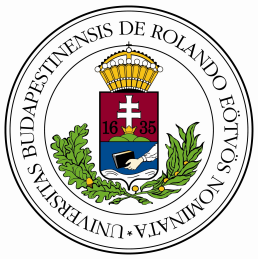


Eötvös Loránd Tudományegyetem

Informatikai Kar

Programozási Nyelvek és Fordítóprogramok Tanszék

Logikai ágensek szimulációja

dr. Tejfel Máté
adjunktus

Pitlik Mátyás
programtervező informatikus Bsc

Budapest, 2016

Tartalomjegyzék

1. fejezet

Bevezetés

Az ágens alapú szimuláció, modellezés fontos eszköz a programozásban. A logikai ágenseknek főleg a mesterséges intelligencia területén van jelentősége, közös jellemzőik ([1]), hogy

- rendelkeznek a környezet, a külvilág egy reprezentációjával,
- egyfajta következtetési módszert felhasználva új tudásra tesznek szert,
- ismereteik alapján döntéseket hoznak a további tevékenységüket illetően.

Érdemes még szót ejteni az ágensek rugalmasságáról is. Ha új információk birtokába jutnak, a meglévő adatok módosítása után gyorsan alkalmazkodnak a megváltozott körülményekhez.

Az ágensek sikeres működésének kulcsa általában az általános formában megszerzett információk átalakítása, kombinálása és a következtetési képesség. A következtetés terén gyakran használatosak a matematikai logika különböző formái, például az ítéletlogika vagy az elsőrendű logika. Noha az ítéletlogika lényegesen kisebb kifejezőerejű az elsőrendűnél, alkalmas a logika alapvető eszközeinek bemutatására.

Szakdolgozatom olyan logikai ágensek szimulációjához készült, amik speciális zérusrendű logikai formulák, klózok formájában tárolják az ismereteiket, melyeket a felszerelt közelségérzékelőik segítségével a környezetükről szereztek. Működésük során a következtetések levonására rezolúciós kalkulust használnak, mely mint döntési eljárás alkalmas arra, hogy az összeállított input adatokon az eljárás megállási feltételét elérve bizonyítottnak tekinthessük, hogy az ágens által vizsgált objektum az általa feltételezett anyagból van. A kitalált környezet ezáltal példát nyújt a zérusrendű rezolúció egy alkalmazási lehetőségére.

2. fejezet

Felhasználói dokumentáció

A felhasználói dokumentáció keretében először az elkészült szoftverről esik pár szó, majd az alkalmazás használatba vételéhez fontos információk kerülnek említésre a környezeti követelményekkel együtt, végül bemutatásra kerül az alkalmazás felhasználói felülete, részletesen ismertetve a futtatás során látható ablakokat.

2.1. Az alkalmazás

A program egy zérusrendű rezolúciót alkalmazni képes logikai ágensek szimulációjára készült asztali alkalmazás. A középpontba nem a rezolúciós kalkulus lépésről lépésre történő illusztrálása került, hanem a rezolúció mint eszköz kitalált környezetben való használata, valamint a levonható következtetések és a nem egyértelmű (kikövetkeztethetetlen) szituációk szemléltetése.

Ezek alapján a program alapvetően oktatási környezetben használható demonstrációs eszköznek tekintendő, melyben a biztosított kereteken belül tetszőlegesen megválasztott paraméterek mellett sem csökken az ágensek következtetésének ereje, kizárólag pontosan leírható együttállások esetén maradnak kikövetkeztetetlen objektumok. A szimuláció még érdekesebbé tételére több ágens együttes használatának lehetősége is biztosított.

A kitalált környezet megalkotásakor a valóság egyszerűsített leképezése ellenére az említett eszközök (közelségérzékelők) létező technológiákra utalnak. Ezért - amennyiben nincs fizikai vagy egyéb akadálya egy ilyen robot megalkotásának, és a

valódi érzékelőkből kinyert adatok összehangolásra kerülnek az ágens által elvárt bemenő adatokkal - az ágensek által használt háttérlogika akár valós élethelyzetekben is alkalmazásra kerülhetne.

2.2. Rendszerkövetelmények

Az alkalmazás használatba vételéhez az alábbi szoftveres környezet szükséges:

- Windows 7 vagy újabb operációs rendszer
- .NET 4.5.2 vagy újabb keretrendszer ([5])
- Microsoft SQL Server 2014 Express ([6])
- Microsoft SQL Server Management Studio

Minimális hardverkövetelmény (a telepítendő szoftverek igényeit is beleszámítva):

- 1 GHz-es processzor
- 1 GB RAM
- 10 GB tárhely a merevlemezen

Ajánlott hardverkövetelmény:

- 2 GHz-es processzor
- 2 GB RAM
- 10 GB tárhely a merevlemezen

Az alkalmazás beépített térképei 5 MB tárhelyet igényelnek, de a használat során adatbázisba mentett további térképek ezt jelentős mértékben növelhetik.

2.3. Az alkalmazás használatba vétele

Első lépésként biztosítani kell a követelmények között felsorolt szoftveres környezetet. Az SQL Server és a Management Studio telepítésénél minden maradjon az alapértelmezett értéken, különösen ügyelve a Windows-autentikáció kiválasztására.

A telepítés után indítsuk el a Management Studio-t, jelentkezzünk be a szerverre, és a feltüntetett szervernevet (ennek alakja általában MYPC\SQLEXPRESS) jegyezzük meg. Futtassuk a mellékelt szkript-fájlt (*SimulationData.sql*). Végső lépésként adjuk meg a most létrehozott adatbázis elérhetőségét a programnak. Ehhez

tetszőleges szövegszerkesztőben meg kell nyitni a *LogicalAgents.exe.config* állományt, majd azon belül a `data source=<SERVER_NAME>` részt le kell cserélni az előbb használt bejelentkezési adatok alapján `data source=MYPC\SQLEXPRESS` tartalomra.

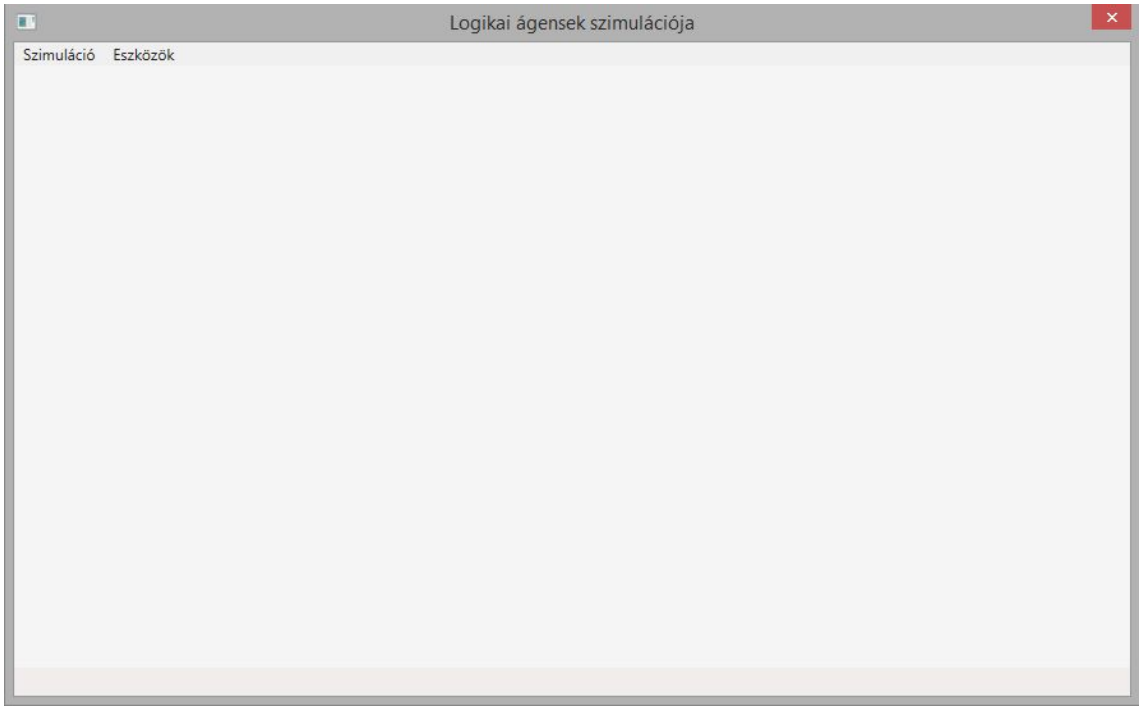
A fenti lépéseket végrehajtva a *setup.exe* futtatása után az alkalmazás használatba vehető és elindítható a Start menün keresztül is.

2.4. A felhasználói felület

Az alkalmazás egy főablakból és a használat során megnyitható két dialógusablakból áll. A párbeszédablakok megnyílása után a főablakra visszatérni nem lehet, a vezérlés akkor adódik vissza, ha a dialógusablak bezárásra került.

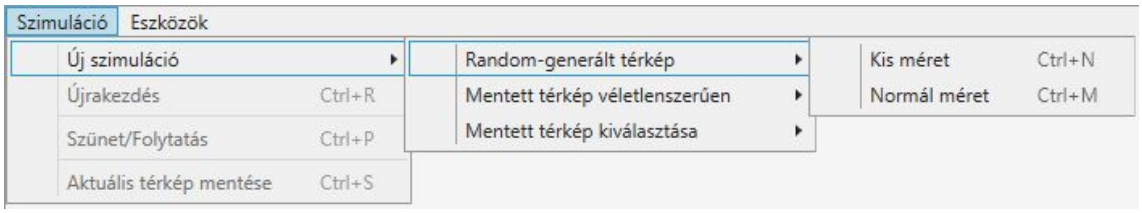
2.4.1. A főablak

A program indítása után a főablak kezdőállapota látható (2.1. ábra). Az ablak a képernyő közepén nyílik meg, mérete rögzített, nem módosítható.



2.1. ábra. Az alkalmazás kezdőképernyője

Az ablak felső részén egy menüsor helyezkedik el (2.2. ábra). A *Szimuláció* menüponton belül van lehetőség különféle új szimulációk indítására (*Új szimuláció* almenü), az aktuális szimuláció újraindítására (*Újrakezdés* menüpont), az automatikus léptetés megállítására és folyatatására (*Szünet/Folytatás* menüpont), valamint generált térképek esetén azok adatbázisba való elmentésére (*Aktuális térkép mentése* menüpont), hogy később újra be lehessen mutatni rajtuk a szimulációt. A mentés sikerességéről, sikertelenségéről felugró ablak informál. Az *Új szimuláció* három további almenüből áll, melyek mindegyike kis és normál méret szerinti menüpontokban végződik. A megadott paraméterek (ld. *Beállítások*) szerint létrehozható véletlenszerű térkép (*Random-generált térkép*), továbbá a tároltak közül is sorsolható ki térkép a mérete alapján (*Mentett térkép véletlenszerűen*), végül konkrét térkép (*Mentett térkép kiválasztása*) is betölthető a szimulációhoz. Az *Eszközök* menü egyetlen menüpontból, a *Beállítások*ból áll.

