Eötvös Loránd Tudományegyetem

Informatikai Kar

[Programozáselmélet és Szoftvertechnológiai](http://www.inf.elte.hu/karunkrol/szervezet/tanszekek/Lapok/Inform%C3%A1ci%C3%B3sRendszerekTansz%C3%A9k.aspx) Tanszék

**Életszimulátor játék**

*Témavezető:*

Várkonyi Teréz Anna

egyetemi adjunktus, Ph.D.

*Szerző:*

Dóra László

Programtervező informatikus BSc.

*Budapest, 2021*

Tartalomjegyzék

[Bevezetés 3](#_Toc71815523)

[Alkalmazott technológia 3](#_Toc71815524)

[Fejlesztői környezet 4](#_Toc71815525)

[Felhasználói dokumentáció 5](#_Toc71815526)

[Rendszerkövetelmény 5](#_Toc71815527)

[Rövid leírás 5](#_Toc71815528)

[Használati útmutató 6](#_Toc71815529)

[Fejlesztői dokumentáció 23](#_Toc71815530)

[Tervezés/Követelményelemzés 23](#_Toc71815531)

[Feladat leírása 23](#_Toc71815532)

[Use-case diagram 24](#_Toc71815533)

[Megjelenítés terve 24](#_Toc71815534)

[Egyszerűsített osztálydiagram 29](#_Toc71815535)

[Nem funkcionális követelmények 29](#_Toc71815536)

[Megvalósítás 31](#_Toc71815537)

[A szoftver felépítése 31](#_Toc71815538)

[Implementáció során hozott döntések 32](#_Toc71815539)

[Modell 32](#_Toc71815540)

[Nézet 38](#_Toc71815541)

[Perzisztencia 42](#_Toc71815542)

[Felhasználói történetek/tesztelés 45](#_Toc71815543)

[Összefoglaló 56](#_Toc71815544)

[Továbbfejlesztési lehetőségek 56](#_Toc71815545)

[Köszönetnyilvánítás 56](#_Toc71815546)

[Irodalomjegyzék 57](#_Toc71815547)

# Bevezetés

Szakdolgozatom témája – mint a címből is kiderül - egy életszimulátor játék az Android és iOS operációs rendszerekre megjelent BitLife nevű játék mintájára. Három fő motivációm volt a program elkészítésében: egy időben sokat játszottam az említett játékkal, amikor unatkoztam, és többször is elgondolkodtam rajta, hogy mennyi munkába kerülhet egy ehhez hasonló játékot elkészíteni.

A másik motivációm az volt, hogy Windows operációs rendszereken is elérhető legyen egy ehhez hasonló játék. Bár a Real Lives 2007 megfelelőnek tűnhet erre a célra, de az már eléggé elavult és beszerezni sem egyszerű.

Végső motivációm az eseményvezérelt alkalmazások témaköre volt. Harmadik félévemben végeztem el az ehhez tartozó tárgyat, és bár nem volt egyszerű teljesíteni, rengeteget dolgoztam a beadandókon és a ZH-kra való készülés is rengeteg időmbe telt, de végül nagyon megkedveltem a témát, és már akkor eldöntöttem, hogy szakdolgozatom témáját is egy ilyen alkalmazás fogja adni.

A fenti okok miatt döntöttem úgy, hogy ebből írom a szakdolgozatomat, és majd meglátjuk, hogy meddig jutok. A témabejelentőben kitűzött feladatokat sikerült teljesítenem, viszont úgy érzem, hogy még bőven van hova fejlődnie a programomnak (erről majd az összefoglaló részben írok bővebben).

## Alkalmazott technológia

Az alkalmazás elkészítéséhez a Windows Forms [1] grafikus könyvárat használtam. Ez egy ingyenes, nyílt forráskódú könyvtár, amely a Microsoft .NET Framework [2] része. A fejlesztő számára könnyű hozzáférést biztosít a Windows operációs rendszer grafikus felületét képző control-okhoz, mint pl. Button, ListBox, TextBox, ComboBox.

Előnyei közé tartozik az egyszerű kezelhetőség (drag-and-drop design), a részletes dokumentáció, valamint a kis erőforrásigény. Hátránya, hogy az alap könyvtár nem biztosít túl sok testreszabási lehetőséget az alkalmazásunk számára, így pl. a WPF-nél sokkal nehezebb olyan külsejű alkalmazást megalkotnunk, amilyet szeretnénk. Ennek az orvoslása lehetséges a piactérről letölthető különböző control-okkal, keretrendszerekkel, viszont ezeknek a nagy részéért sajnos fizetni kell, nem is keveset.

## Fejlesztői környezet

Az alkalmazás fejlesztéséhez a Microsoft Visual Studio-t [3] használtam, azon belül a 2019-es verziót. Személy szerint nekem ez az egyik kedvenc fejlesztőkörnyezetem a következők miatt:

* Rengeteg nyelvet és keretrendszert támogat, szinte bármilyen programot, rengetegféleképpen megvalósítva meg lehet benne írni a nulláról.
* Nagy mértékben testreszabható kezelőfelület, mindenki megtalálhatja a magának megfelelő beállításokat.
* GitHub-integráció: könnyedén tudunk foglalkozni az alkalmazásunk verziókezelésével anélkül, hogy külön programot kéne hozzá letölteni.
* A hibás kód javítására javaslatokat ad, amelyek nagyon hasznosak tudnak lenni, az egyszerűbb problémákra, elírásokra teljesen jó megoldást szokott adni.
* A debugger az egyik legjobb, amit valaha használtam. Könnyen kezelhető, gyors és minden információt kiír, amire szükség lehet. Ezen kívül be lehet állítani, hogy csak akkor álljon meg a break point-on, ha bizonyos feltételek teljesülnek.

Természetesen van egy pár apróbb hátránya is:

* Könnyen el lehet veszni a beállításokban, viszont van egy keresőfelület, amivel ezeket könnyedén lehet keresni
* Rengeteg helyet foglal, főleg ha több nyelven, keretrendszerben fejlesztünk.
* Gyengébb gépeken rendkívül lassan tölt be.
* Néha hajlamos lefagyni (de könnyen vissza tudjuk állítani, amin dolgoztunk), ezen kívül egy pár kisebb bug-gal is találkoztam a fejlesztés során, de ez annyira nem meglepő egy ilyen alkalmazásnál.

# Felhasználói dokumentáció

## Rendszerkövetelmény

* Dual Core 1GHz-es, vagy gyorsabb processzor
* minimum 1 GB RAM
* Szoftver:
  + Visual Studio 2019 a buildeléshez és futtatáshoz
  + .NET Core 3.1

## Rövid leírás

A program egy életszimulátor játék. Egy véletlenszerű, vagy általunk generált ember életét vezethetjük végig benne. Ennek az embernek vannak különböző statisztikái, (egészség, kinézet, intelligencia, boldogság) amelyek a játék különböző aspektusaiban játszanak szerepet. Ezek a statisztikák évről évre változnak minimális mértékben, de különböző tevékenységekkel lehet növelni őket. Ha az egészség statisztika 0-ra csökken, akkor a karakter meghal.

A játékban lehet munkát vállalni, amely minden évben növeli a játékos karakter pénzét egy bizonyos összeggel (ez növelhető előléptetésekkel). A jobb munkákhoz egyetemi képzettség is kell. A pénzből lehet venni különböző dolgokat. (pl. lakást, autót, nyaralást) amelyek a különböző statisztikákat növelik.

Ezen kívül fontos a karakter ismerőseivel tartani a jó kapcsolatot, mivel ez plusz boldogságot jelent, ami az egészségre is kihatással van.

Lehet romantikus kapcsolatot is kialakítani ellenkező nemű karakterekkel, akikkel gyermeket is lehet vállalni. Ekkor halál esetén a legidősebb gyermek felett vehetjük át az irányítást.

Konkrét cél nincsen a játékban, viszont található benne egy ún. achievement-rendszer, ami különböző mérföldkövek teljesítéséért jutalmazza a játékost. Ez biztosítja azt, hogy azoknak is legyen valami kihívás/élvezeti faktor a játékban, akiknek mindenképpen kell valami elérendő cél, hogy élvezzenek egy játékot.

## Használati útmutató

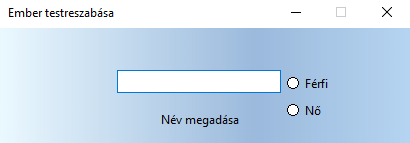
A játék elindításakor egy ablak jelenik meg, ahol kiválaszthatjuk, hogy egy véletlenszerűen generált emberrel, vagy egy testreszabható emberrel kezdünk neki a játéknak.

**A képen szöveg látható

Automatikusan generált leírás**

*1. ábra: Kezdőképernyő*

Ha az előbbi opciót választjuk, akkor a játék véletlenszerűen generál nekünk vezetéknevet, keresztnevet, valamint nemet és egyből a játékban találjuk magunkat, viszont az utóbbi opció esetén testreszabhatjuk karakterünk nevét és nemét.

****

*2. ábra: Ember testreszabása*

A szövegdobozba íráskor oda kell figyelni arra, hogy 2 szót adjunk meg (vezeték- és keresztnév), valamint arra, hogy mindkét megadott szó tartalmazzon karaktert a szóközön kívül. Ha ezen feltételek valamelyike nem teljesül, a következő hibaüzenetekkel találkozhatunk:

**A képen szöveg látható

Automatikusan generált leírás**

*3. ábra: Túl sok szó*

**A képen szöveg látható

Automatikusan generált leírás**

*4. ábra: Vezeték- vagy keresztnév hiányos*

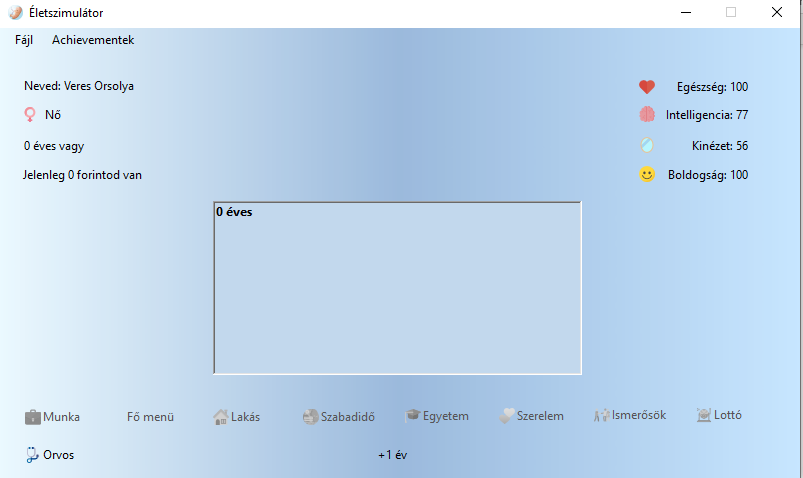
Miután mindezzel végeztünk, a játék elindul, és egy felugró ablak (lásd 5. ábra) tudatja a játékossal szüleihez és a karakteréhez tartozó adatokat.

