és Model komponest köti össze, az ide tartozó osztályok felelősek a két téteg együttműködésért, semilyen számítást végző osztályt vagy felületi elemet nem tartalmazhat.

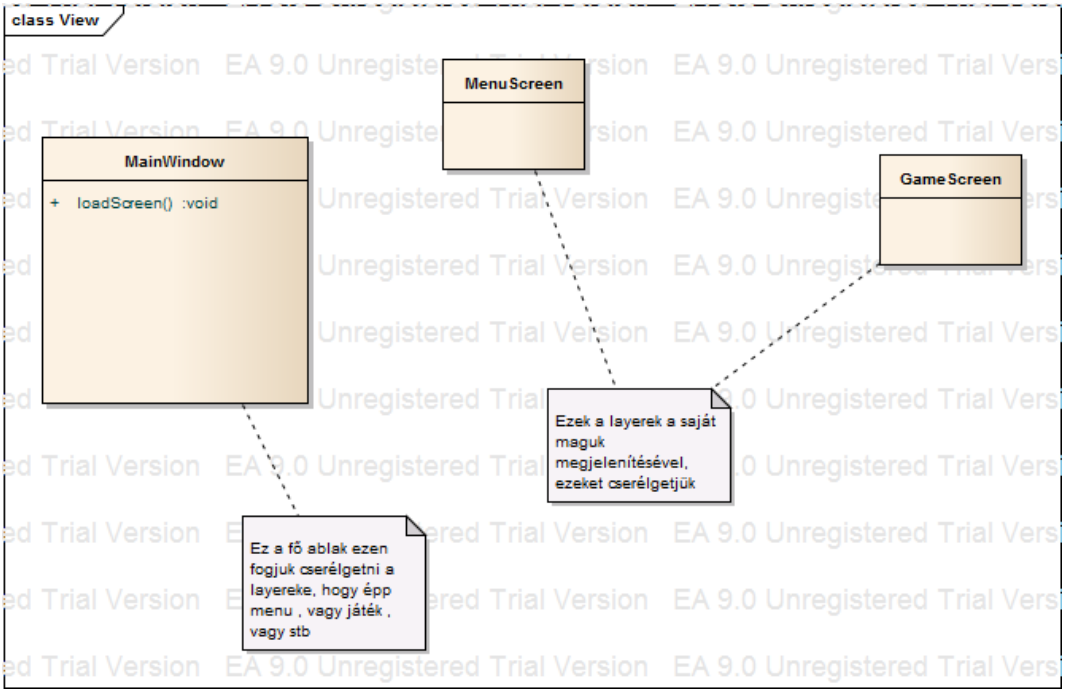
2.2. Osztálymodell

2.2.1. View

2.2.2. Model

A Model névtér tartalmazza a játék logikájához szükséges osztályokat. Itt tároljuk az aktuális játékállást, a térképet, a játékosok egységeit stb.

Gyakorlatilag egy kiszolgáló, egyetlen példány létezik belőle (singleton), minden adatbáziskérés vagy háttéradat-módosítás itt történik. Van néhány



2.1. ábra. View Osztálydiagram

funkció, ami visszaad értéket a ViewModelnek, a többi csak frissíti az adatagokat amik hozzá vannak kötve a ViewModelen kereszül a Viewhoz, és ha frissülnek szól a Viewnak, ami mindent lefrissít a változtatásoknak megfelelően.