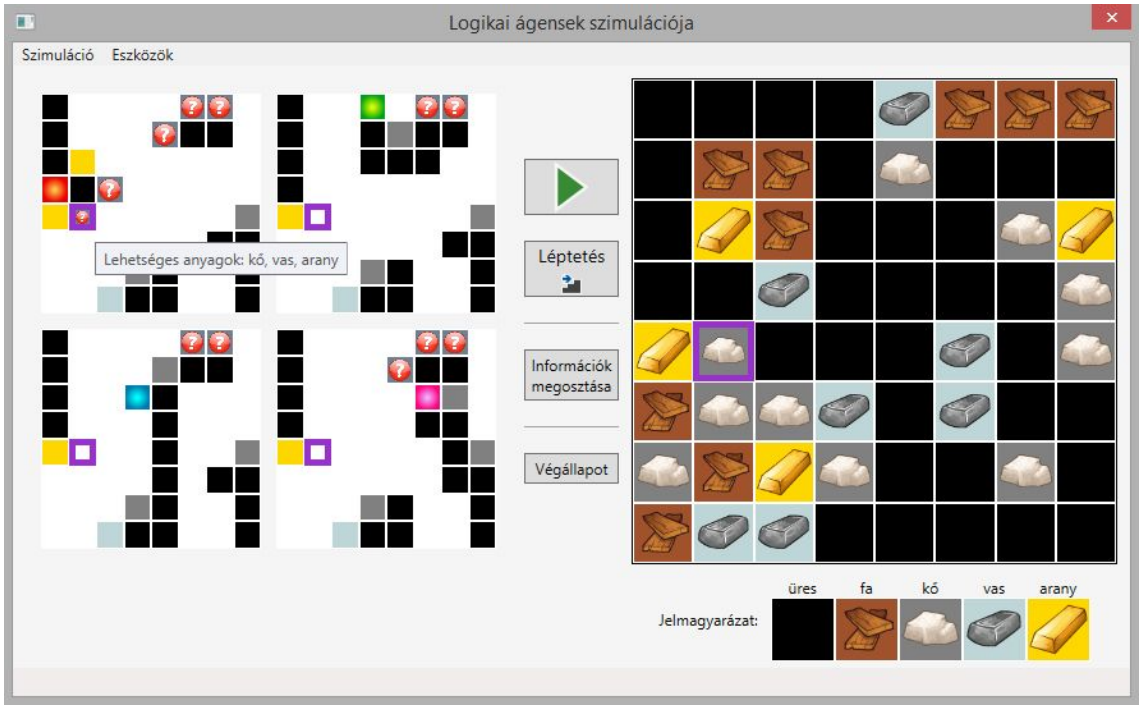


2.2. ábra. Az alkalmazás menüjének egy részlete

A felhasználói élmény növelése érdekében a menüpontok mindegyikéhez tartozik egy-egy billentyűkombináció, melyek kiolvashatók a menüből és igyekeznek igazodni más alkalmazásokban megszokhatott hatásokhoz.

Az alkalmazás elindítása után a főablak középső területe teljesen üresen áll, szimuláció-indítás hatására töltődik fel tartalommal, mely három hasábba rendeződik. Jobb oldalon a szimulációhoz kiválasztott térkép látható, ami alatt a térkép celláihoz tartozó jelmagyarázat helyezkedik el. Egy szimuláció futtatása során ez az egész terület változatlan marad, a következő szimuláció indításakor frissül.

Bal oldalon a szimulációban részt vevő ágensek számának megfelelő darabszámban a választott térképpel megegyező cellaszámú terület található, amennyiben az ágensek között nincs folyamatos ismeret-megosztás, mert ellenkező esetben egyetlen területen belül kerülnek megjelenítésre az ágensek. Ezek a térképpel teljes összhangban cellákra osztott területek az ágensek által a térképből addig felfedezett részt és a tapasztalt információkat jelenítik meg (2.3. ábra). Az ágensek által a cellákhoz rendelt adatok többek között azok színében fejeződnek ki.



2.3. ábra. Az alkalmazás pillanatképe

Fehér cella mutatja, ha az ágens nem rendelkezik információval az adott cellát illetően, beleértve ebbe azt is, hogy azt sem feltételezi, hogy az létezik és oda eljuthat.

Az átmenetes piros, zöld, rózsaszín és kék színezés jelöli az ágensek pozícióit.

Fekete színűre változik egy cella, ha az szabadnak (üresnek) bizonyult és az ágensek valamelyike már járt rajta.

A jobboldali térkép színeivel megegyező színű cellák esetén az ágens már rájött, milyen anyagú objektum található ott. Amíg ez nem következik be, szürke alapon piros kérdőjel jelzi, hogy az adott pozíción az addigi ismeretek alapján egyelőre meg nem határozható anyagú tárgy van.

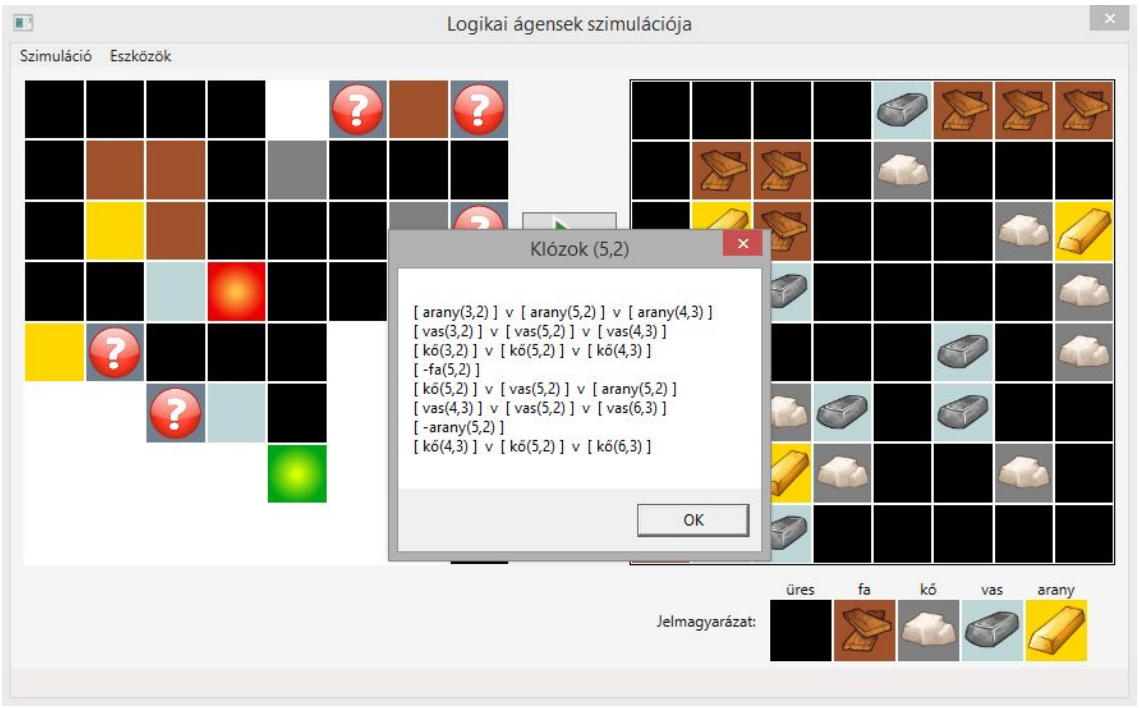
Az egér által mutatott cella kerete lilára változik arra az időtartamra, amíg az egér felette helyezkedik el, ez a hatás az azonos sor- és oszlopkoordinátákkal rendelkező további cellák esetén is érvényre jut, ami megkönnyíti a területek összevetését.

Az ágensek tevékenységének egyszerűbb nyomon követhetősége érdekében villogás kíséri a cellák színének megváltozását.

A színeken túl további információ-tartalom is kötődik a cellákhoz. Az egérmutató hatására rövid, szöveges formában értesülhetünk arról, hogy egy szabad cella esetében az ágens mely érzékelői jeleztek, ez a szöveg a pozícióra történő első belépés pillanatában bejegyzésre kerül, és onnantól nem is változik meg. Egy objektumot tartalmazó cella esetében pedig annak környezetében történt érzékelésekből követ-

kező lehetséges anyagok listája olvasható, mely másik irányból való megközelítés hatására módosulhat.

Mindezeken felül egy már a bejárt résszel szomszédos cellára kattintva felugró ablak sorolja fel azokat a zérusrendű klózokat (2.4. ábra), melyek legalább egy literálja az adott mezőre vonatkozó információt tartalmaz. Például a -arany(5,2) literál azt fejezi ki, hogy nincs arany az ötödik sor második oszlopában. A szimuláció során ez a lista új klózokkal egészülhet ki, vagy már meglévő klóz literálja törlődhet. Ilyen törlés akkor fordul elő, ha az ágens az érzékelői alapján klózokat állított össze, amik egy egyelőre ismeretlen cellára hivatkoznak, majd belép erre cellára, tehát kiderül, hogy az üres. Ekkor az összes olyan literál törlésre kerül, ami erre a szabad cellára vonatkozik, de az érintett klózok többi lierálja változatlan marad. A változások következtében kialakuló redundáns klózok egyből eliminálásra kerülnek.



2.4. ábra. Egy cellához tartozó klózlista

A vizualitás fokozása érdekében egy ágens esetén a térkép alatt ledek formájában kerül bemutatásra a érzékelők működése. Több ágens esetén az áttekinthető felület megőrzése miatt ez a kiegészítő információ nem jelenik meg.

A főablak középső hasábjában vannak elhelyezve gombok formájában a szimuláció vezérléshez szükséges legfontosabb funkciók. Egyrészt a *Szünet/Folytatás* menüpont gyorsabb elérését szolgáló gomb, rajta a szokásos módon váltakozó két ikonnal, majd a manuális léptetést lehetővé tevő gomb, melynek hatása megegyezik

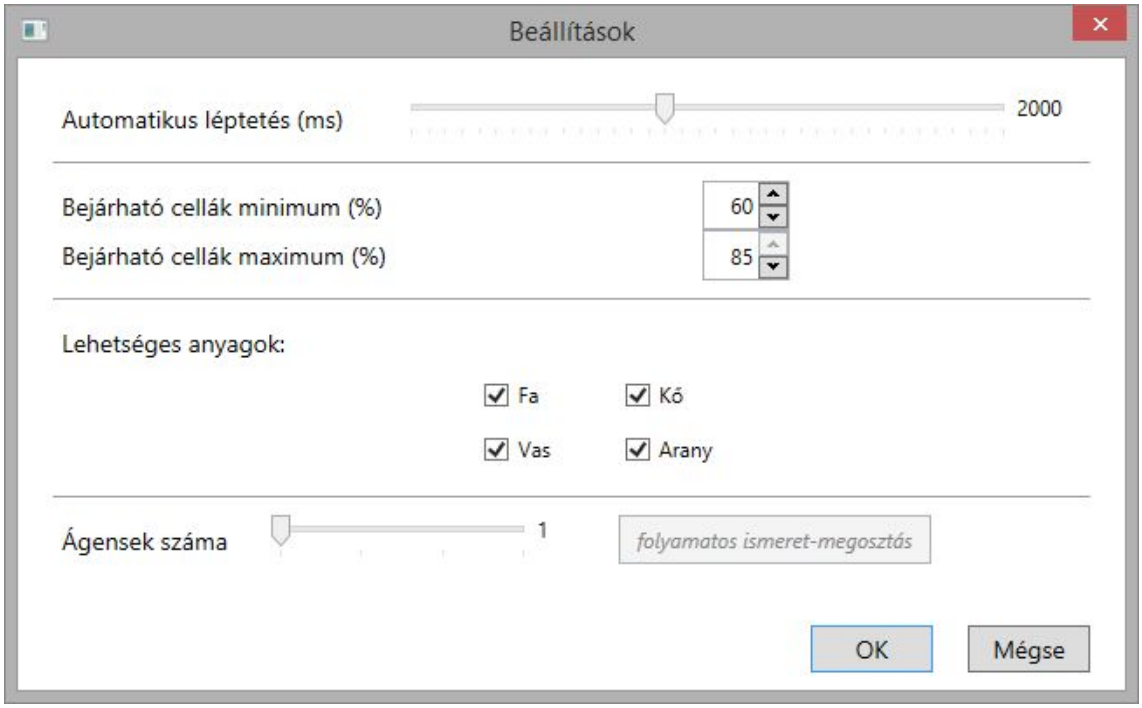
az időzítő által kezdeményezett automatikus léptetés hatásával, csupán a felhasználó igényeihez igazodó ritmusban történik mindez. Az *Információk megosztása* gomb kizárólag nem folyamatosan együttműködő ágensek választása esetén játszik szerepet és a felületen is csak ekkor látszódik. Gombnyomásra az egyébként csak rögzített számú lépésenként automatikusan bekövetkező információ-csere azonnal megtörténik. A *Végállapot* gombra kattintva a hátralevő teljes szimuláció játszódik le jelentősen felgyorsított ütemben, mely során már szüneteltetésre nincs lehetőség, cserébe az újraindítás menüpont tetszőleges számú futtatást biztosít.