A képen szöveg látható

Automatikusan generált leírás

*5. ábra: Statisztikák*

Ha mindezt tudomásul vettük, akkor elkezdődhet a játék, megjelenik a fő ablak. Itt a felső menüsorban található egy ’Fájl’ menü, ahol a mentést és a betöltést tudjuk elvégezni, valamint egy ’Achievementek’ menü, ahol a játék során elért teljesítményeinket tudjuk figyelemmel kísérni. Ezen kívül itt találhatóak a különböző panelekre vezető gombok (munka, lakás, egyetem, szerelem stb.), a lottó gomb, az orvos gomb, valamint az a gomb, amivel tudjuk léptetni az éveket. Amikor a játék elkezdődik, akkor a fő panelt láthatjuk, ahol a különböző statisztikáink, valamint az eddig történt események vannak nyilvántartva.

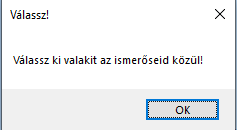


*6. ábra: A játék fő ablaka, valamint a fő panel statisztikáinkkal*

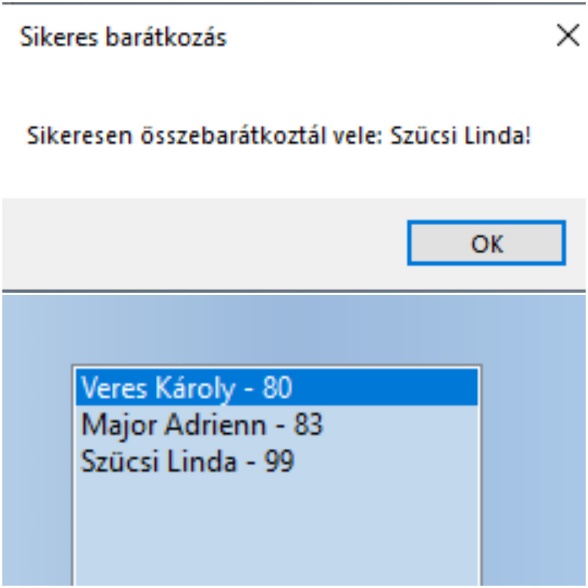
Ahogy az ábrán is látszik, eleinte nincsen sok lehetőségünk, de ahogy egyre idősebbek leszünk, úgy fog megnyílni egyre több és több lehetőség. Ezek közül az első 3 évesen az ’Ismerősök’ menüje lesz, egy listába rendezve láthatjuk az összes ismerősünket, akikkel tudunk közös programot szervezni, ezzel növelve a kapcsolatpontunkat az adott személlyel (ehhez ki is kell választani valakit a listáról, különben nem fog sikerülni, lásd 8. ábra), valamint megpróbálni barátkozni valakivel, de ez nem mindig jön össze.



*7. ábra: Ismerősök menü*



*8. ábra: Ez történik, ha nem választunk ismerőst közös programra*



*9. ábra: Sikeres barátkozás*

Az ismerősrendszerhez még hozzátartozik az is, hogy évente van rá esély, hogy összevesszünk ismerőseinkkel. Ekkor természetesen csökken a velük való kapcsolat. Érdemes a kapcsolatainkat minél jobban ápolni, mert ez hozzájárul a boldogságértékünk növekedéséhez vagy csökkenéséhez.

A következő menüpont 12 éves korunkban nyílik meg, és ez nem más, mint a ’Szabadidő’ menü. Itt tudunk olyan tevékenységeket végezni, mint pl. edzés (egészség, boldogság, kinézet növelése), olvasás (intelligencia, boldogság növelése), valamint vakáció (boldogság növelése nagy mértékben, ezt csak 18 éves kortól). Mindegyiket évente egyszer tudjuk csinálni és 18 éves kor felett pénzbe kerülnek. Ha nincs meg a szükséges pénzmennyiség, akkor természetesen nem tudjuk az adott szabadidős tevékenységet űzni.

A képen szöveg látható

Automatikusan generált leírás

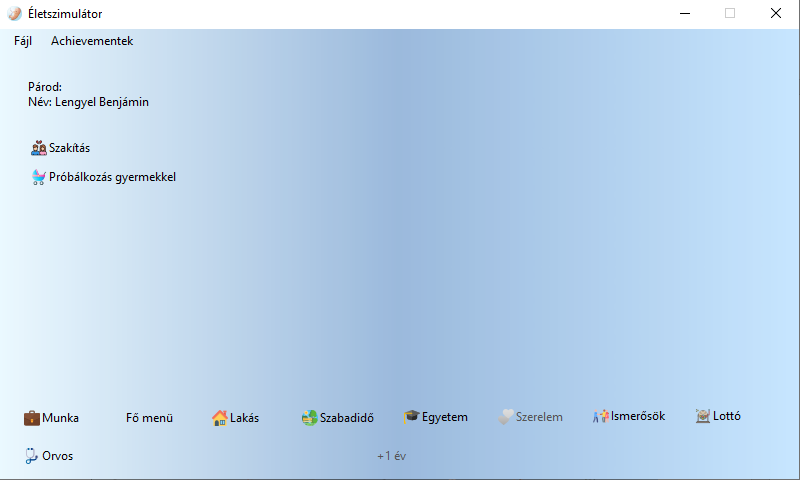
*10. ábra: Szabadidő menü*

Ez után 14 éves korunkban nyílik meg a következő menü, ez pedig a szerelem menü. Itt először csak partner keresésére lesz lehetőségünk. Ha ezt tesszük, akkor felkínál nekünk a játék egy potenciális párt, akinek így megtudjuk a nevét, korát, kinézetét, intelligenciáját, valamint, hogy mennyi esélyünk van nála. Ez a mi kinézet és intelligencia statisztikánktól is függ. Minél jobb a partnerhez képest, annál nagyobb az esély.



*11. ábra: Szerelem menü potenciális partnerrel*

Ha a ’Kapcsolat megpróbálása’ gombra nyomunk, akkor véletlenszerűen vagy összejövünk a potenciális partnerrel, vagy nem. (ha 100 az esély, akkor mindenképp, ha 0, akkor semmiképp). Amennyiben sikeres a kapcsolat, abban az esetben frissül a menü, és láthatjuk, hogy ki az új partnerünk. Ezen kívül megnyílik a szakítás és a gyermekvállalás lehetősége is (ez csak 18 éves kortól).



*12. ábra: Szerelem menü az új lehetőségekkel*

Szakítás esetén kapunk egy figyelmeztető ablakot, aminél ha igent mondunk, akkor megtörténik a szakítás, és a szerelem menü visszaáll a 11. ábrán látható alap állapotba. Ha gyermekkel próbálkozunk és sikerrel járunk, akkor várnunk kell a következő évig mire megszületik, ekkor kapunk róla egy értesítést (13. ábra) és bekerül az ismerőseink listájába.

A képen szöveg látható

Automatikusan generált leírás

*13. ábra: Gyermek születése*

A képen szöveg látható

Automatikusan generált leírás

*14. ábra: Gyermek bekerült az ismerőslistába*

18 éves korunktól már minden funkció elérhető, ezek közül az egyik az ’Egyetem’ menüpont. Itt ki tudunk választani egy egyetemet, amit el tudunk végezni. Intelligenciaszintünktől függően bekerülhetünk államilag finanszírozott, valamint önköltséges képzésre is. Az önköltséges képzés költségeit az egyetem elvégzése után kell elkezdeni visszafizetni. Miután elvégeztük a képzést, (ez 3-6 évig tarthat szaktól függően) kapunk róla egy értesítést. (16. ábra)

A képen szöveg látható

Automatikusan generált leírás

*15. ábra: Egyetem menü*

*A képen szöveg látható

Automatikusan generált leírás*

*16. ábra: Egyetemi képzés elvégezve*

A következő elérhető menüpont a ’Munka’ menü. Itt tudunk munkát vállalni, valamint már meglévő munkából felmondani. Ha egy munkához olyan képzettség szükséges, amivel mi nem rendelkezünk, akkor azt nem tudjuk elvállalni. (lásd 18. ábra)

A képen szöveg látható

Automatikusan generált leírás

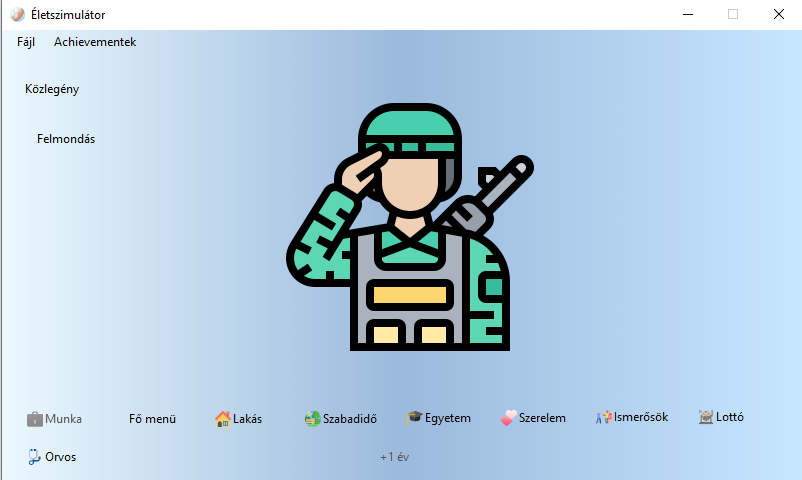
*17. ábra: Munka menü*

*A képen szöveg látható

Automatikusan generált leírás*

*18. ábra: Egyetemi képzés hiánya*

Ha sikeresen megkaptuk az állást, akkor ezt egy kép fogja jelezni, ami az állásunkat jelképezi.



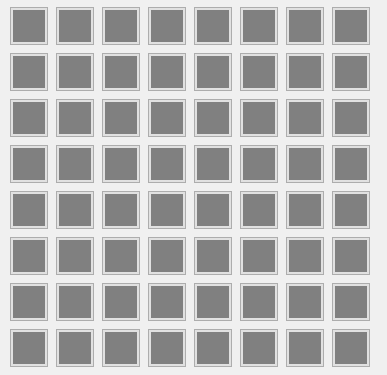
*19. ábra: Munka menü már meglévő munkával*

A munkához tartozik, hogy minden évben kapunk érte egy fix fizetést, ami az előléptetésekkel fokozatosan növekszik. Ezen kívül a katonai karrier során kis eséllyel behívhatnak minket külföldi missziókra, amelyek során egy aknakereső játékot kell lejátszani. Ekkor a 20. ábrán látható üzenetet láthatjuk, ami után megjelenik az aknakereső ablaka. Ekkor a fő játék ablakán minden gomb elérhetetlenné válik, amíg a missziót nem teljesítjük.