Unit

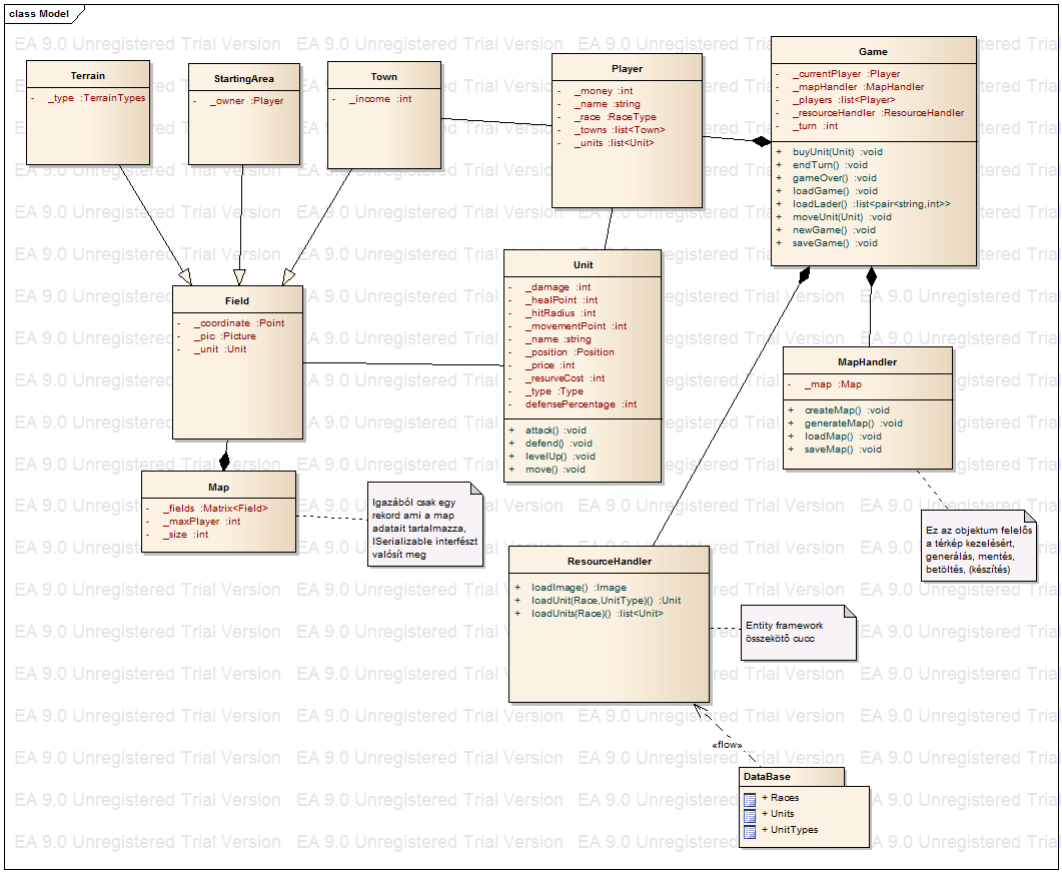
A Unit osztály írja le az egységeket. Az egység három különböző faj lehet, mindegyik három szintű lehet. A Unit osztályból mindig az összes létező egység számával megegyező példány létezik. Feladata egy adott egység tulajdonságainak tárolása.

A tulajdonságokat property-kben tároljuk, amiket getterrel és setterrel érünk el.