A menüpontok és a felület gombjainak elérhetősége függ az alkalmazás pillanatnyi állapotától, és a használat során többször is megváltozhat. Új szimuláció indítására mindig van lehetőség, akkor is, ha egy szimuláció folyamatban van. A megállítás és folytatás értelemszerűen éppen futó szimulációnál aktív, a menüpont és a felületre kihelyezett gomb állapota folyamatosan összhangban van. Az újraindítás mindvégig elérhető, amint a kezdőállapotból az első szimuláció elindításra került, és mindig az aktuális szimulációt kezdi újra, akár folyamatban van még, akár befejeződött már. Térkép mentéséhez egy újonnan véletlenszerűen generált térképen kell futnia a szimulációnak, hiszen a beépített és a később mentett térképeket értelmetlen lenne újra elmenteni. A kézzel történő léptetés feltétele, hogy az időzítőt megállítsuk. Ugyanez érvényes a *Végállapot* és az *Információk megosztása* gomb aktívvá válására is, az utóbbi esetében azzal a kiegészítéssel, hogy nem folyamatosan együttműködő ágensekkel kellett indítani a szimulációt.

A programból való kilépés a főablak jobb felső sarkában elhelyezkedő bezáró gomb segítségével tehető meg, de a végleges bezárás előtt a felhasználónak meg kell erősítenie erre irányuló szándékát.

2.4.2. A *Beállítások* ablak

A *Beállítások* menüpontra kattintva megnyílik egy nem átméretezhető párbeszédablak a képernyő közepén, ami biztosítja a szimuláció összes módosítható paraméterének egy helyen való kezelését (2.5. ábra). Ehhez olyan grafikus elemek lettek választva, amik kizárják a hibás felhasználói input lehetőségét, így semmilyen lehetséges érték mellett sem omlik össze a program. Az alkalmazás kezdeti állapotában megnyitva a beállításokat az alapértelmezett értékek olvashatók ki.



2.5. ábra. A *Beállítások* párbeszédablak

Az automatikus léptetés időzítőjének értékét ezredmásodpercben kell megadni, mely az 500-4000 ms zárt intervallum kerek százas diszkrét értékeiből kerülhet ki.

A bejárható cellák arányának megadásakor legfeljebb egy 15 százalékos intervallumra szűkíthetők az értékek, melynek célja, hogy mind a felhasználói szándék minél pontosabb kifejezésére lehetőséget adjon, mind a véletlenszerű térképgenerálás hatékonyságát, gyorsaságát biztosítsa. Azonban így is elképzelhető (pl. 3x3-as méret esetén), hogy nem sikerül rövid időn belül a feltételeket kielégíteni, ekkor a program felugró ablakban tájékoztatja a felhasználót. (Ekkor természetesen meg lehet próbálni a beállítások megtartásával új térképet generálni, mert az üzenet felléphetett csupán szerencsétlen véletlen folytán is, de valószínűbb, hogy többszöri próbálkozás ellenére sem sikerül új szimuláció indításáig eljutni, ezért célszerű a beállított értékeket újragondolni.) A nem átléphető abszolút alsó és felső korlát célja, hogy a generált térkép a rezolúciós következtetés szempontjából érdekesebb legyen.

Az anyagok kijelölése csak a lehetséges anyagok halmazát adja meg, és a véletlenre van bízva, hogy a létrejövő térkép valóban tartalmaz-e minden bejelölt anyagot.

Egyetlen egy ágens szimulációjakor az ismeret-megosztásnak nincs jelentősége, ezért ilyenkor az arra vonatkozó választógomb azonnal inaktívvá válik. Aktív állapotban ezen gomb jelentését illetően rövid magyarázó szöveg jelenik meg, ha az egeret a gomb fölé mozgatjuk.

Az új beállítások csak akkor menthetők (*OK* gomb), ha legalább két anyag jelölőnégyzete be van pipálva, ennek nem teljesülése szintén felugró ablakos figyelmeztetést eredményez. A sikeres mentést követően a különböző beállítások különböző időpontokban jutnak érvényre. Az automatikus léptetés sebességének megváltozása azonnal megtapasztalható. Az üres cellák aránya és a lehetséges anyagok köre az elkövetkező véletlenszerűen generált térképekre fog vonatkozni mindaddig, míg azokat új beállításokkal felül nem írjuk. Az ágensek beállításai bármilyen típusú szimuláció indításakor (akár a jelenlegi újraindításakor) alkalmazásra kerülnek, a kezdőpozíciók egy-egy sarokcellára vagy azok közelébe helyeződnek.

A *Mégse* gombot használva az ablakban módosított beállítások elvesznek, és újbóli megnyitás esetén pontosan azok az értékek fognak megjelenni, mint amik a nem mentett módosítások beírása előtt érvényben voltak.

2.4.3. A *Térkép kiválasztása* ablak

Új szimuláció típusának megválasztása után a szimuláció a menüpontra kattintva azonnal indul. Kivétel ez alól a tárolt térkép kiválasztásának esete, ugyanis ez egy dialógusablakon keresztül tehető meg, mely alapértelmezetten a főablak közepén nyílik meg, és aminek átméretezése nem megengedett (2.6. ábra). Kis és közepes méret választásától függetlenül a megnyíló ablak szerkezete pontosan ugyanolyan.

Az ablak bal oldalán elhelyezkedő listából kell választani. A lista elemeinél rövid leírás is olvasható, mely az előre hozzáadott térképek esetében informál a méretről és az adott térkép jellegzetességéről a szimuláció szempontjából, illetve a később a felhasználó által elmentett random-generált térképeknél csak azok méretéről (és a véletlenszerű létrehozás tényéről). Egy listaelemet kijelölve a listától jobbra megjelenik a térképhez egy betekintő kép, ha a térkép előre hozzáadott volt. A későbbi mentésekhez egy alapértelmezett kép társul.

2.6. ábra. Dialógusablak térkép kiválasztásához

Az *Indítás* gombra kattintva máris indul a szimuláció, *Mégse* esetén visszatér a vezérlés a főablakhoz, és ha volt futó szimuláció, az fog folytatódni.

2.4.4. Hibaüzenetek

A fentebb ismertetett figyelmeztető, tájékoztató és hibaüzeneteken kívül elképzelhető, hogy az alkalmazás használata során más tartalmú üzenetek is előfordulnak. Ezek mindegyike a háttérben használt adatbázison végzett tevékenység kapcsán következhet be, ebből kifolyólag ezzel összefüggésben keresendő a kiváltó ok. Például ha az alkalmazáshoz mellékelt kezdeti adatbázisrekordok közül kitörlődik egy méretkategória, akkor ilyen méretű random-generált térkép választása esetén hibaüzenet jelenik meg: *A választott kategória nem létezik!* Egy másik lehetséges hibaüzenet látható, ha az adatbázis éppen térkép mentésekor valamiért nem elérhető: *A mentés nem sikerült!*

3. fejezet

Fejlesztői dokumentáció

3.1. Elméleti háttér

Az ítéletlogika (zérusrendű logika) a legegyszerűbb eszköztárral rendelkező formális logika, melyben egyszerű állításokat és az ezekből logikai kapcsolatok segítségével képzett összetett állításokat lehet megfogalmazni.

A zérusrendű logika leírásához használt ábécé ítéletváltozók szimbólumaiból (X, Y, Z), logikai műveleti jelekből $(\neg, \vee, \wedge, \supset)$ és elválasztójelekből $((,))$ áll. Az ezen alapuló szintaxist az ítéletlogikai formulák jelentik. Minden ítéletváltozó egyben formula is (ún. prímformula). Meglévő formulá(k)ból (A, B) a negáció $(\neg A)$ illetve a binér logikai műveletek segítségével $((A \vee B), (A \wedge B), (A \supset B))$ is formulák képezhetők. (Minden ítéletlogikai formula az előbbi konstrukciók véges sokszori alkalmazásával állítható elő.) Az így képzett formulákra szerkezetük alapján negációs, diszjunkciós, konjunkciós és implikációs formulákként hivatkozhatunk.

Egyes speciális alakú formulák külön elnevezéssel is rendelkeznek. Literálnak nevezünk egy ítéletváltozót vagy annak negáltját $(X, \neg X)$, melyben az ítételváltozó a literál alapja. Elemi diszjunkciót kapunk különböző alapú literálok diszjunkciójaként $(X \vee \neg Y \vee Z)$. Az elemi diszjunkciót más szóval klóznak nevezzük. Konjunktív normálformáról (KNF) beszélünk, ha egy formula elemi diszjunkciók konjunkciója $((X \vee \neg Y \vee Z) \wedge (\neg Z))$.

Az ítéletlogika szemantikája esetében csak az ítéletváltozókat kell interpretálni. Interpretációnak azt nevezzük, amikor minden ítéletváltozóhoz megadjuk, hogy *igaz* vagy *hamis* igazságértékű. Az interpretációkból kiindulva a formulákhoz helyettesítési érték rendelhető. Ez alapján egy interpretáció kielégít egy formulát, ha a formula helyettesítési értéke *igaz* az interpretációban. Egy formulát akkor nevezünk kielégíthetetlennek, ha egyetlen interpretáció sem elégíti ki. Egy formulahalmaz pedig olyankor kielégíthetetlen, ha minden interpretációban van olyan formulája, ami *hamis*.