A képen szöveg látható

Automatikusan generált leírás

*20. ábra: Bevetésre hívtak*



*21. ábra: Aknakereső ablaka betöltéskor*

Ez egy teljesen átlagos aknakereső játék. Egy 8x8-as mezőn kell 15 véletlenszerűen elhelyezett aknát megtalálni. A bal egérgombbal tudjuk felfedni a kívánt mezőt és a környékét, a jobb egérgombbal pedig meg tudjuk jelölni a gyanús mezőket, ekkor egy zászló kerül a mezőre. A felfedett mezőkön lévő szám jelzi, hogy hány aknamező van közvetlenül mellette. Ha a bal egérgombbal olyan mezőre kattintunk, ahol akna van, akkor vége a játéknak, és a fő játékon belül egészséget vesztünk, akár meg is halhatunk. Ha viszont sikeresen azonosítjuk az összes aknát, akkor a missziónk sikeres, közelebb kerülünk az előléptetéshez, valamint egy kis bónusz pénzösszeget is kapunk.

A képen elektronika látható

Automatikusan generált leírás

*22. ábra: Folyamatban lévő aknakereső játék*

A következő elérhető menüpont a ’Lakás’ menü. Itt ha van elég pénzünk, akkor el tudunk költözni a szülői lakásból, ami extra boldogságot jelent minden évben. Érdemes viszont odafigyelni arra, hogy a drágább házaknak többe kerül az éves fenntartása, így ha még nem elég magas a fizetésünk, érdemes olcsóbb lakással kezdeni, az is jobb, mint a semmi. A meglévő lakásunkat el is tudjuk adni, de ekkor nem a teljes vételárat kapjuk vissza, csak annak egy részét.

A képen szöveg látható

Automatikusan generált leírás

*23. ábra: Lakás menü*

A képen szöveg látható

Automatikusan generált leírás

*24. ábra: Lakás menü megvásárolt lakással*

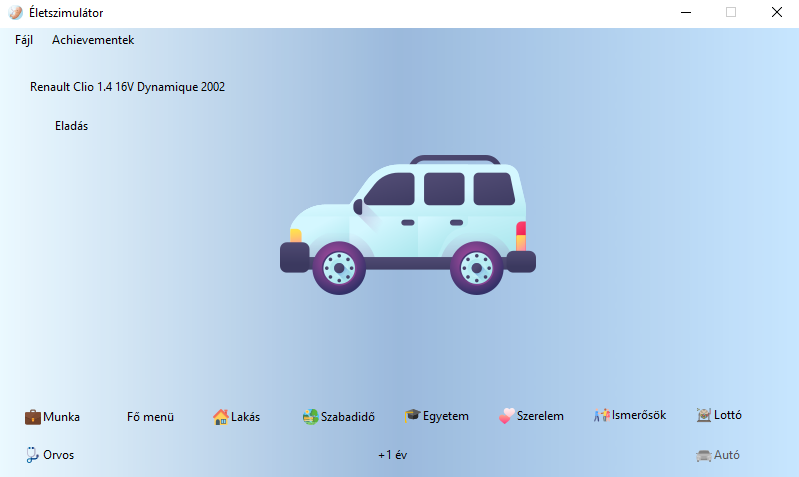
Az utolsó panellel rendelkező menü az ’Autó’ menü. Itt 18 éves korunktól le tudjuk tenni a jogosítványt, ami egy random KRESZ kérdés megválaszolását jelenti és 250 000 forintunkba kerül.

A képen szöveg látható

Automatikusan generált leírás

*25. ábra: Jogosítvány vizsga*

Ha sikeresen megválaszoltuk a kérdést, akkor onnantól kezdve rendelkezünk jogosítvánnyal, így tudunk autót venni. Minél jobb autónk van, annál pozitívabb hatással van a boldogságpontjainkra. Mint a lakást, az autót is el lehet adni, természetesen kicsivel kevesebb pénzért, mint amennyiért vettük.



*26. ábra: Autó menü*

Aki szerencsét akar próbálni, az kipróbálhatja a lottózás lehetőségét. 5000 forint egy lottószelvény, a nyeremény pedig változó mértékű 2 millió és 250 millió forint között.

A játék részét képzik a különböző véletlenszerűen bekövetkező események is. Ilyen pl. a betegség elkapása. Ez lehet egy kisebb megfázás, de lehet akár valami komolyabb betegség is, amit csak orvos tud diagnosztizálni és gyógyítani. Ez pénzbe kerül, és nem mindig sikerül a kezelés.

*A képen szöveg látható

Automatikusan generált leírás*

*27. ábra: Betegség diagnosztizálása, sikertelen kezelés*

Ezen kívül, ha repülőgéppel utazunk, (nyaralás, pilóta karrier) akkor megvan az esélye annak, hogy repülőgépünk balesetet szenved, és meghalunk, bár ennek az esélye igen csekély.

*A képen szöveg látható

Automatikusan generált leírás*

*28. ábra: Repülőgép-szerencsétlenség*

A játék elmentésére és betöltésére is van lehetőségünk a ’Fájl’ menüpont alatt. Mindkét esetben ki kell választanunk, hogy hova/honnan szeretnénk menteni/betölteni a mentésfájlt. Ha valamilyen oknál fogva a mentés/betöltés nem végezhető el, (pl. hibás a betöltendő fájl) akkor hibaüzenetet kapunk. Emiatt nem igazán ajánlott a meglévő mentési fájlt saját kezűleg szerkeszteni, mert ez könnyen hibát okozhat.

*A képen asztal látható

Automatikusan generált leírás*

*29. ábra: Mentés/betöltés felülete*

*A képen szöveg látható

Automatikusan generált leírás*

*30. ábra: Betöltési hiba*

Végül, de nem utolsó sorban a játék rendelkezik egy achievement-rendszerrel is. Ez azt jelenti, hogy léteznek különböző mérföldkövek - mint például nyugdíjba vonulás katonai karrierből - amiknek a teljesítése után a játékos egy felvillanó üzenet formájában értesítést kap. Ezek nem vesznek el, minden indításkor megmaradnak, a játék gyökérkönyvtárában vannak eltárolva, az achievements.ach fájlban. Ez egy jó motiváció lehet azoknak a játékosoknak, akik szeretnek valamilyen célért küzdeni. A meglévő achievement-jeinket játék közben az ’Achievementek’ menüpontra kattintással lehet megtekinteni. A zöld betűvel írt achievement-eket már megszereztük, a piros betűseket pedig még nem.

A képen szöveg látható

Automatikusan generált leírás

*31. ábra: Achievement menüpont*

# Fejlesztői dokumentáció

## Tervezés/Követelményelemzés

### Feladat leírása

A játék egy szöveg alapú életszimulátor, amely hasonlóan működik, mint az Android és iOS operációs rendszerekre megjelent Bitlife. Egy véletlenszerűen generált, vagy egy játékos által testreszabható (név, nem) karakterrel láthatunk neki a játéknak. A játéknak konkrét célja nincs, de a játékos tűzhet ki magának célokat, (minél tovább éljen a karakter, minél több pénze legyen) és egy achievement-rendszer is biztosítja, hogy legyen valami cél, amiért küzdhet a játékos.

A játékot egy gomb megnyomásával vihetjük előre. A játékosnak négy fő tulajdonsága van: egészség, kinézet, intelligencia és boldogság. Ezek a statisztikák minden előrehaladással változnak minimális mértékben, de különböző tevékenységekkel magunk is növelhetjük ezeket. Az egészségre és a boldogságra különösen oda kell figyelni, mivel ha az egészség leesik 0-ra, akkor meghaltunk, vége a játéknak, a boldogság pedig befolyásolja az egészséget, így érdemes ezt magasan tartani. Ezt úgy lehet elérni, hogy jó kapcsolatot ápolunk ismerőseinkkel, vagy szabadidős tevékenységeket végzünk.

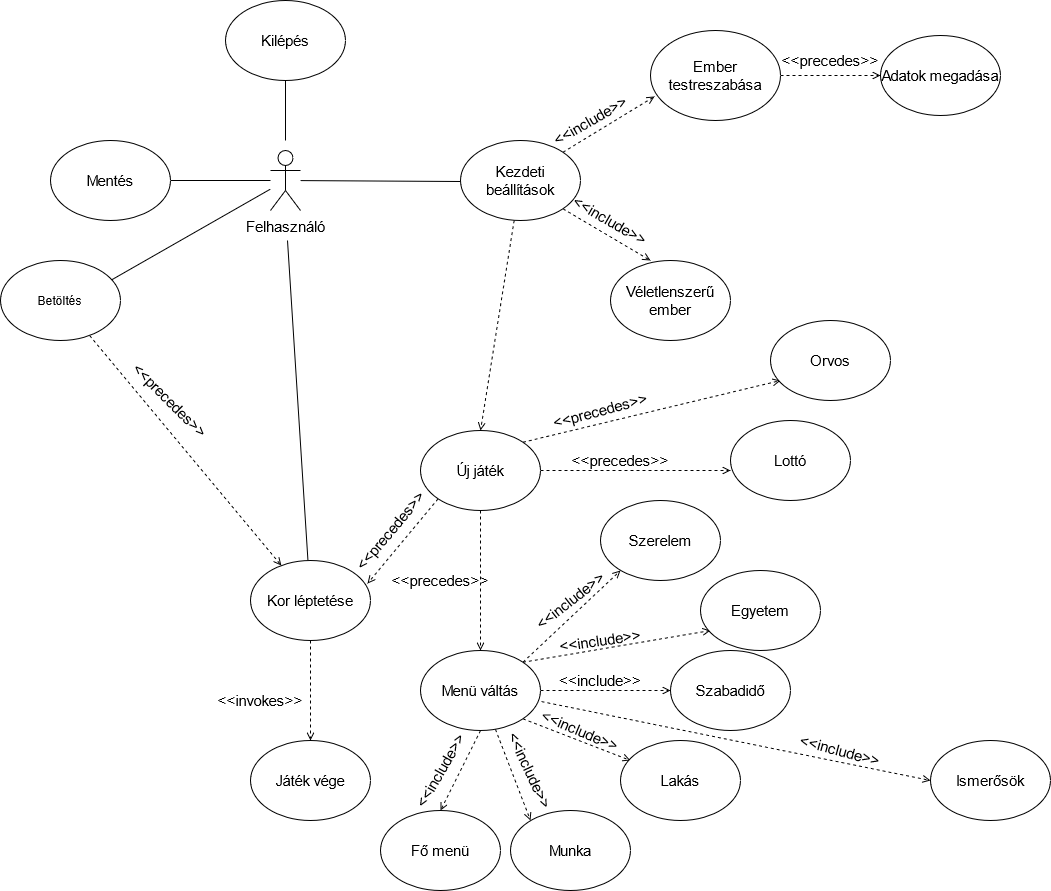
A játékos karakter 18 éves korától munkát vállalhat, amivel pénzt kereshet. A pénzzel tudjuk végezni a szabadidős tevékenységeket, valamint ez kell a házak, autók megvételéhez (ehhez jogosítvány is kell, ami szintén pénzbe kerül, valamint egy teszt is kell hozzá) és a lottózáshoz is. Vannak munkák, amelyeket csak bizonyos egyetemi képzettséggel lehet végezni. Az egyetem elvégzése is pénzbe kerülhet, ha a játékos karakter nem elég intelligens.