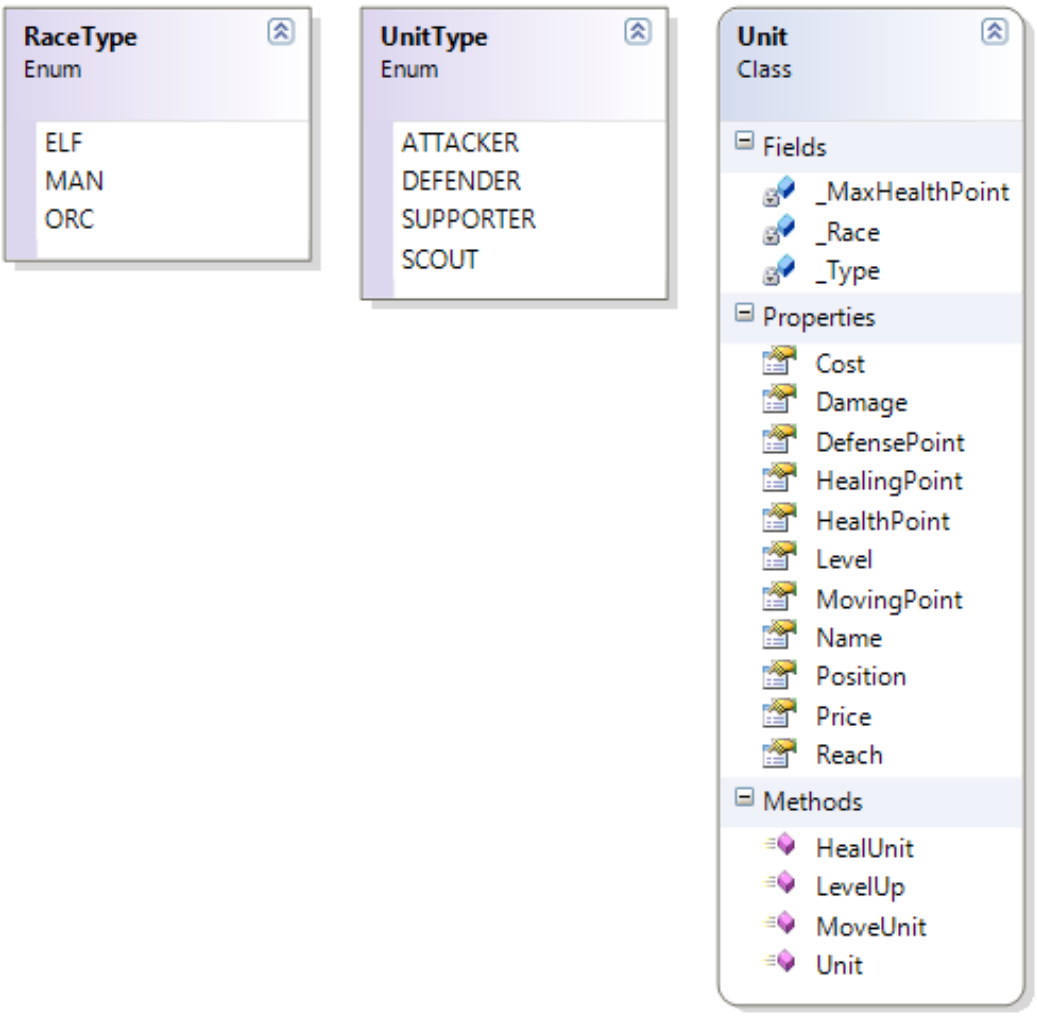
A fajokat egy felsoroló típus írja le. A faj három féle lehet: ELF/HUMAN/ORC.

Minden fajon belül négy típusú egységet hozhatunk létre. Ennek tárolására is egy felsorolási típust használunk, melynek elemei: ATTACKER/DEFENDER/SUPPORTER/SCOUT.

A konstruktor a fenti két paraméter segítségével hozza létre az egységeket a megfelelő tulajdonságokkal.



2.2. ábra. Model Osztálydiagram



2.3. ábra. Unit osztály

A *MoveUnit* metódus felelős az egység elmozdításáért. Paraméterül az új helyet és a mozgás költségét kapja. Ha a mozgás költsége nagyobb, mint a rendelkezésre álló mozgáspont, az egység helyben marad, egyébként elmozdul a megadott helyre. Visszatérési értéke igaz, ha a mozgás sikerült, egyébként hamis.

A *HealUnit* metódus "gyógyítja" az egységeket. Paraméterül egy egész számot kap, ennyivel kell növelni az egység életerejét. Visszatérési értéke az egység életereje.

A *LevelUp* függvény felelős az egységek szintlépéséért. Meghívásakor az egység megfelelő tulajdonságai megnöveli. Visszatérési értéke az egység új szintje.

Game

A Game osztályhoz fut be minden kérés, ő a Model lelke, ellenőrzi a kívánt műveletek (lépések stb.) érvényességét, frissíti a térképen lévő egységeket, mezőket, magát a térképet, amely változtatásokról a View értesül.