A szemantikus következményfogalom összefüggésbe hozható a kielégíthetetlenséggel. Definíció szerint egy G formula szemantikus következménye az $\mathcal{F}$ formulahalmaznak ($\mathcal{F} = \{F_1, F_2, ..., F_n\}$), ha minden $\mathcal{F}$-et kielégítő interpretáció kielégíti G-t is. Tétel mondja ki továbbá, hogy $\mathcal{F}$-nek akkor és csak akkor következménye G, ha $\mathcal{F} \cup \neg G$ (azaz $F_1 \wedge F_2 \wedge ... \wedge \neg G$) kielégíthetetlen. Ezzel megkapjuk az egyik szemantikus eldöntésproblémát, ami nem más, mint egy ítéletlogikai formuláról eldönteni, hogy kielégíthetetlen-e.

Ennek eldöntése az ítéletváltozók számának növekedésével kezelhetetlenné válik, hiszen egy n-változós formula esetén akár 2^n különböző interpretáció vizsgálata is szükséges lehet. Az algoritmikus megoldás helyett ezért olyan módszerek születtek, ahol valamilyen eljárást követve annak eredménye alapján lehetséges választ adni a szemantikus eldöntésproblémára.

Az ilyen eljárást döntési eljárásnak, kalkulusnak nevezzük. Ennek lényege, hogy a bemeneti adatokból kiindulva az adatok egy sorozatát állítja elő, amit levezetésnek hívunk. A sorozat létrehozása levezetési szabályok segítségével történik. Minden döntési eljárás esetén létezik valamilyen megállási feltétel, ami lehet egy kitüntetett elem megjelenése a levezetésben, vagy jelentheti a sorozat speciális szerkezetűvé válását. A megállási feltételt elérve az eljárás véget ér, ami egyben annak az eldöntendő kérdésnek egyik válaszopcióját jelöli ki, amit a kalkulus segítségével meg szeretnénk válaszolni. (Kihangsúlyozandó azonban, hogy a megállási feltétel el nem éréséből nem következik a másik opció érvényessége.)

A zérusrendű rezolúció egy formula kielégíthetetlenségének eldöntésére szolgáló kalkulus. Az ítéletlogika egyik ún. eldönthető formulaosztályából, a konjunktív normálformából indul ki. Mivel a szemantikus következmény-fogalom ekvivalens módon megadható egy formula kielégíthetetlensége segítségével, továbbá mivel léteznek konstrukciós módszerek tetszőleges formulával ekvivalens konjunktív normálforma megadására, ezért a feladat az így előállítható KNF alakú formula kielégíthetetlen-

ségének vizsgálata, azaz a formulában található klózok halmazának kielégíthetetlensége.

Ezzel meghatároztuk a rezolúció mint döntési eljárás által használt input adatokat (klózhalmaz). A klózokra alkalmazható levezeti szabály pedig a rezolvens-képzés. Rezolvensről akkor beszélhetünk, ha adott két pontosan egy komplemens literálpárt tartalmazó klóz: $C_1 = C' \vee L_1$, $C_2 = C'' \vee L_2$, ahol $L_1 = \neg L_2$. Ezeknek a klózoknak létezik rezolvense és $C' \vee C''$ alakban írható fel. (Ha $C_1 = L_1$ és $C_2 = L_2$, akkor $res(C_1, C_2) = \square$, az üres klóz.)

Egy S klózhalmazból a C klóz levezetése egy olyan $k_1, k_2, ..., k_m \quad (m \geq 1)$ véges sorozatot jelent, hogy minden $j = 1, 2, ...m$-re $k_j \in S$ vagy van olyan $1 \leq u, v < j$, hogy $res(k_u, k_v) = k_j$. Továbbá annak is teljesülni kell, hogy $k_m = C$. A levezetés megállási feltétele az üres klóz levezetése, tehát a rezolúciós kalkulus eldöntésproblémája az, levezethető-e S-ből az üres klóz.

A szemantikus eldöntésproblémára a kalkulus helyessége és teljessége segít választ adni:

- A rezolúciós kalkulus helyes: ha S tetszőleges klózhalmazból levezethető az üres klóz, akkor S kielégíthetetlen.
- A rezolúciós kalkulus teljes: ha egy S véges klózhalmaz kielégíthetetlen, levezethető belőle az üres klóz.

A feladatban a táblaként leképezett helységben mozgó ágensek közelségérzékelői egy szomszédos cellán található fából, kőből, vasból vagy aranyból készült tárgy, objektum esetén jeleznek. Az érzékelés alapján szerzett információ egyből klózok formájában írható fel:

$$\{fa(3,1) \vee fa(2,2), \quad vas(3,1) \vee vas(2,2),$$
$$fa(3,1) \vee vas(3,1), \quad \neg k\H{o}(3,1), \quad \neg arany(3,1),$$
$$fa(2,2) \vee vas(2,2), \quad \neg k\H{o}(2,2), \quad \neg arany(2,2),$$
$$vas(2,2), \quad \neg fa(2,2)\}$$

Az ágens célja, hogy kikövetkeztesse a $(3,1)$ cellán található objektum anyagát. Ezért a remélt következmény-formula negáltját ($\neg fa(3,1)$) hozzáveszi a klózhalmazhoz. A rezolúciós levezetés a példában a következőképpen nézne ki:

$$fa(3,1) \vee fa(2,2), \quad \neg fa(2,2), \quad fa(3,1), \quad \neg fa(3,1) \quad \square$$

Mivel levezethető volt az üres klóz és a kalkulus helyes, ezért bizonyítottnak tekinthető, hogy a $(3, 1)$ mező objektumának anyaga fa.

A fenti logikai háttérre alapozva az volt a feladat, hogy elkészüljön egy WPF grafikus felületű alkalmazás zérusrendű rezolúciót alkalmazó logikai ágensek szimulációjához. A fejlesztés egyik fő szempontja volt, hogy az ágensek minél hatékonyabban oldják meg a kiszabott feladatukat, ugyanakkor minél könnyebb legyen őket a kitalált környezet esetlegesen megváltozó körülményeihez (új anyagok, konkrétabb célok) adaptálni. További szempontként az alkalmazás szimulációs eszköz jellegét is figyelembe kellett venni, azaz szemléletes megjelenítést kellett biztosítani az ágensek működésének nyomon követhetőségére.

3.2. Tervezés

A fejlesztési folyamat kezdetén a feladat részleteinek kidolgozása és a megvalósítandó funkciók kijelölése mellett sor került a fejlesztés mérföldköveinek meghatározására:

- Reprezentáció
  - a szimulációban kezelt anyagok pontos listája
  - a szimuláció környezetének reprezentációja
- Ágens-térkép kapcsolat
  - az ágensek felé biztosított interfész
  - az ágensek különböző lépési lehetőségei, bejárás
- Szimulációs adatok
  - adatbázis létrehozása,
  - adatbázis-elérés kiépítése
  - mintaadatok
- Ágensek logikai jellege
  - klózok létrehozása, tárolása, kezelése
  - rezolúciós kalkulus megvalósítása
- Felhasználói felület
  - az alkalmazás ablakai
  - a menü szerkezete
  - elrendezések
- Funkciók bővítése
  - végig együttműködő ágensek (közös tudásbázis)
  - külön tevékenykedő ágensek (időnkénti információ-megosztás)

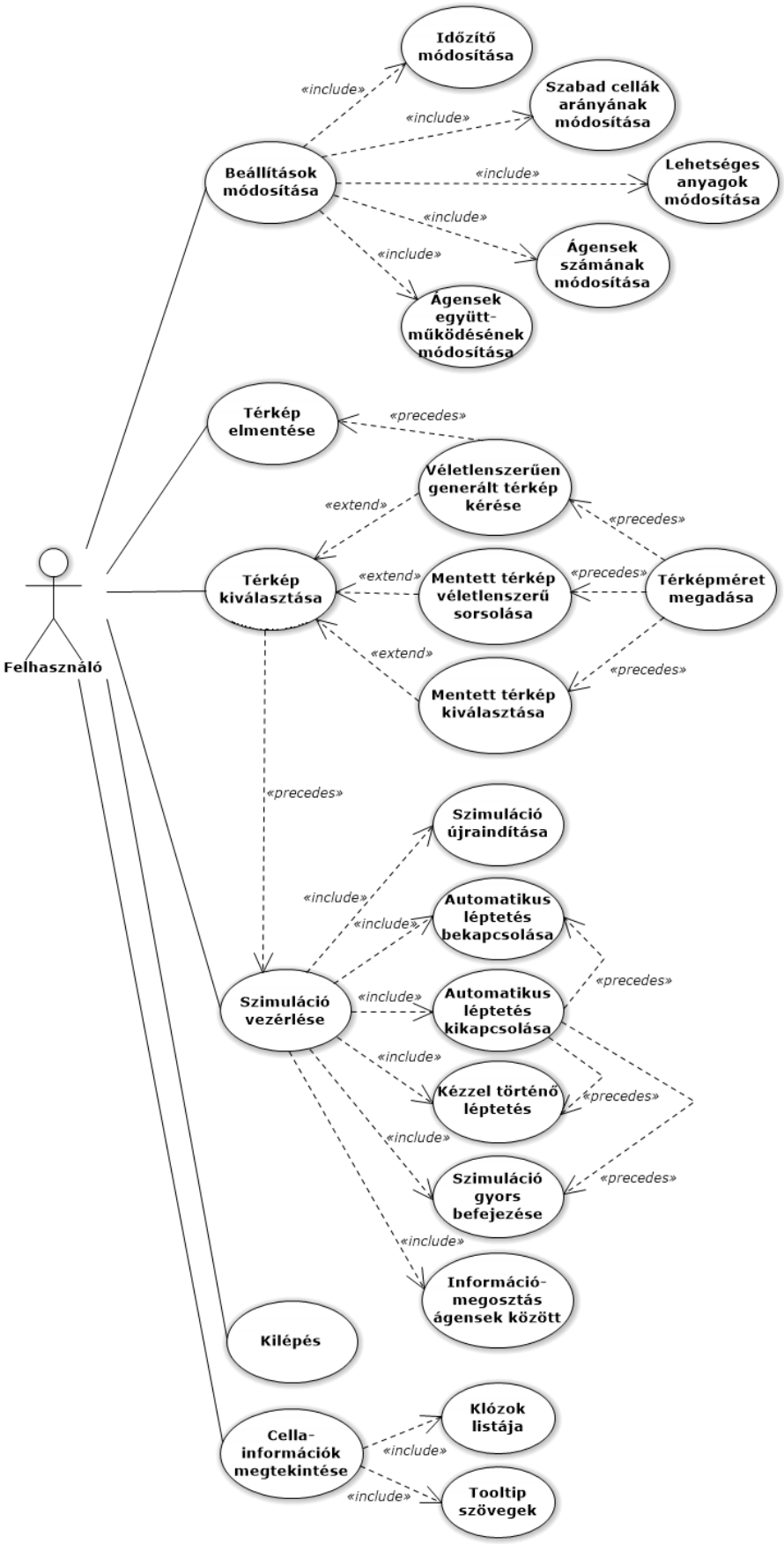
A fejlesztéshez használt szoftverkörnyezet:

- NET 4.5.2. keretrendszer
- Visual Studio 2015 fejlesztőkörnyezet
  - EntityFramework
  - Extended.Wpf.Toolkit
  - Expression.Blend.Sdk
  - Moq
- SQL Server 2014 Standard Edition

3.3. Felhasználói esetek

Az alkalmazás futtatása során az alábbi felhasználói esetek különíthetők el egymástól (3.1. ábra). Először ezek vázlatos felsorolása következik, majd pedig részletesebben is bemutatásra kerülnek.