A játékos karakter összejöhet egy ellenkező nemű NPC-vel, valamint gyermeket is vállalhat vele. Ezzel extra ismerősökre tehet szert, valamint ha karakterünk meghal, akkor a legidősebb gyermeke felett átvehetjük az irányítást.

Ezen kívül történhetnek különböző véletlenszerű események: repülőgép-szerencsétlenség (pilóta karrier, vakáció), autóbaleset (ha a játékos rendelkezik kocsival), behívhatnak bevetésre (katonai karrier, ez egy aknakereső minijátékot indít el), összeveszés ismerőssel, valamint betegség elkapása. Az utóbbit az orvos tudja gyógyítani, de ez nem minden esetben sikeres.

A játékot el lehet menteni, valamint be is lehet tölteni bármikor, amikor a játékos szeretné. Ha ez valamilyen oknál fogva nem sikerülne, akkor a program nem száll el, hanem hibaüzenettel figyelmezteti a játékost, és a már folyamatban lévő játék megy tovább, mintha mi sem történt volna.

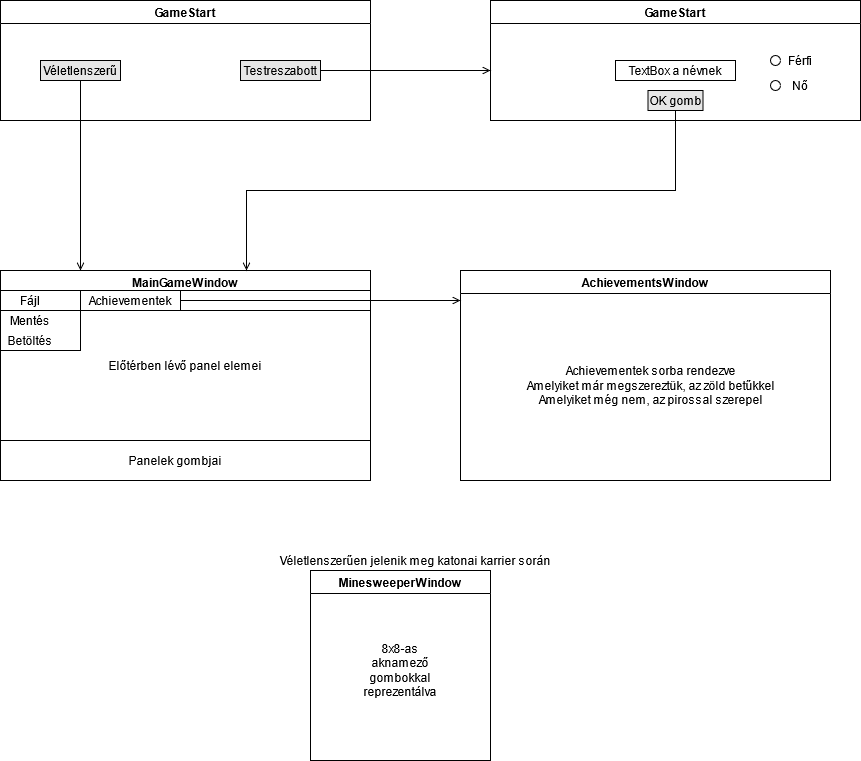
### Use-case diagram



*32. ábra: Főbb felhasználási esetek diagramja*

### Megjelenítés terve

A program több különböző ablakból áll, és az ablakokon belül is több panel váltakozik attól függően, hogy éppen melyik menüben van a játékos.



*33. ábra: Megjelenítés terve egyszerűsítve*

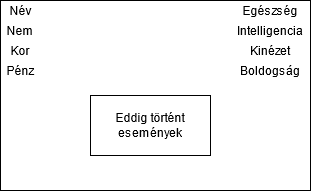
A **GameStart** ablakban 2 lehetőségünk van. A *Véletlenszerű* gombot megnyomva bezáródik ez az ablak és előjön a **MainGameWindow** ablak. Ha a *Testreszabott* gombot nyomja meg a játékos, akkor az ablak egy másik panelre vált, ahol meg tudja adni a karaktere nevét és nemét. Ezután az *OK gomb*ra nyomva jut a **MainGameWindow** ablakra.

A **MainGameWindow** ablakban a felső menüsorban láthatunk egy *Fájl* menüpontot, amelyben a *Mentés* és a *Betöltés* opciók szerepelnek. Az *Achievementek* menüpontra kattintva előjön egy új ablak (**AchievementsWindow**), amelyen az elért achievement-jeinket követhetjük nyomon. A fő ablakon továbbá láthatóak még az aktuális panelhez tartozó információk, valamint a panelekhez tartozó gombok is.

A **MinesweeperWindow** akkor jöhet elő véletlenszerűen, ha a játékos katonai karrierben dolgozik, ezen 8x8 darab gomb található, amelyek a mezőket reprezentálják.

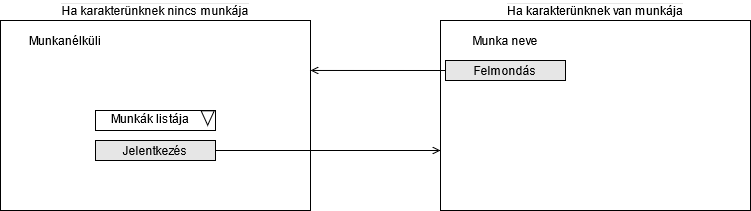
A panelek a következők:

**Fő panel:**



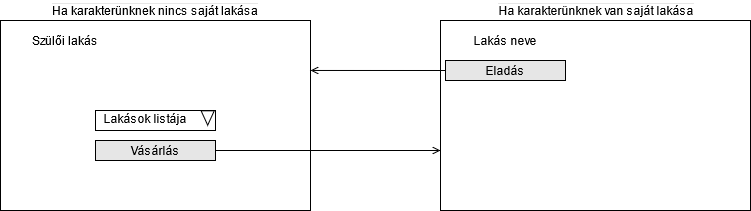
*34. ábra: Fő panel terve*

**Munka panel:**



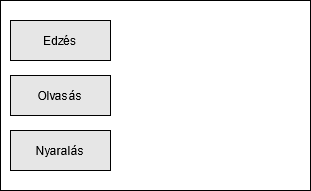
*35. ábra: Munka panel terve*

**Lakás panel:**



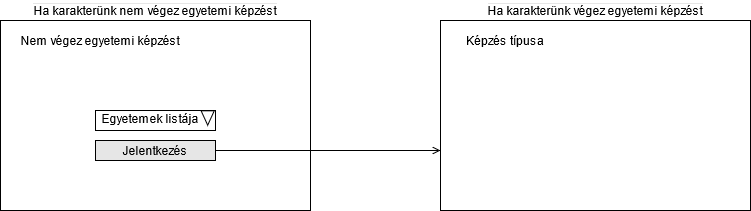
*36. ábra: Lakás panel terve*

**Szabadidő panel:**



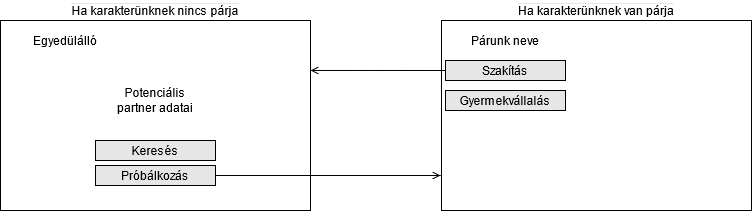
*37. ábra: Szabadidő panel terve*

**Egyetem panel:**



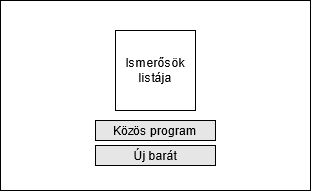
*38. ábra: Egyetem panel terve*

**Szerelem panel:**



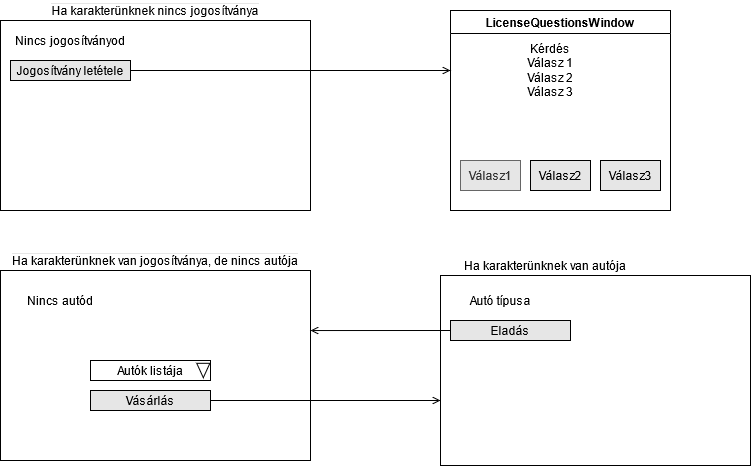
*39. ábra: Szerelem panel terve*

**Ismerősök panel:**



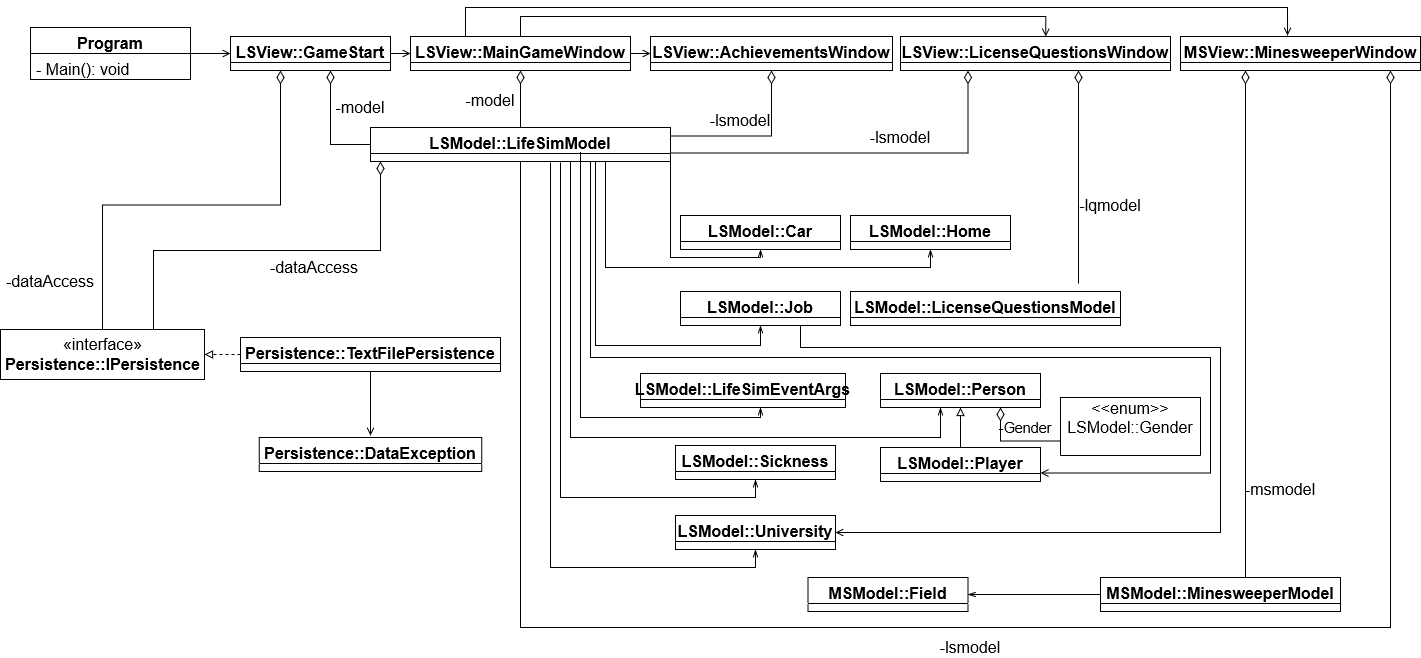
*40. ábra: Ismerősök panel terve*

**Autó panel és LicenseQuestionsWindow:**



*41. ábra: Autó panel és LicenseQuestionsWindow terve*

### Egyszerűsített osztálydiagram



*42. ábra: Osztálydiagram az adattagok és függvények nélkül*

### Nem funkcionális követelmények

Mivel maga a program nem túlságosan bonyolult, ezért mindenképpen pozitív dolog, ha az alkalmazás szépen néz ki. Ezt valamennyire behatárolja a WinForms, de ebből is lehet szép alkalmazást összehozni, ha úgy akarja az ember. Olyanokra kell gondolni, mint pl. legyen lehetőleg közel minden gombnak valamilyen hozzá illő ikonja, a statisztikák ikonja változzon a statisztika mértékétől függően stb.

Ezen kívül érdemes törekedni arra, hogy maga a programkód legyen szép, rendezett, az átláthatóság érdekében lehetőleg ne legyen benne túl sok ismétlés. Ha valamit többször végre kell hajtani, arra érdemes egy függvényt írni, majd azt a függvényt a megfelelő helyeken meghívni.

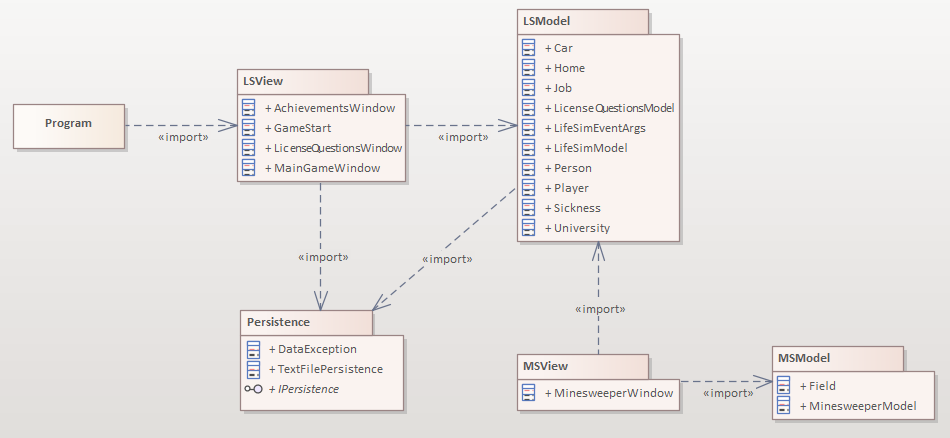
## Megvalósítás

### A szoftver felépítése

A szoftver 3 fő rétegből áll:

* ***Modell:*** A játéklogikát foglalja magában. Ebbe beletartoznak a játékos karakterének és a többi karakternek a statisztikái, a játék különböző elemeihez tartozó osztályok (pl. Person, Home, Job), a játék léptetéséhez szükséges függvények, metódusok, valamint a különböző események, amelyek jelzik a nézet felé, ha valamilyen változás történt a modellben.
* ***Nézet:*** Ez a réteg a játék megjelenéséért felelős. Ebben vannak definiálva a különböző Controlok (Button, Label, ComboBox, ListBox), ezen Controlok működése, valamint azok a függvények, amelyek valamilyen esemény végrehajtását követően változást eszközölnek valamelyik Controlban (pl. statisztikák változása, ismerőslista kiegészítése).
* ***Perzisztencia:*** Ha esetleg a játékosnak fontos dolga akad, vagy már nincs kedve játszani, akkor valószínűleg szeretné elmenteni azt, amit eddig elért. Erre szolgál a Perzisztencia réteg, amelyben a játékállás fájlba mentésére, valamint fájlból betöltésére szolgáló függvényeket valósítja meg. Ezen kívül az achievementek mentésére és betöltésére szolgáló függvények is itt találhatóak.

A szoftver a következő csomagokból áll:



*43. ábra: Csomagdiagram*

### Implementáció során hozott döntések

Egy dolog volt, amit különösen bonyolult volt megvalósítani, mégpedig az a funkció, hogy átvehetjük a karakterünk gyermeke felett az irányítást a karakterünk halála esetén. Ehhez először annyit valósítottam meg, hogy a játékos karakter halálakor meghívtam a játékos karakter legidősebb gyermekére a *changeToPlayer* függvényt, amely inicializál egy Player típusú adattagot és ezt adja vissza, majd ezt értékül adtam a *You* tulajdonságnak. Ezzel a gyermek karakter statisztikáit jól át lehetett ültetni egy Player típusú objektumba, ezzel nem is volt baj, viszont akadtak még problémák ezzel.

Ha a gyermek másik szülője még él, akkor őt nem szeretnénk elveszíteni, ezért őt el kell tárolni valahogy. A *Player*-nek van egy *Partner* nevű tulajdonsága, ám ez nekünk nem jó, mivel a játékos karakter a játék folyamán tud szakítani a párjával és összejönni egy új személlyel, így az ott szereplő személy nem biztos, hogy a legidősebb gyermek szülője.

A probléma megoldására végül egy Dictionary-t alkalmaztam (*childParentPairs*), ahol eltároljuk az összes szülő-gyermek párt. A kulcs egy Person, aki nyilvánvalóan a szülő lesz, míg az érték Person-ök listája, ahol a szülőnek a gyermekei szerepelnek. Ha összejön a játékos karakter egy személlyel, akkor ebbe a Dictionary-be bekerül egy kulcs-érték pár, ahol a kulcs a személy, akivel összejött karakterünk, az érték pedig egyelőre csak egy üres lista. Sikeres gyermekvállaláskor a partnerünk értékében található lista kiegészítésre kerül a gyermekkel.

A játékos karakter halálakor meghívódik a *takeControlOfChild* függvény, amely elvégzi a szükséges műveleteket: átállítja a *You* tulajdonság értékét a legidősebb gyermekre meghívott *changeToPlayer* függvénnyel, majd ellenőrzi, hogy a hozzá tartozó szülő él-e még (azaz szerepel az emberek listájában). Ha igen, akkor ő lesz a másik szülő és a lista kiürítése után ismét hozzáadódik.

Ezen kívül, amit még említésre érdemesnek találok, hogy azokat az adattagokat, amelyeket kívülről is el kell érni és auto-property-k [5] segítségével valósítottam meg a kód könnyebb olvashatósága és a hatékonyság érdekében.

### Modell

#### LSModel csomag

Az LSModel csomagban találhatóak azok az osztályok, amelyek a fő játék logikájáért felelősek.