Képes a játék állásának minden összetevőjét közvetlenül vagy közvetetten elérni, bizonyos dolgokat önmaga is tárol a játékállásról.

Adattagjai:

- *currentPlayer*: A játékban soron lévő játékos. A jelenlegi játékállásban a következő lépés csak számára engedélyezett; ha más próbál lépni, az nem megengedett művelet.
- *mapHandler*: A Game ezen az osztályon keresztül lép kapcsolatba a térképpel, hiszen ő felelős a térkép kezeléséért: generálás, mentés, betöltés.
- *players*: A játékosok listája. A Game őket arra használja, hogy pl. az adott lépést képes-e a játékos a meglévő pénzével, fajával stb. megtenni. A Game rajta keresztül látja a Unitokat is.
- *resourceHandler*: A Game ezt az osztályt használja a bináris adatok, és az adatbázisban tárolt adatok betöltésére.
- *turn*: A jelenlegi kör száma.

Metódusai:

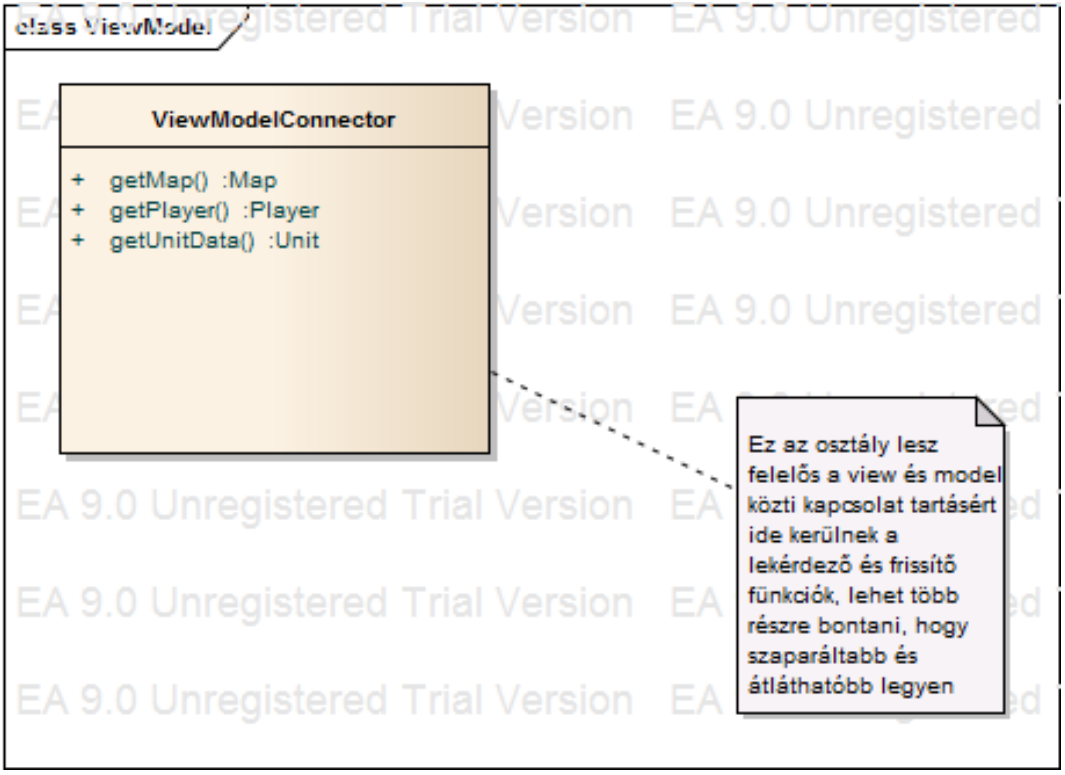
- *newGame()*: A játék elején hívódik meg, durván egy logikai konstruktornak is tekinthető. Létrehozza a játék elején szükséges objektumokat, beállítja a játékosok, egységek stb. alapértelmezett értékeit.
- *loadGame()*: Betölt egy korábban elmentett játékállást. Létrehozza a szükséges objektumokat, beállítja az értéküket, a MapHandler-rel legeneráltatja a térképet stb.
- *saveGame()*: Menti a jelenlegi játékállást úgy, hogy később betölthető legyen, ld. loadGame().
- *gameOver()*: A játék végén hívódik meg, durván egy logikai destruktornak is tekinthető. Elvégzi a játék végén fontos teendőket; pl. frissíti a Lader-t, felszabadítja az erőforrásokat stb.
- *endTurn()*: Minden kör végén hívódik meg. A teljesítmény alapján a játékosok pénzt kapnak, illetve az egységeik száma alapján zsoldot fizetnek.
- *moveUnit()*: Az egységek mozgatásához használt metódus. A mozgással lehet falvat foglalni, az ellenséggel harcolni, vagy egyszerűen másik mezőre lépni. Ezek mindegyike másféleképp befolyásolja az egységek tulajdonságait.
- *buyUnit()*: Egységek vásárlásakor hívott metódus. Létrehozza az új egységet, és lehelyezi a térkép speciális részére, a játékos kezdőterületére.