- Beállítások módosítása
  - Időzítő módosítása
  - Szabad cellák arányának módosítása
  - Lehetséges anyagok módosítása
  - Ágensek számának módosítása
  - Ágensek együttműködésének módosítása
- Térkép elmentése
- Térkép kiválasztása
  - Véletlenszerűen generált térkép kérése
  - Mentett térkép véletlenszerű sorsolása
  - Mentett térkép kiválasztása
  - Térképméret megadása
- Szimuláció vezérlése
  - Szimuláció újraindítása
  - Automatikus léptetés bekapcsolása
  - Automatikus léptetés kikapcsolása
  - Kézzel történő léptetés
  - Szimuláció gyors befejezése
  - Információmegosztás ágensek között
- Cellainformációk megtekintése
  - Klózok listája
  - Tooltip szövegek
- Kilépés



3.1. ábra. Felhasználói esetek diagramja

3.3.1. Beállítások módosítása

Az *Eszközök* menü *Beállítások* menüpontján keresztül érhető el a beállítások párbeszédablaka. Befolyásolható az automatikus léptetés időzítőjének működése, a random-generált térképek szabad celláinak aránya és az objektumot tartalmazó cellák lehetséges anyagainak halmaza, a szimulációban részt vevő ágensek száma és az együttműködésük módja.

A szabad cellák megadása egy minimális és egy maximális értéken keresztül történik, ahol ha a minimális növelésével a feltételül szabott 15 százaléknál kisebbre változna a kijelölt intervallum mérete, a maximális érték automatikusan módosul ennek megakadályozására. Az együttműködési mód megválasztásának előfeltétele, hogy több mint egy ágens legyen kiválasztva a szimulációhoz.

3.3.2. Térkép kiválasztása

Három almenü hat menüpontja segítségével lehet hozzáférni a különféle térképválasztási lehetőségekhez a program futtatása során szinte bármikor (csak a nyitott dialógusablakok akadályozzák a funkciót).

Az eredményként megjelenített térkép szempontjából van köztük átfedés, mivel a *Mentett térkép véletlenszerűen* menüpont pontosan azok közül sorsol ki egyet, amik a *Mentett térkép kiválasztása* menüpontban felkínálásra kerülnének. A random-generálásnál is fenn áll a matematikai valószínűsége annak, hogy az új térkép valamely (akár elmentett) korábbival teljes mértékben megegyezzen, és ez a valószínűség a legkisebb méretű térképek esetén jelentősen megnő.

A generálás végén a térképnek minden beállításokbeli feltételnek meg kellene felelnie, de ha ez nem következik be bizonyos számú kísérleten belül, felugró ablak jelenik meg tájékoztatásképpen, így a program sem végtelen ciklusba nem kerül, sem a válaszidő nem növekszik meg túlságosan.

3.3.3. Térkép elmentése

Akkor van lehetősége erre a felhasználónak, ha véletlenszerűen generált térképet használ éppen, és az a térkép nem egy felhasználói mentés alapján került betöltésre. A sikeres mentést követően a térkép elérhetővé válik a méretének megfelelő térképválasztó menüponton keresztül. A megjelenő ablak legnagyobb sorszámú listaeleme lesz hozzárendelve.

3.3.4. Szimuláció vezérlése

Egy térkép kiválasztása után a szimuláció automatikusan elindul, vagyis az automatikus léptetés lép alapértelmezetten érvénybe. Ezt szüneteltetve a felhasználó kezébe kerül a szimuláció lefolyásának ütemezése, de az időzítő visszakapcsolható marad.

A kézzel történő léptetés az ágensek működésére, mozgására ugyanolyan hatással van, mint az automatikus léptetés. A Végállapot gomb szintén az ágenseket lépteti, csak olyan gyorsasággal, hogy az egyes lépések elkülöníthetetlenné válnak.

A szimuláció újraindításának legnagyobb előnye, hogy nem mentett térképen egymás után többször is futtatható a szimuláció, mentett térkép esetén pedig annak ismételt betöltése helyett tekinthető egyfajta kényelmi funkciónak.

Az információ-megosztásnak nem folyamatosan együttműködő ágensek esetén van jelentősége. Az ágensek által a szimuláció elejétől fogva vagy a legutóbbi megosztás óta külön-külön felfedezett területek ezáltal mindegyikük ismerethalmazához hozzáadódnak. A program gondoskodik róla, hogy fix számú lépésenként ezek az adatok mindenképpen kicserélődjenek, hogy ne fordulhasson elő olyan eset, hogy az ágensek egyesével a teljes térképet végigjárják.

3.3.5. Cellainformációk megtekintése

Az ágensek által megalkotandó térkép celláira kattintva felugró ablakban olvasható az oda kapcsolódó klózok listája egy egyszerű szöveges formában. A lista módosulásának nyomon követésével látható, az ágensek milyen helyzetekben mi-

lyen formában tárolják el a megszerzett ismereteket. Ezek pontosan azok a klózok, amiken a rezolúció algoritmusa végre lesz hajtva.

További cellainformációk jeleníthetők meg az egér segítségével tooltipek formájában, melyek vagy az érzékelők adta információkat, vagy a rezolúció megkezdésekor felhasználható feltételezések alapját tartalmazzák.

3.3.6. Kilépés

A programból való kilépés a főablak jobb felső sarkában elhelyezkedő bezáró gomb segítségével tehető meg, de a végleges bezárás előtt a felhasználónak meg kell erősítenie erre irányuló szándékát.

3.4. A rendszer felépítése

Az alkalmazás tervezése és megvalósítása a modell-nézet-nézetmodell (*Model-View-ViewModel, MVVM*) architektúra alapján történt.

Ez a minta a modell-nézet-prezentáció (*MVP*) architektúrából alakult ki, és elsősorban a felhasználói felületben rejlő lehetőségek kihasználásában tér el attól ([3]). A .NET platformon alapuló WPF és Silverlight alkalmazások fejlesztésekor egyfajta best practise-nek tekinthető a használata.

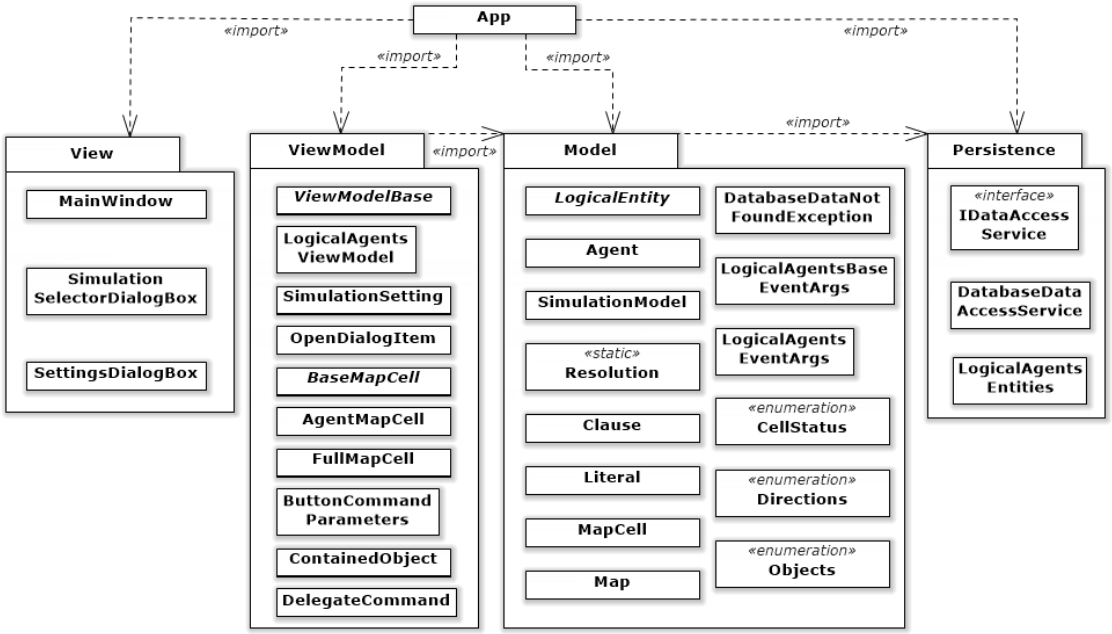
Jellemzője a programrétegek szerepkörének további csökkentése és pontosabb behatárolása, mely párhuzamos fejlesztési folyamatoknál egyértelmű teljesítménynövekedést eredményez, de az egyszemélyes fejlesztések esetén is fokozza az áttekinthetőséget és a módosíthatóságot, bővíthetőséget.

A modell-nézet architektúrához hasonlóan szintén a felhasználói felület és az üzleti logika szétválasztásából indul ki. A modellbeli elemek például olyan feladatokat láthatnak el, mint a felhasználó elé tárt adatok tárolása, vagy beviteli adatok validálása. A nézet ellenben mindössze a felület elemeinek összessége, amik a felhasználónak jelenítenek meg adatokat és interakciót biztosítanak a programmal.

Azonban mindez kiegészül a nézetmodellel, ami egyfajta átalakító, valamint a nézet absztrakciója. A modell bizonyos adatait teszi elérhetővé a nézet számára olyan módon, hogy annak felhasználása ott a lehető legegyszerűbb és leghatékonyabb legyen, továbbá tevékenységek végrehajtódásáért is felelős. A nézetmodellnek ezáltal a modell-jellege erősödik fel, hiszen megvalósítja a felhasználói esetek köré szerveződő működést és az ehhez szükséges üzleti logika elérését, így a nézet mint logikai rendszer tervezési minősége is javul.

A módszer legfontosabb jellegzetessége a nézetmodell és a nézet közti kapcsolat, melynek alapvető mechanizmusai az adatkötések, a változáskövetés és a parancsok ([7]). Ezek leírására WPF alkalmazásokban az XAML (*eXtensible Application Markup Language*) jelölőnyelv szolgál, mely egy XML-alapú formátum .NET keretrendszerbeli objektumok rendszerének kényelmes leírására. A kapcsolat az összes vizuális elem által örökölt DataContext tulajdonságon keresztül jöhet létre, vagy a nézetmodell a projekt erőforrásává (ResourceDictionary) tehető, hogy hivatkozni lehessen rá, amikor szükséges.

Az MVVM architektúrának megfelelően a programban `View`, `ViewModel` és `Model` névterek kerültek kialakításra. Sőt, a strukturáltabb felépítés érdekében a modellből kiemelésre került az adatelérés (`Persistence`), ezáltal négyrétegű architektúráról beszélhetünk. A program csomagszerkezete a 3.2. ábrán látható.



3.2. ábra. A program csomagszerkezete

A projekt fizikai felépítése szinte mindig a logikai felépítést követi. Egy-egy osztály külön-külön forrásfájlban (*<osztálynév>.cs*) került megvalósításra. Ez alól kivételt képez a *SimulationEnums.cs* fájl, ami a `CellStatus`, `Objects` és `Directions` felsorolási típusokat foglalja magába. A nézet más jellegű fizikai felépítése a `View` névtérnél lesz jellemezve.