A képen szöveg, képernyő, képernyőkép látható

Automatikusan generált leírás

*44. ábra: LSModel osztálydiagram*

* *Car*: A játékban megvásárolható autók reprezentációjáért felel.
  + String *Type*: A jármű márkáját, típusát adja meg egy karakterláncban.
  + int *Price*: Az autó ára.
  + int *YearlyExpenses*: Az autó után fizetendő éves költségek.
  + Bitmap *Image*: Az autóhoz tartozó kép.
  + int *HappinessGain*: Ennyi boldogságot kap a játékos minden évben az autó után.
* *Home*: A játékban megvásárolható járművek reprezentációjáért felel. Ugyanazokkal a tulajdonságokkal rendelkezik, mint a *Car* osztály.
* *LicenseQuestionsModel*: Ez az osztály tartalmazza a jogosítvány vizsgánál feltehető kérdéseket egy Dictionary-ben, ahol a kulcs maga a kérdés, az érték pedig a helyes válasz sorszáma (*questionsAnswers*). Rendelkezik egy *Question* és egy *Answer* tulajdonsággal, amelyek a konstruktorban kapnak értéket. Ez egy véletlenszerűen kiválasztott kérdés-válasz pár lesz a már említett *questionsAnswers* Dictionary-ből.
* *Job*: Az elvállalható munkák reprezentációjáért felelős osztály.
  + Dictionary<String,int> *JobLevels*: A munkához tartozó szintek. A kulcs a pozíció neve, míg az érték a pozícióhoz tartozó fizetés.
  + University *DegreeNeeded*: Az adott munkához szükséges egyetemi végzettség.
  + int *MaxJobLevel*: a munka maximum szintje.
* *Person*: A játék karaktereit reprezentáló osztály.
  + String *FirstName*, valamint String *LastName*: a karakter vezeték- és keresztneve.
  + int *Age*: A karakter kora.
  + Gender *Gender*: A karakter neme.
  + int *Health*: A karakter egészsége.
  + int *Intelligence*: A karakter intelligenciája.
  + int *Appearance*: A karakter kinézete.
  + int *Happiness*: A karakter boldogsága.
  + int *Relationship*: A kapcsolatunk a karakterrel.
  + Player *changeToPlayer(Job Job, Home Home, University University, Car Vehicle)*: Átalakítja a Person-t Player-ré. Erre a funkcióra akkor van szükség, ha át akarjuk venni az irányítást a játékos karakter gyermeke fölött.
* *Player*: A játékos karaktert reprezentáló osztály. A *Person* leszármazottja. Fontosabb egyedi tulajdonságai:
  + List<Person> *Children*: A játékos karakter gyermekeinek listája.
  + Person *Partner*: A játékos karakter párja.
  + int *Money*: A játékos karakter egyenlege.
  + int *PromotionMeter*: Megadja, hogy a játékos milyen közel jár az előléptetéshez.
* *Sickness*: A különböző elkapható betegségeket reprezentáló osztály.
  + String *Name*: A betegség neve.
  + int *ApproximateEffectOnHealth*: Körülbelül ennyit fog levonni a betegség a játékos egészségéből a betegség. (ha kezelendő, akkor évente)
  + bool *NeedsMedicalAttention*: Szükség van-e orvosi beavatkozásra a gyógyításhoz.
  + int *ChanceToHeal*: Mekkora az esélye a betegség gyógyításának.
* *University*: Az egyetemi képzéseket reprezentáló osztály.
  + String *Type*: Egyetemi képzés típusa.
  + int *YearsToFinish*: Ennyi év alatt végezhető el az egyetem.
  + int *CostPerSemester*: Ennyibe kerül egy félévre az egyetem.
* *LifeSimModel*: A játék logikáját megvalósító osztály. Rendelkezik egy felsorolóval (*Gender*). Fontosabb adattagjai tulajdonságai tárolják többek között az adatelérést (*dataAccess*), az emberek listáját (*People*) a szülő-gyermek párokat (*childParentPairs*), a játékos karaktert (*You*), valamint az achievement-eket (*Achievements*). A különböző események is itt foglalnak helyet. Érdekesebb függvények:
  + A két konstruktor inicializálja a játék főbb elemeit (autók, házak, munkák, achievement-ek stb.). Amelyik csak a perzisztenciát tartalmazza paraméterként, azt a véletlenszerű emberrel való kezdéskor használjuk, míg a másikat a testreszabott emberrel való kezdéskor.
  + void *newGame()*: Inicializálja a játékos karakter szüleit a megkapott adatok alapján, statisztikát számol nekik, majd a szülők statisztikájának átlagából (plusz-mínusz 10) létrehozza a játékos karaktert is. Ezeket a karaktereket hozzáadja az emberek listájához.
  + void *age()*: Növeli az emberek listájában lévő karakterek korát, valamint ezzel együtt minimális mértékben változtatja a statisztikáikat. Ezekben segítenek a *calculateHealth* és a *calculateHappiness* függvények, ezekről később lesz szó. Ha egy személy meghal, az kiváltja a halálhoz tartozó *DeathEvent* eseményt. A játékos karakterhez tartozó események nagy része itt váltódhat ki, mint pl.: egyetem elvégzése (*SmartGraduateEvent*, *DumbGraduateEvent*), előléptetés (*PromotionEvent*), gyermek megszületése (*ChildBornEvent*), nyugdíjba vonulás (*RetirementEvent*), katonai misszió (*OnMilitaryMissionEvent*). A függvény végén meghívódik a *randomSickness* függvény, amely kis eséllyel egy betegséget adhat karakterünknek.
  + Tuple<Person,int> *newLove()*: Ez a függvény egy új potenciális párt generál számunkra. Úgy állítja be a korát, hogy ne legyen a karakterünknél sokkal fiatalabb, valamint idősebb (plusz-mínusz 2 év különbség). Ezen kívül kiszámolja a statisztikáink és a potenciális pár statisztikái alapján, hogy mennyi az esélyünk a generált karakternél. Végül visszaad egy Tuple-t, ami a partnerünk adatait, valamint az esélyünket tartalmazza egész számmal reprezentálva.
  + void *takeControlOfChild()*: Átadja az irányítást az eddigi játékos karakter legidősebb gyermeke fölött. Ezen kívül megkeresi, hogy ennek a karakternek él-e még a másik szülője, és ha igen, akkor hozzáadja őt is az emberek listájához.
  + int *chanceOfMutualLove(Person crush)*: Kiszámolja, hogy a játékos karakternek mennyi esélye van összejönnie a *crush* karakterrel. Ezt az értéket adja vissza. 0 és 100 között lehet, 20-as inkrementumokban.
  + int *calculateHealth(Person p*): Kiszámolja, hogy a *p* személynek mennyivel fog nőni/csökkeni az egészsége, és ezt az értéket adja vissza. Ehhez hozzájárul a karakter boldogsága, kora, valamint esetleges betegségei.
  + int *calculateHappiness(Person p)*: Az előző függvényhez hasonlóan kiszámolja, hogy mennyivel fog nőni/csökkeni a *p* személy boldogsága. Ez függ az ismerősei számától, a velük függő kapcsolatától, a birtokolt autótól, valamint járműtől.
* *LifeSimEventArgs*: A LifeSim eseményargumentumait tartalmazó osztály. Az *EventArgs* osztály leszármazottja. Ha egy adott eseménynek szüksége van egy vagy több paraméterre (mert mondjuk ki kell írni valamilyen adatot a nézetben), akkor azt innen kapja meg. Ide tartoznak például egy személy adatai, egy betegség adatai és az elért achievement neve.

#### MSModel csomag

Az MSModel csomag az aknakereső játék logikáját valósítja meg. Két osztály tartozik ide, a *Field* és a *MinesweeperModel*.

A képen szöveg látható

Automatikusan generált leírás

*45. ábra: MSModel osztálydiagram*ja

* *Field*: Ez az osztály a játék mezőit reprezentálja.
  + bool *Revealed*: Tulajdonság, amely megmondja, hogy az adott mező fel lett-e már fedve.
  + bool *Mine*: Tulajdonság, amely megmondja, hogy az adott mezőn akna található-e.
  + int *MinesInProximity*: Tulajdonság, amely megmondja, hogy az adott mező közvetlen szomszédságában hány akna található.
  + bool *Marked*: Tulajdonság, amely megmondja, hogy az adott mezőt megjelölte-e gyanús mezőként a játékos.
* *MinesweeperModel*: Ez az osztály valósítja meg a játék fő logikáját. A fontosabb adattagok, tulajdonságok, függvények a következők:
  + int *marks*: Ez az adattag adja meg, hogy a játékos még hány mezőt tud megjelölni.
  + Field[,] *MineField*: Tulajdonság, amely a játék mezőit tárolja el egy mátrixban.
  + bool *GameOver*: Tulajdonság, amely megmondja, hogy vége van-e a játéknak.
  + void *newGame(int fieldNum)*: A mezők inicializálására szolgáló függvény, a *fieldNum* paraméterbe annak a mezőnek a száma kerül, amelyikre először kattintott a játékos (azért, hogy az első kattintás soha ne jelentse a játék végét). Az algoritmus igyekszik nem túlságosan egymás mellé rakni az összes aknát.
  + void *recon(int fieldNum)*: A mezők felderítésére szolgáló függvény. A *fieldNum* számú mező az, amelyikre a játékos kattintott, ekörül fedi fel rekurzívan a szükséges mezőket (erről alább részletesebben írok). Ez a függvény ellenőrzi továbbá, hogy a játékos aknára kattintott-e. Ha igen, akkor kiváltja a *GameOverEvent* eseményt.
  + void *revealFields(int i, int j)*: Az előbb említett függvény végül ezt a függvényt hívja meg. Az *i* a kattintott mező X-, míg a *j* az Y-koordinátája. Ha a mező szomszédságában egy akna sincs, akkor a *revealRecursion* függvény segítségével felfedi a kattintott mezővel szomszédos mezőket is, feltéve, hogy azok nem tartalmaznak aknát és nincsenek megjelölve.
  + void *mark(int fieldNum)*: A *fieldNum* számú mezőt jelöli meg, amennyiben az még nincs megjelölve, nincs felderítve, valamint van még a játékosnak elegendő számú jelölése. Ha a mező már meg van jelölve, akkor kiveszi onnan a jelölést. Ez a függvény ellenőrzi továbbá azt is, hogy a játékos nyert-e (azaz az összes aknát megjelölte-e a mezőn). Ha igen, akkor kiváltja a *GameWonEvent* eseményt.

### Nézet

#### LSView csomag

Az LSView csomag tartalmazza a fő játék megjelenítéséért felelős osztályokat.



*46. ábra: LSView osztálydiagramja*

* *AchievementsWindow*: Ebben az ablakban achievement-jeinket tudjuk nyomon követni.

Tartalmazza a játék modelljét és az achievement-ek listáját. Az *AchievementsWindow\_Shown(object sender, EventArgs e)* függvény hívódik meg az ablak megjelenésekor. Ez kirajzol az ablakra annyi Labelt ahány achievement összesen elérhető a játékban, majd ezekbe beleírja az achievement-ek nevét és leírását. Végül kiszínezi a szövegeket attól függően, hogy az achievement-et megkaptuk-e már, vagy sem (piros – nem, zöld – igen).

* *GameStart*: A játék indításakor megjelenő ablak. Maga az ablak 3 Buttont, 2 RadioButton-t és egy TextBoxot tartalmaz.

Tartalmazza a játék modelljét és az adatelérést. A *randomButtonClick(object sender, EventArgs e)* függvény a ’Random ember’ gomb megnyomásakor hajtódik végre és inicializálja a modellt az egyparaméteres konstruktorral. Az *inputButtonClick(object sender, EventArgs e)* függvény a ’Testreszabott ember’ gomb megnyomásakor hajtódik végre, ez előhoz egy TextBoxot, valamint két RadioButtont, amelyek a név, valamint a nem megadására szolgálnak. Ha ezeket helyesen kitölti a játékos, akkor inicializálódik a modell a háromparaméteres konstruktorral. Mindkét esetben létrejön egy *MainGameWindow*, amelyet egyből meg is jelenít.

* *LicenseQuestionsWindow*: Ez az ablak jelenik meg, ha a játékos jogosítvány vizsgát szeretne tenni.

Tartalmazza a saját modelljét, valamint a fő játék modelljét. Az ablak megjelenésekor egy Label-be kiírja a megválaszolandó kérdést. Három Button közül lehet választani. A *checkAnswer(int num)* függvény leellenőrzi a válasz helyességét, és ezt az információt továbbítja a fő játék modelljének.

* *MainGameWindow*: Ez a játék fő ablakát megvalósító osztály. Az ablak több Panelből áll, amelyek között az alsó részen található Buttonök segítségével navigálhatunk. Ezeken a Paneleken különféle Controlok (Label, Button, ComboBox, ListBox, RichTextBox) foglalnak helyet.