2.2.3. ViewModel

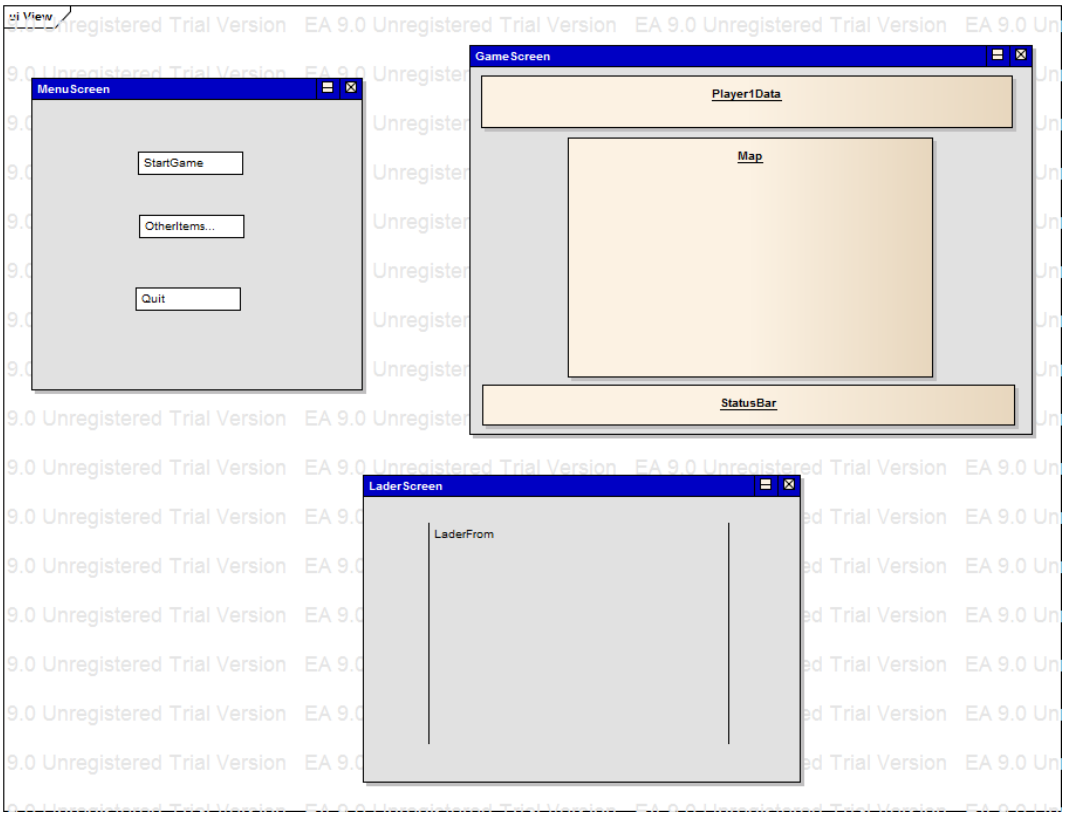
2.3. Dinamkus működés

2.4. Felhasználó-felület modell

2.5. Részletes programterv



2.4. ábra. ViewModel Osztálydiagram



2.5. ábra. Felület Tervek

3. fejezet

Implementáció

3.1. Fejlesztő eszközök

Az implementácó a Microsoft Visual Studió 2010-val történik, a szoftvert komponens alapú módon fejlesztjük, hogy ne legyenek kódolási átfedések mindenkinek csak a saját komponensébe kell kódolnia, valamint a tesztelés és karbantartás/további fejlesztések könnyen menjenek. A kód vázát az Enterprise Architect program segítségével generáltattuk le az előzetes tervekből, melyeket később kiegészítünk vagy módosítunk ha szükséges. A programot a WPF technológia segítségével valósítjuk meg MVVM(Model-View-ViewModel) architektúrával.

3.2. Forráskód,futtatható kód

3.2.1. Vew-ViewModel komponens

A view a következő fájlokból áll:

- App.xaml, App.xaml.cs: A fő program, ez az osztály vezárli a GUI szálakat és tölti be az ablakot.
- MainWindow.xaml, MainWindow.xaml.cs : A fő ablak kódja, ezen az ablakon cserálgetjük a különböző kontrolokat a megjelenítés szerint.
- MainMenUserControl.caml, MainMenuUserControl.xaml.cs: A főmenü megjelenítése és eseményeinek kezelése.

- LadderUserControl.xaml, LadderUserControl.xaml.cs: A ranglétra megjelenítése.
- GameUserControl.xaml, GameUserControl.xaml.cs: A játék felület, amiben majd játszhatunk.
- GameStartOptionsuserControl.xaml,GameStartOptionsuserControl.xaml.cs: A játék előtti kezdeti beállítások, játékos neve faja.