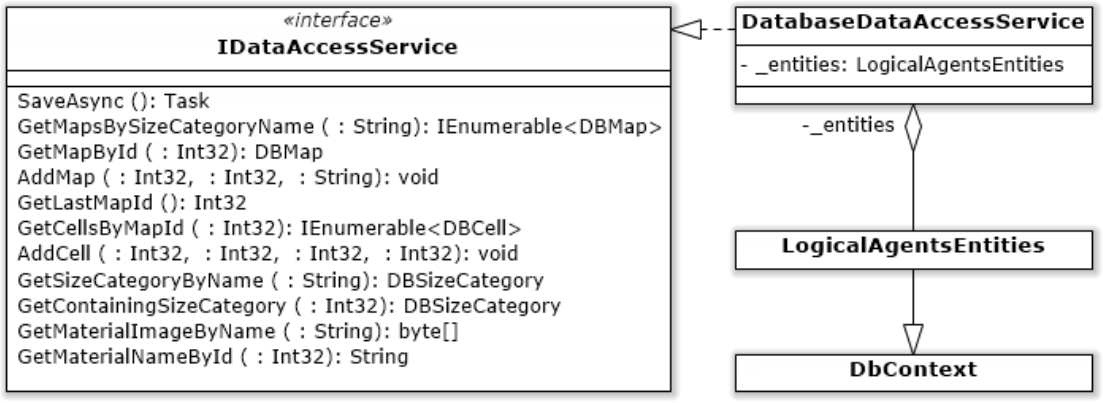
Az architektúra kialakításának egyik fő szempontja, hogy a programegységek között minél gyengébbek legyenek a kapcsolatok. Ennek egy lehetséges eszköze a függőség-befecskendezés (dependency injection), mely esetünkben az alkalmazás-környezet feladata. A nézetbe a nézetmodell tulajdonságon (DataContext) keresztül kerül (setter injection), a nézetmodellbe a modell és a modellbe a perzisztencia konstruktoron keresztül kerül (constructor injection). Ezek a lépések alakítják ki az alkalmazásban a függőségi viszonyokat. A függőségek ilyen módon való megadása az inversion of control elv alapján történik.

3.4.1. A `Persistence` névtér

Az alkalmazás modellje számára szükséges adatelérési műveletek az `IDataAccessService` interfészben (3.3. ábra) találhatók. Ezt az interfészt valósítja meg a `DatabaseDataAccessService` osztály, melynek implementációja az *EntityFramework* által biztosított adatbázis-elérésre támaszkodik. A hozzáférés a `DbContext` osztályból származtatott `LogicalAgentsEntities` parciális osztályon keresztül történik az objektum-relációs leképezés segítségével.

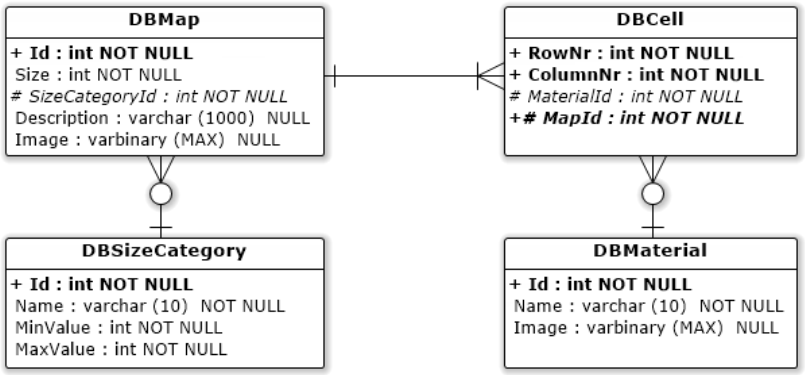
Az interfész megvalósítandó metódusai többnyire egyszerű lekérdezés-jellegű műveletek. A megvalósítás szempontjából csoportosíthatók aszerint, hogy végrehajtásukkor mely adatbázisbeli tábla rekordjai között kell keresni.

A térképek elmentési lehetősége miatt szükség van új adatok adatbázisba írására is, természetesen a perzisztencia réteg alacsony szintű műveleteinek megvalósítása független a felhasználás helyétől. Ennek első lépése az új adatok létrehozása (`AddMap()`, `AddCell()`), majd ezt követően explicit módon meg kell hívni a mentés műveletet (`SaveAsync()`), különben az új rekordok elvesznének. A mentés sikerességének vizsgálata fontos feladat, hiszen az adatbázisbeli idegen kulcs kapcsolatok miatt ügyelni kell a konzisztencia megőrzésére. Ha a fellépő probléma jelzése megtörtént, annak lekezelése már a felhasználási helyhez kötődő feladat.



3.3. ábra. A `Persistence` névtér

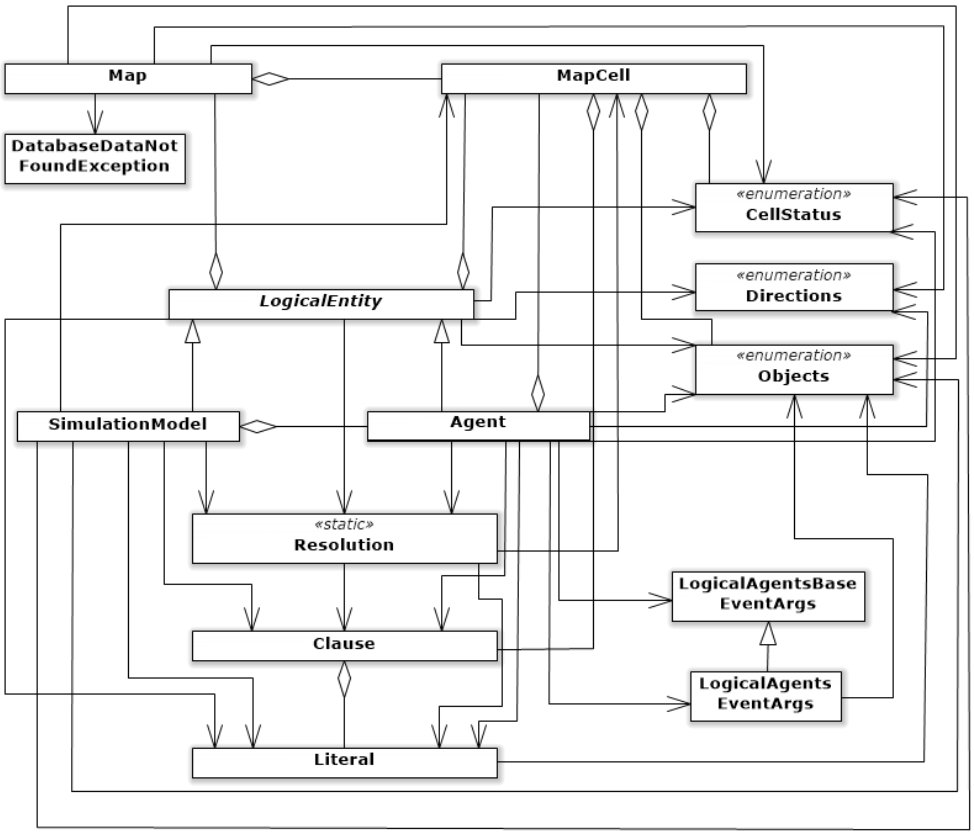
Az adatbázis (3.4. ábra) programkódba történő leképezése eredményeképpen elkészülnek a táblák relációsémáinak megfelelő osztályok (`DBMap`, `DBCell`, `DBMaterial` valamint `DBSizeCategory`), továbbá ezen típusok gyűjteményét (`DbSet<T>`) visszaadó tulajdonságok, amik a reláció-előfordulásnak feleltethetők meg.



3.4. ábra. Az adatbázis egyed-kapcsolat diagramja

3.4.2. A `Model` névtér

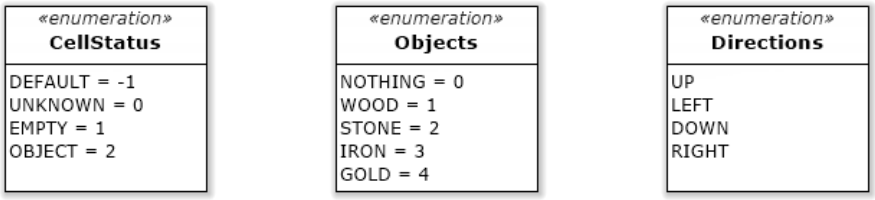
A `Model` névteret (3.5. ábra) a feladatot reprezentáló és megoldó osztályok építik fel. Ide tartozik maga a szimuláció, a térkép, amin a szimuláció fut, a szimulációban részt vevő ágensek, azok működéséhez elengedhetetlen zérusrendű logikai formulák, továbbá segédosztályok és az architektúrából adódó osztályok.



3.5. ábra. A `Model` névtér

A névtérbeli felsorolási típusok (3.6. ábra) jelölik ki a szimuláció határait, az ezekben szereplő értékekre épül rá az egész modell. Utólagos módosításuk a modellbeli megoldások újragondolásával járna. A `CellStatus` típus az ágensek környezetről alkotott tudását kategorizálja. A `Directions` által a térképen való mozgás leírása beszédesebb.

Az `Objects` típusértékei az adatbázis `DBMaterial` táblájával vannak összhangban. A felsorolási típus létrehozásának oka, hogy a reláció-előfordulás alapján körülményesebb és nehezebben megoldható lenne a térképen található anyagok típusának kezelése. Új anyag hozzáadását mindkét helyen külön jelezni kell, ehhez az ágensek működése a parszolás aktualizálása után azonnal alkalmazkodna, de a megjelenítést segítő műveletek és osztályok számos kiegészítést igényelnének.

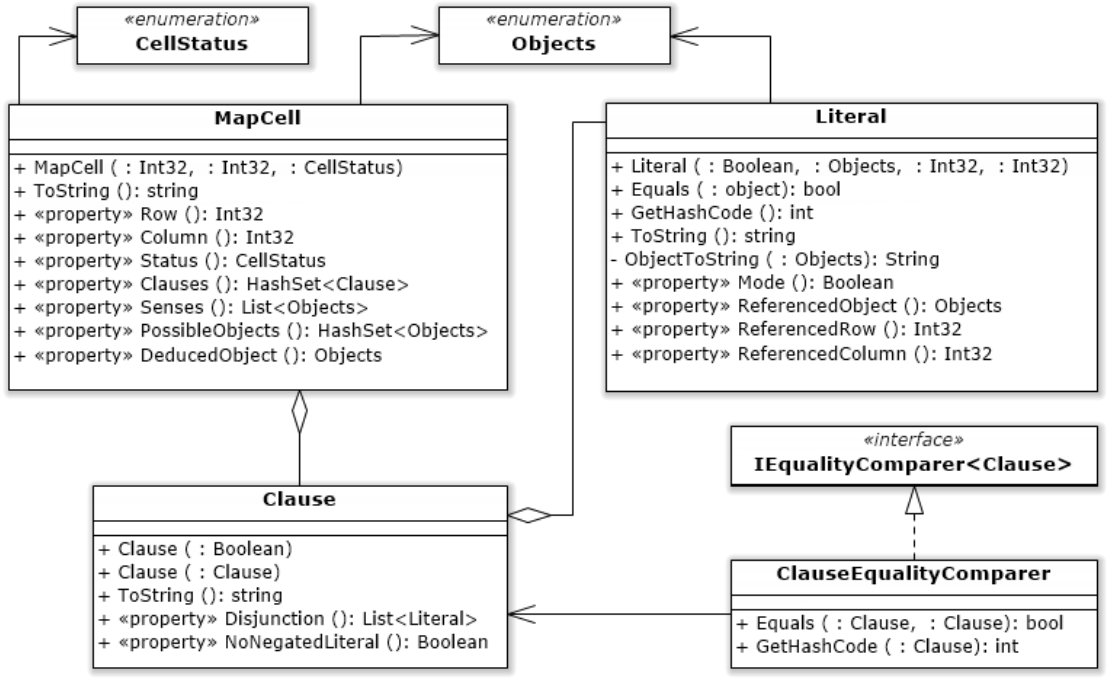


3.6. ábra. A felsorolási típusok

A `Literal` osztály (3.7. ábra) feladata a literálok leírása. A programban minden prímformulát (ítéletváltozót) a térképbeli pozíciójával és egy anyaggal azonosítunk. Literálnak nevezünk egy ilyen ítéletváltozót vagy annak negáltját. Tehát például egy literál a következő információt hivatott kifejezni: nincs arany a második sor ötödik oszlopában (-arany(2,5)). A felüldefiniált egyenlőségvizsgálat a tulajdonságonkénti egyezést ellenőrzi. A `ToString()` függvény két kényelmi funkciót is ellát, egyrészt a nullától való indexelést a hétköznapi szemlélethez igazítja, másrészt a modellbeli anyagneveket magyar szavakra cseréli.