Rendelkezik a fő játék modelljével, valamit temporary Stringet és ennek a hosszát eltároló adattagokkal (ezek a RichTextBox helyes működéséhez kellenek). Főbb függvények:

* + void *MainGameWindow\_Load(Object sender, EventArgs e)*: Az ablak betöltésekor végrehajtódó függvény. Feliratkoztatja a modell eseményeit az ezekhez tartozó függvényekre, beállítja a képeket, feltölti a ComboBoxokat, ListBoxokat, valamint kiírja egy MessageBoxba a játékos karakter és szülei statisztikáit.
  + *void Model\_DeathEvent(object sender, LifeSimEventArgs e)*: A modellben kiváltódó halál eseményhez tartozó függvény. Megnézi, hogy ki halt meg, majd ennek megfelelően frissíti a RichTextBoxot (logolja az elhunyt személyt), az ismerősök ListBoxát (kiveszi onnan az elhunyt személyt), valamint a fő panelen található információkat (ha a játékos karakter halt meg, és új játékot szeretne kezdeni).
  + *void ageButton\_Click(object sender, EventArgs e)*: Meghívja a modell *age()* függvényét, frissíti a RichTextBoxot az aktuálisan megtörtént eseményekkel, ezen kívül frissíti a játékos statisztikáihoz tartozó Labeleket, és ha szükséges, megváltoztatja a hozzájuk tartozó ikonokat.
  + *void refreshControls()*: Ez a függvény szolgál a Controlok frissítésére, amikor erre szükség van. A kornak megfelelően be- és kikapcsolja a szükséges Buttonöket, frissíti a Labelöket, törli a szövegeket a RichTextBoxból, valamint a ListBoxot is feltölti a szükséges adatokkal. Erre a függvényre új játék kezdésekor, valamint játék betöltésekor van szükség.

#### MSView csomag

Az MSView csomag az aknakereső kinézetéért felel. Egy osztályt tartalmaz, a *MinesweeperWindow*-t.



*47. ábra: MSView osztálydiagramja*

* *MinesweeperWindow*: Itt találhatóak a játék megjelenítéséhez szükséges adattagok, függvények. A fontosabbak a következők:
  + MinesweeperModel *msmodel*, valamint LifeSimModel *lsmodel*: az aknakereső és a fő játék modellje.
  + void *MinesweeperWindow\_Shown(object sender, EventArgs e)*: Ez a függvény hívódik meg, amikor megjelenik az aknakereső ablaka. Az aknamezőt reprezentáló gombok kirajzolásáért felelős.
  + void *btn\_MouseDown(object sender, MouseEventArgs e)*: Amikor a játékos megnyomja az egér valamely gombjával az egyik gombot/mezőt, akkor ez a függvény hajtódik végre. Bal egérgomb esetén felderítés, jobb egérgomb esetén megjelölés történik, ha teljesülnek a feltételek.
  + void *panelRefresh()*: Az előbbi függvényben hívódik meg, amennyiben még nincs vége a játéknak. Újra rajzolja a gombokat. A felfedett mezőket fehérre színezi, valamint ha van a szomszédságukban akna, akkor kiírja rá ezek számát. Felfedett aknás mező esetén pirosra színezi. Megjelölt mezőre zászló ikont rak. Ha ezek egyike sem teljesül, akkor a mező szürke marad.

### Perzisztencia

A perzisztencia réteg szolgál a játékállás mentésére és betöltésére, valamint az achievement-ek eltárolására. Ezek a műveletek a hatékonyság érdekében aszinkron [4] módon lettek megvalósítva. A hosszú távú adattárolás lehetőségét az **IPersistence** interfész adja meg. Az ebből az interfészből származtatott **TextFilePersistence** osztály felel a szöveges fájl alapú adatkezelésért. Az **Exception** osztályból származó **DataException** osztály a fájlkezelés során történt hibákat kezeli, pl. korruptálódott mentési fájl betöltése esetén.

A képen asztal látható

Automatikusan generált leírás

*48. ábra: Persistence osztálydiagramja*

*LoadGame(String path):*

* Ez a függvény a paraméterben megadott elérési útvonalon (path) lévő fájlt fogja beolvasni a jelenlegi játékállást. Task<List<String>>-gel tér vissza, amely egy Stringekből álló listát ad eredményül. Ebben a listában a betöltött adatok szerepelnek.

*SaveGame(String path, List<String> values):*

* Ez a függvény a paraméterben megadott elérési útvonalra (path) menti a jelenlegi játékállást, amelynek adatait a values paraméterben megadott tömbben kapja meg. Task visszatérési értékkel rendelkezik.

*AppendToFile(String path, int value)*

* Ez a függvény a paraméterben megadott elérési útvonalon (path) lévő fájlba beleírja az elmentendő achievement sorszámát, amelyet a value paraméterben megadott egész számmal kapja meg. Task visszatérési értékkel rendelkezik.

*LoadAchievements(String path)*

* Ez a függvény a paraméterben megadott elérési útvonalon (path) lévő fájlból olvassa be a már megszerzett achievement-eket. Task<List<int>>-tel tér vissza, amely egy intekből álló listát ad eredményül. Ebben a listában szerepelnek a megszerzett achievement-ek sorszámai.

## Felhasználói történetek/tesztelés

Az alábbi táblázatban azt láthatjuk, hogy a program futása során történő különböző események milyen állapotváltozásokat váltanak ki a programon belül. Az ’ID’ oszlopban egy azonosító található, amellyel az eseményre lehet hivatkozni. Az ’Event’ oszlopban található az esemény, amely kiváltja az állapotváltozást. A ’Given’ oszlopban az alkalmazás jelenlegi állapota látható. A ’When’ oszlopban az esemény kiváltásához szükséges interakció van feltüntetve. A ’Then’ oszlopban a változás szerepel, amelyet az esemény kiváltott.

A tesztelés is részben ezen táblázat alapján történt.