- pic/menubackground.png : A menü valamint az játék előtti ablakok hátter.

A viewModel a következő fájlokból áll:

- ViewModelBase.cs: A ViewModel alap funkcióit szokás egy ősosztályba kiemelni.
- StartegyGameViewModel.cs: A konkrét ViewModel, a base-ből származik itt vannak megvalósítva a program specifikus függvényei.
- test/ModelStab.cs: A view tesztelésére szolgáló stub osztály.

3.3. Alkalmazott kódolási szabályok

Msdn microsoft C# Conventions: http://msdn.microsoft.com/en-us/library/ff926074.aspx. Az osztályok nevei a funkciójuknak megfelelő előtaggal és a típusnak megfelelő utótaggal rendelkeznek.

4. fejezet

Tesztelés

Mivel a szoftvert komponens alapon fejlesztettük ezért könnyű tesztelni. A tesztelés alapvetően fekete doboz módon történik, Mockolás segítségével. A komponensek a velük kapcsolatban álló osztályokat stubok és driverek segítségével helyettesítik, hogy a saját függvényeik működését és helyes meghívását ellenőrizzék.

4.1. View-ViewModel

A View-t természetéből kifolyóan nehéz tesztelni hagyományos fehér doboz módszerekkel, inkább a fekete doboz tesztelés a megszokott. A view-t megfelelő kattintás felvevő és lejátszó teszteszköz hiányában, csak kézzel teszteltük, minden gomb végigkattintásával. A View-Model kapcsolatot azonban már könyebb kezelni, a model helyettesíthető egy megfelelő stubbal, ami kírja hogy a megfelelő funkció meghívódott esetleg buta módon fix értékekkel tér visssza, így a közreműködés a két komponens között tesztelhető.

5. fejezet

Felhasználói dokumentáció