A `Clause` osztály (3.7. ábra) alapvetően olvashatósági célokat szolgál, literálok listáját valósítja meg. A lista adatszerkezet előnye lesz a felhasználás során, hogy hatékonyan lehet tetszőleges elemeket hozzáadni, és a rövid listákon a törlés sem túl költséges az elemmozgatás ellenére. A rezolúció hatékonyságán javít, hogy külön tulajdonságba kiemelésre került, tartalmaz-e a klóz negált literált.

A klózok gyűjteményekben (`HashSet<T>`) való tárolása indokolttá tette egy szabványos összehasonlítást végző osztály implementálását. Noha a klózok literáljaiban szereplő anyagok csak a végigiterálás sorrendjében fordulhatnak elő, az azonos anyagra, de különböző cellákra vonatkozó klózok miatt a permutációk lehetőségét is figyelembe kell venni az egyenlőségvizsgálat megvalósításánál.



3.7. ábra. A `Literal`, `Clause` és `MapCell` osztályok

Az ágensek az összegyűjtött információkat cellához kötődően tárolják, ez valósul meg a `MapCell` osztály (3.7. ábra) által, melyben számos korábbi osztály felhasználásra került. Kiolvasható, mely pozícióra (`Row`, `Column`) vonatkoznak az ismeretek, kategorizálja (`Status`) a cellát a fent említett típus alapján. A környezetből adódó klózokat, érzékeléseket, feltételezéseket foglal magába, és helyet biztosít a kikövetkeztetett tudás tárolására (`DeducedObject`). Az osztály `ToString()` műveletének felüldefiniálásával a literálok szintjéig visszanyúlva próbálja a klózok listáját könnyen olvashatóvá és értelmezhetővé tenni. Pl.: [fa(2,5) v arany(2,5)]

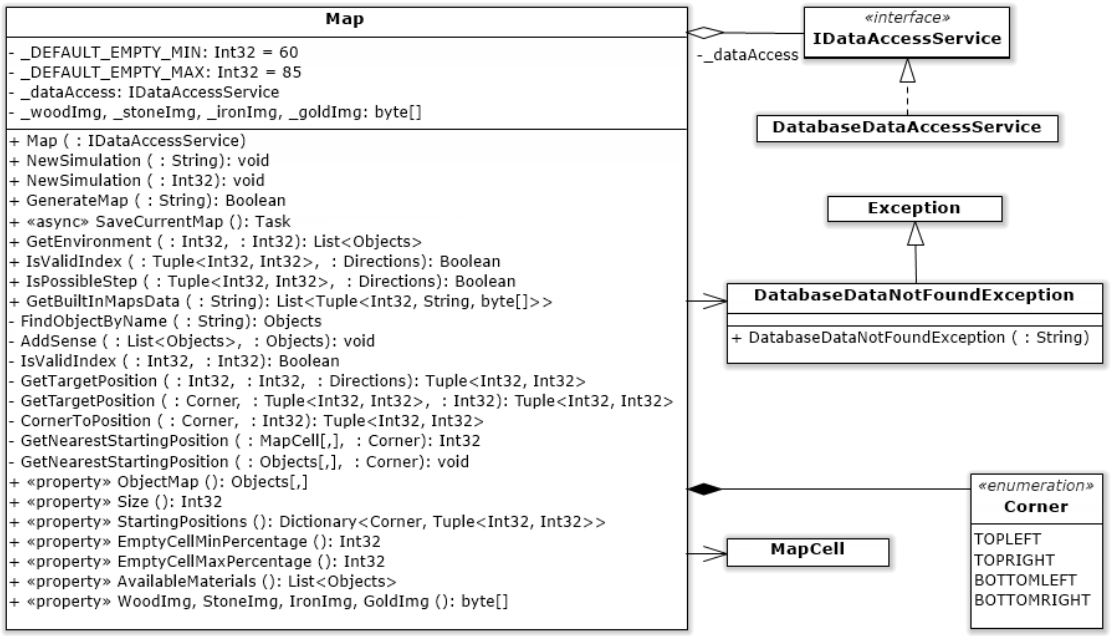
A `Map` osztály összetett feladatkörrel rendelkezik, az alkalmazás több komponensével is szoros kapcsolatban van, ennek oka, hogy a futó szimuláció az aktuális térképen alapul.

A `_dataAccess` mezőn keresztül biztosított a kapcsolat a perzisztencia réteggel. Az `IDataAccessService` interfész azonban lehetővé teszi, hogy az osztály ne függjön az adatelérés konkrét megvalósításától, így a perzisztencia cserélhető rétege az alkalmazásnak.

A interfész függvényeinek használatakor ennek az osztálynak a feladata, hogy a visszakapott eredményt értelmezze a szimuláció szempontjából, ezért a továbbfelhasználásra alkalmatlan esetekben hibát kell jeleznie. Ebből a célból került a modell-

be az `Exception` osztály leszármazottjaként a `DatabaseDataNotFoundException` saját kivétel, melynek példányosításkor kell megadni a tapasztalt hiba szöveges leírását, amit a felsőbb rétegek fognak felhasználni.

Új szimuláció indításakor mindig lesz olyan kérés, ami a `Map` osztályhoz (3.8. ábra) fut be. Az előbb említett adatbázis-elérés szempontjából ez lehet kulcs illetve nem kulcs szerinti térkép-lekérdezés (`NewSimulation()`), utóbbi esetben még véletlenszerű kiválasztás is szükséges a visszakapott értékek közül. Véletlenszerű térkép generálásakor (`GenerateMap()`) az adatbázisra azért van szükség, hogy az előálló térkép esetleges későbbi elmentésekor az idegen kulcs szerinti összefüggések ne sérüljenek. A generáló algoritmus részletesebb ismertetésére egy későbbi fejezetben fog sor kerülni. Minden említett esetben a végeredményként létrejövő, a szimulációhoz használandó térkép az anyagok egyszerű kétdimenziós tömbjeként képeződik le (`ObjectMap`). A folyamat végén az ágensek lehetséges kezdőpozíciói is kijelölésre kerülnek, ebben van szerepe a `Corner` beágyazott felsorolási típusnak (3.8. ábra).



3.8. ábra. A `Map` osztály

Egy térkép adatbázisba mentése (`SaveCurrentMap()`) két lépcsőben történik, először az általános jellemzők kerülnek be a `DBMap` táblába. A sikeres mentést követően elérhető az automatikusan növelt térkép-azonosító, és ezt az értéket kell felhasználni minden egyes cella leképezésekor.

Az osztály az ágensek működéséhez is alapvető publikus metódusokat biztosít. Mivel itt tárolódik a térkép részletes felépítése, kézenfekvő az ágensek pozíciótól füg-

gő információigényének ez alapján történő kielégítése. Az osztály foglalkozik a térkép határaival, vagyis az érvényes indexek kezelésével (`IsValidIndex()`), egy cella létező szomszédai alapján meghatározza, az adott helyen mit kell jeleznie az érzékelőknek (`GetEnvironment()`), és innen származik azon ismeret is, hogy az ágens által következőnek felfedezendő mező szabad-e vagy objektumot rejt (`IsPossibleStep()`).

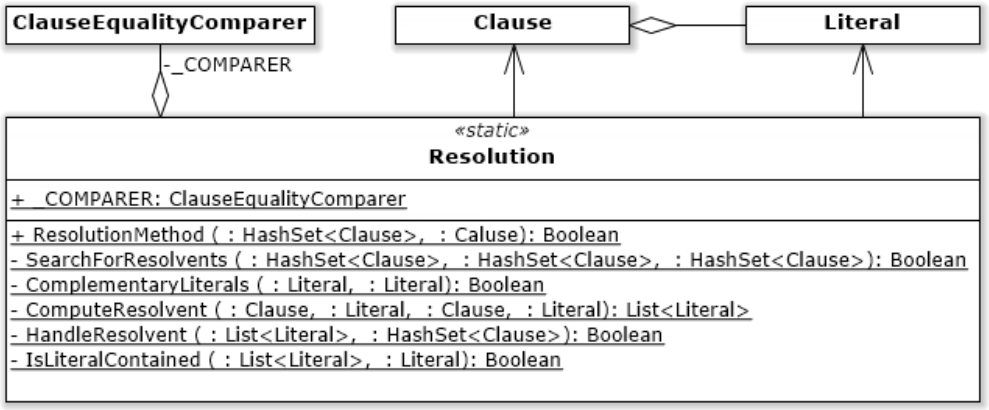
Mivel a szimuláció vezérlése, finomhangolása a felületen a felhasználó által történik, ezért a nézetmodell felé is kapcsolatot kell kiépíteni. A mentett térképek listából való kiválaszthatóságához jól megjeleníthető formában kell kinyerni az adatbázisból az összes kért információt (`GetBuiltInMapsData()`). A szimuláció beállításainak egy része a random térképgenerálás algoritmusát befolyásolja, emiatt ezen értékek változásait is szinkronizálni kell.

Együttműködő ágensek esetén a logikai tevékenység csak az ágensekhez kötődik, a szimuláció minimális koordináló szereppel bír. Ellenben a folyamatos ismeretmegosztás megszüntetésével a szimulációra hárul a feladat, hogy az ágensek működését – ugyanazon logikai tevékenység szűkített eszközkészletét is felhasználva – összehangolja. Emiatt került létrehozásra egy absztrakt osztály (`LogicalEntity`), ami a minimális, közös logikai tevékenység meglétét biztosítja a belőle származó osztályoknak, amit azok tetszőlegesen felhasználhatnak működésük során.

A származtatott osztályok a `_fullMap` mező által birtokolják a szimuláció teljes térképét, ismereteiket a környezetről egy saját térképbe képezik le (`Map`). Rezolúciót alkalmazhatnak objektumok kikövetkeztetésére (`TryExtendedResolution()`). Ennek elkülöníthető lépéseit külön eljárásokba szerveztük.