| **ID** | **Event** | **Given** | **When** | **Then** |
| --- | --- | --- | --- | --- |
| 1 | alkalmazás elindítása | az alkalmazás nem fut | alkalmazás megnyitása | az alkalmazás elindul |
| 2 | új játék kezdése véletlenszerűen generált emberrel | a játékos a kezdőképernyőn van | ’Random ember’ gomb megnyomása | új játék indul egy véletlenszerűen generált emberrel |
| 3 | ember testreszabása | a játékos a kezdőképernyőn van | ’Ember testreszabása’ gomb megnyomása | bejön egy panel, ahol a játékos beírhat egy nevet, valamint nemet választhat |
| 4 | új játék kezdése testreszabott emberrel | a játékos a testreszabó panelen van | a játékos beírt egy elfogadható nevet, valamint kiválasztotta karaktere nemét | új játék indul a testreszabott emberrel |
| 5 | játék mentése | a játék folyamatban van | ’Fájl’ menüponton belül mentés megnyomása, mentés helyének kiválasztása | a játék mentésre kerül egy .sav kiterjesztésű fájlba |
| 6 | játék betöltése | a játék folyamatban van | ’Fájl’ menüponton belül betöltés megnyomása, betöltendő fájl kiválasztása | a játékállás betöltésre kerül |
| 7 | játék előrehaladása | a játék folyamatban van, és a fő menüben vagyunk | ’+1 év’ gomb megnyomása | a játékos karaktere 1 évvel idősebb lesz, ezzel együtt statisztikái is változnak minimálisan |
| 8 | betegségek diagnosztizálása | a játék folyamatban van | ’Orvos’ gomb megnyomása | kiírásra kerülnek a betegségek, amikben a karakter szenved |
| 9 | betegségek gyógyítása | a karakter rendelkezik elég pénzzel/még nem töltötte be 18. életévét | ’Orvos’ gomb megnyomása | megpróbálkozás betegségek gyógyításával |
| 10 | Ismerős menü megnyitása | a karakter betöltötte 3. életévét, és nem az Ismerős menüben vagyunk | ’Ismerősök’ gomb megnyomása | előtérbe kerül az Ismerős panel |
| 11 | közös program ismerőssel | 10-es esemény végrehajtódott, és ki van választva egy ismerős | ’Közös program’ gomb megnyomása | adott ismerőssel való kapcsolat pontok növelése |
| 12 | új barát keresése | 10-es esemény végrehajtódott | ’Új barát keresése’ gomb megnyomása | ½ eséllyel találunk új barátot, ekkor bekerül az ismerőseink listájába |
| 13 | Szabadidő menü megnyitása | a karakter betöltötte 12. életévét, és nem a Szabadidő menüben vagyunk | ’Szabadidő’ gomb megnyomása | előtérbe kerül a Szabadidő panel |
| 14 | edzés | 13-as esemény végrehajtódott, a karakter rendelkezik elég pénzzel, vagy még nem töltötte be 18. életévét | ’Edzés’ gomb megnyomása | a karakter egészsége, kinézete és boldogsága megnő |
| 15 | olvasás | 13-as esemény végrehajtódott, a karakter rendelkezik elég pénzzel/még nem töltötte be 18. életévét | ’Olvasás’ gomb megnyomása | a karakter intelligenciája és boldogsága megnő |
| 16 | nyaralás | 13-as esemény végrehajtódott, a karakter rendelkezik elég pénzzel, és betöltötte 18. életévét | ’Nyaralás’ gomb megnyomása | a karakter boldogsága megnő |
| 17 | Szerelem menü megnyitása | a karakter betöltötte 14. életévét, és nem a Szerelem menüben vagyunk | ’Szerelem’ gomb megnyomása | előtérbe kerül a Szerelem panel |
| 18 | új partner keresése | 17-es esemény végrehajtódott, a karakter egyedülálló | ’Partner keresése’ gomb megnyomása | kiírásra kerül egy új potenciális partner neve, tulajdonságai |
| 19 | kapcsolat megpróbálása | 18-as esemény végrehajtódott | ’Kapcsolat megpróbálása’ gomb megnyomása | statisztikáinktól függő eséllyel összejövünk a pot. partnerrel, vagy nem |
| 20 | szakítás | 17-es esemény végrehajtódott, a karakter nem egyedülálló | ’Szakítás’ gomb megnyomása, figyelmeztetés elfogadása | a játékos karakter ismét egyedülálló lesz |
| 21 | gyermekvállalás | 17-es esemény végrehajtódott, a karakter nem egyedülálló, és betöltötte 18. életévét | ’Próbálkozás gyermekkel’ gomb megnyomása | ½ eséllyel gyermeket várunk |
| 22 | gyermek születése | 21-es esemény, majd a 7-es esemény végrehajtódott | - | megszületik a gyermek, aki bekerül az ismerősök listájába |
| 23 | Munka menü megnyitása | a karakter betöltötte 18. életévét, és nem a Munka menüben vagyunk | ’Munka’ gomb megnyomása | előtérbe kerül a Munka panel |
| 24 | munkára jelentkezés | 23-as esemény végrehajtódott, a karakter munkanélküli, és megvan a szükséges képzettsége | ’Jelentkezés’ gomb megnyomása | karakterünk megkapja az adott munkát |
| 25 | felmondás | 23-as esemény végrehajtódott, a karakter nem munkanélküli | ’Felmondás’ gomb megnyomása | a karakter ismét munkanélküli lesz |
| 26 | Lakás menü megnyitása | a karakter betöltötte 14. életévét, és nem a Lakás menüben vagyunk | ’Lakás’ gomb megnyomása | előtérbe kerül a Lakás panel |
| 27 | lakás vásárlása | 26-os esemény végrehajtódott, a karakter nem rendelkezik lakással és van elég pénz a választott lakásra | ’Vásárlás’ gomb megnyomása | a lakás a karakter tulajdonába kerül |
| 28 | lakás eladása | 26-os esemény végrehajtódott, a karakter rendelkezik lakással | ’Lakás eladása’ gomb megnyomása | a karakter eladja a lakást, ezzel visszakap egy kis részt az árából |
| 29 | Egyetem menü megnyitása | a karakter betöltötte 18. életévét, és nem az Egyetem menüben vagyunk | ’Egyetem’ gomb megnyomása | előtérbe kerül az Egyetem panel |
| 30 | egyetemre jelentkezés | 29-es esemény végrehajtódott, a karakter jelenleg nem végez egyetemi képzést | ’Jelentkezés’ gomb megnyomása | a karakter elkezdi elvégezni az adott egyetemet |
| 31 | egyetem elvégzése | 30-as esemény, majd a 7-es esemény végrehajtódott | - | a karakter megkapja a diplomát, és szükség szerint elkezdi fizetni a díjat |
| 32 | lottózás | a karakter betöltötte 18. életévét | ’Lottó gomb megnyomása’ | a karakter kis eséllyel nyerhet egy nagyobb pénzösszeget |
| 33 | Autó menü megnyitása | a karakter betöltötte 18. életévét, és nem az Autó menüben vagyunk | ’Autó’ gomb megnyomása | előtérbe kerül az Autó panel |
| 34 | jogosítvány vizsga | 33-as esemény végrehajtódott, a karakter nem rendelkezik jogosítvánnyal, és rendelkezik elég pénzzel | ’Jogosítvány letétele’ gomb megnyomása, figyelmeztetés elfogadása | megjelenik egy új ablak, ahol a vizsgakérdés, és a lehetséges válaszok láthatóak |
| 35 | rossz válasz | 34-es esemény végrehajtódott | valamelyik rossz válaszlehetőség megnyomása | az ablak egy üzenet után bezáródik, a pénz levonódik |
| 35 | rossz válasz | 34-es esemény végrehajtódott | valamelyik rossz válaszlehetőség megnyomása | az ablak egy üzenet után bezáródik, a pénz levonódik |
| 36 | jó válasz | 34-es esemény végrehajtódott | jó válaszlehetőség megnyomása | az ablak egy üzenet után bezáródik, a pénz levonódik, a karakterünk megkapja a jogosítványt |
| 37 | jármű vásárlása | 33-as esemény végrehajtódott, a karakter rendelkezik jogosítvánnyal, de autóval nem, és rendelkezik elég pénzzel | ’Vásárlás’ gomb megnyomása | a kiválasztott autó megvásárlása |
| 38 | jármű eladása | 33-as, esemény végrehajtódott, és a karakter rendelkezik autóval | ’Eladás’ gomb megnyomása | a karakter járművének eladása, amelynek árából egy részt visszakap |
| 39 | repülőgép-szerencsétlenség | a karakter pilótaként dolgozik/a 16-os esemény közben | pilótaként kis eséllyel a 7-es esemény után | 41-es esemény |
| 40 | autóbaleset | a karakter rendelkezik autóval | kis eséllyel a 7-es esemény után | csökken az egészség |
| 41 | halál | a karakter él | az egészség 0-ra csökkenése | a karakter meghal, kezdhetünk új játékot |
| 42 | irányítás átvétele legidősebb gyermek felett | a karakternek van gyermeke és a 4-es esemény végrehajtódott | - | az új karakterünk a régi karakterünk legidősebb gyermeke lesz |
| 43 | katonai bevetés kezdete | a karakter a katonai karrierben dolgozik | kis eséllyel a 7-es esemény után | megnyílik egy új ablak, ahol egy aknakeresőt kell lejátszanunk |
| 44.1 | mező felfedése | 43-as esemény végrehajtódott | a játékos bal klikkel rákattint egy mezőre, amin nincs akna | a mező és a környezete felfedésre kerülnek |
| 44.2 | mező felfedése | 44.1-es esemény legalább egyszer végrehajtódott | a játékos bal klikkel rákattint egy mezőre, amin akna van | az aknakereső bezáródik, a játékos egészséget veszít |
| 45.1 | mező megjelölése | 44.1-es esemény legalább egyszer végrehajtódott | a játékos jobb klikkel rákattint egy fel nem fedett mezőre | a mezőre egy zászló kerül |
| 45.2 | mező megjelölése | 44.1-es esemény egyszer végrehajtódott | a játékos jobb klikkel rákattint egy felfedett mezőre | nem történik semmi |
| 45.3 | mező megjelölése | 44.1-es esemény egyszer végrehajtódott | a játékos jobb klikkel rákattint egy megjelölt mezőre | megjelölés megszüntetése az adott mezőn |
| 45.4 | mező megjelölése | 44.1-es esemény egyszer végrehajtódott, valamint a játékosnak már csak egy jelölése van, a többi jelölése pedig helyes | a játékos jobb klikkel rákattint az utolsó aknás mezőre | az aknakereső bezáródik, a karakter kap egy kis bónusz pénzt, valamint közelebb kerül az előléptetéshez |

Ezen kívül az aknakereső játékon Unit Test-eket is végeztem, amelyek a *LifeSimUnitTest* projekt *MinesweeperModelUnitTest* osztályában találhatóak. A következő teszt függvényekkel rendelkezik:

* **NewGameTest**: Ez a függvény leellenőrzi, hogy az aknamező inicializálása helyesen történt-e. Ez akkor igaz, ha a mező, amire először lépünk az semmiképpen sem akna, nincsen megjelölve, nincsen a szomszédságában akna, valamint fel van fedve.
* **ReconTestWithoutMine**: Ez a függvény azt az esetet ellenőrzi, amikor olyan mezőre alkalmazunk felderítést, ami nem tartalmaz aknát. Ezt az alapján ellenőrizzük, hogy az adott mező felderítés után fel van-e derítve, tartalmaz-e aknát, valamint nincs megjelölve.
* **ReconTestWithMine**: Ez a függvény azt az esetet ellenőrzi, amikor olyan mezőre alkalmazunk felderítést, ami tartalmaz aknát. Ezt szinte ugyanúgy ellenőrizzük mint előbb, azzal a különbséggel, hogy itt kiváltódik a *GameOverEvent* esemény, amelyhez hozzárendeltük a **Model\_GameOverEvent** függvényt. Ez jelen esetben azt ellenőrzi le, hogy a modellben a *GameOver* property igaz-e.
* **MarkTestOnUnrevealed**: Ez a függvény azt az esetet ellenőrzi, amikor olyan mezőre alkalmazunk jelölést, amely nincs felderítve. Ehhez a felderítés után a mező felderítetlen és megjelölt kell, hogy legyen.
* **MarkTestOnRevealed**: Ez a függvény azt az esetet ellenőrzi, amikor olyan mezőre alkalmazunk jelölést, amely már fel van derítve. Ehhez a felderítés után a mező felderített és megjelöletlen kell, hogy legyen.
* **WinTest**: Ez a függvény ellenőrzi, hogy a játék megnyerésekor minden rendben zajlik-e. Ehhez mind a 15 aknát tartalmazó mezőt be kell jelölni. Ha ez megtörtént, akkor kiváltódik a *GameOverEvent* esemény, amelyhez hozzárendeltük a **Model\_GameWonEvent** függvényt. Ez azt ellenőrzi le, hogy a modellben a *GameOver* property igaz-e.

# Összefoglaló

Összességében szerettem ezen a projekten dolgozni. Úgy érzem, hogy sok hasznos tudást felszedtem a C# programozási nyelvvel és a WinForms könyvtárral kapcsolatban. Amit elvállaltam, azt sikerült teljesítenem, de azért bőven van még lehetőség a program továbbfejlesztésére. Az alábbiakban fel is sorolok egy pár ötletet erre.

## Továbbfejlesztési lehetőségek

Sajnos rá kellett jönnöm arra, hogy a WinForms-ban eléggé meg van kötve a fejlesztő keze a design terén. Amit meg lehet csinálni benne, azt könnyű összehozni, de sajnos nem sok a lehetőség. Ezért mindenképpen jó lenne átültetni az alkalmazást legalább WPF-be, mivel abban sokkal jobban testre lehet szabni a design elemeket.

Ezen kívül mindenképp jó lenne több olyan funkciót implementálni, amely a BitLife-nak is a része: több ország, ennek megfelelő pénznemek, vezetéknevek, keresztnevek, több szabadidős lehetőség, hírnév-rendszer, szerencsejáték (a lottón kívül), háziállatok stb.

Végül, de nem utolsó sorban jó ötletnek tartok egy online high-score rendszert, ahol több különböző érték szerint lehetne szűrni a játékosok végigvitt karaktereit: legmagasabb elért életkor, legtöbb elért pénz, legjobb átlag statisztikák, legtöbb egyszerre szenvedett betegség stb.

## Köszönetnyilvánítás

Szeretnék köszönetet mondani Várkonyi Teréz Annának, amiért elvállalta a témavezetői szerepet, valamint hasznos tanácsokat, tippeket adott a program továbbfejlesztésére konzultációink során. Szeretném továbbá megköszönni családomnak, hogy támogattak a tanulmányaim és szakdolgozatom elkészítése során. Végül, de nem utolsó sorban szeretném megköszönni barátaimnak, csoporttársaimnak az építő jellegű kritikákat és a folyamatos támogatásukat.

# Irodalomjegyzék

[1]

* https://docs.microsoft.com/en-us/dotnet/desktop/winforms/?view=netdesktop-5.0 [**Utolsó elérés dátuma**: 2021.05.13.]
* https://docs.microsoft.com/en-us/dotnet/desktop/winforms/overview/?view=netdesktop-5.0 [**Utolsó elérés dátuma**: 2021.05.13.]

[2] https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet-framework [**Utolsó elérés dátuma**: 2021.05.13.]

[3] https://visualstudio.microsoft.com/vs/ [**Utolsó elérés dátuma**: 2021.05.13.]

[4] https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2013/hh191443(v=vs.120) [**Utolsó elérés dátuma**: 2021.05.13.]

[5] <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/auto-implemented-properties> [**Utolsó elérés dátuma**: 2021.05.13.]

Egyéb: a játék során felhasznált ikonok a flaticon.com oldalról lettek letöltve. [**Utolsó elérés dátuma**: 2021.05.13.]