A szimulációban a következtetés, a feltételezések bizonyítása zérusrendű rezolúciós kalkulus alapján történik. Mivel az algoritmus teljesen független a rendelkezésre álló klózokon felüli mindennemű információtól, létrehoztuk a `Resolution` statikus osztályt (3.9. ábra). Ennek egyetlen publikus metódusa a `ResolutionMethod()`. Az algoritmus egyes részei segédműveletekként lettek megvalósítva. A rezolúció implementálásának részletesebb ismertetésére egy későbbi fejezetben fog sor kerülni.

Ágenseket a `LogicalEntity`-ből származó `Agent` osztály (3.10. ábra) példányosításával hozhatunk létre. Ezek önműködőnek nevezhetők, az inicializálást követően a `Move()` publikus metódussal lehet kezdeményezni a lépésüket. Ezen belül összegyűjtik a szükséges információkat a környezettől, frissítik és kiegészítik a tárolt ismereteiket, és a megjelenést befolyásoló tényezők kapcsán jelzést adnak.



3.9. ábra. A `Resolution` osztály

Minden ágens öröklődés útján rendelkezik az aktuális térkép egy példányával, így annak publikus műveletei segítségével, valamint a már felfedezett részek alapján döntenek a következő lépés irányáról. Ezek a publikus műveletek csak olyan információkat nyújtanak, melyek nem szüntetik meg a következtetés kényszerét a térkép felfedezéséhez.

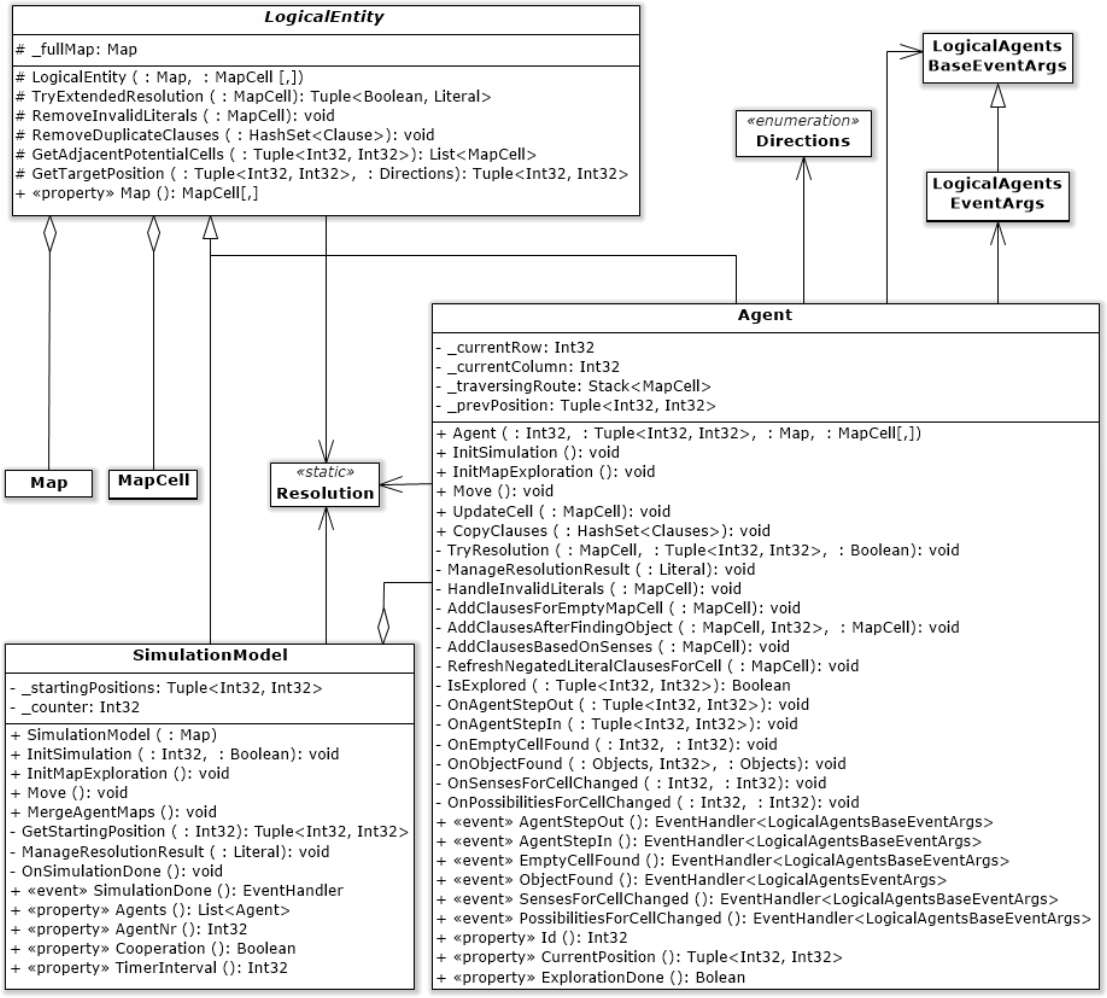
A megszerzett tudást a korábban taglalt `MapCell` osztály példányainak megfelelő tulajdonságaiban tárolják és kezelik. Általánosan elmondható, hogy minden egyes lépés együtt jár érzékelésekből adódó új, adott szituációnak megfelelő szerkezetű klózok megkonstruálásával. A rezolúció számításigényének csökkentésére ismétlődő klózokat nem tárolunk, és a később törlések által fellépő redundanciát is újra és újra megszüntetjük.

Az ágensek a kezdőpozíciójukból induló és ott is végződő mélységi bejáráshoz hasonló utat írnak le. Az indulástól az aktuális pozícióig vezető utat számon tartják (`_traversingRoute`), mely a megvalósítás szintjén egy verem adatszerkezet (`Stack<T>`). Csak üres cellákat tesznek bele, a felfedezett objektumok nem befolyásolják az útvonalat. Az útvonal építésének alapja, hogy tisztában vannak saját térképen belüli helyzetükkel (`CurrentPosition`).

A grafikus felületnek szüksége van az ágensek számos tulajdonságára. Az eseményvezérelt alkalmazásokra jellemző módon a változásokat események kiváltásával jelezzük. Ehhez létrehoztuk a `LogicalAgentsBaseEventArgs` és a belőle származó `LogicalAgentsEventArgs` osztályokat, melyek segítségével a kiváltott esemény képes olyan addicionális információkat is továbbítani, mint az ágens azonosítója, és az érintett sor- ill. oszlopkoordináták. A kiváltott események részben az ágensek moz-

gásával vannak összefüggésben, részben azt jelzik, ha változás történt a térképről kialakított ismereteiket illetően.

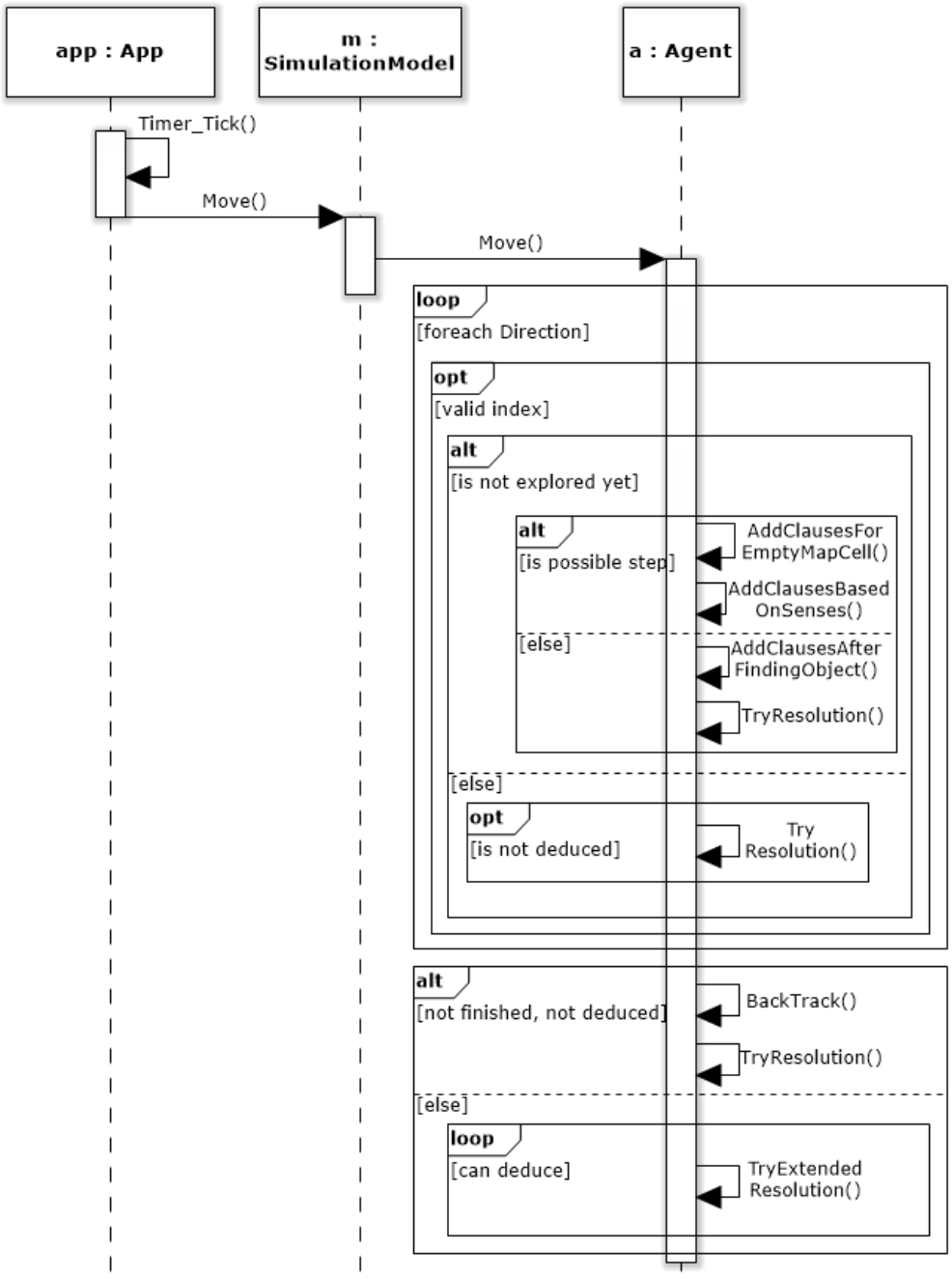
Ha a rezolúció segítségével történő következtetés sikeres, ki kell egészíteni a bejárás során szerzett információkat (`ManageResolutionResult()`). A bekövetkezett változásokról a felsőbb rétegeket is értesíteni kell.



3.10. ábra. A `LogicalEntity`, `Agent` és `SimulationModel` osztályok

A `SimulationModel` osztály (3.10. ábra) szolgál a szimuláció komponenseinek összefogására, és szintén a `LogicalEntity` leszármazottja. Mentesíti az alkalmazáskörnyezetet annak az üzleti logikai folyamatnak a kezelésétől, amivel egy szimuláció inicializálása és lejátszása jár. Az ágensek önműködő jellege miatt ez a feladat alapvetően az ágensek hasonló nevű műveleteinek továbbhívásából áll (pl. 3.11. ábra). Viszont nem folyamatosan együttműködő ágensek esetén a végeredmény szempontjából is központi szerepet tölt be az osztály.

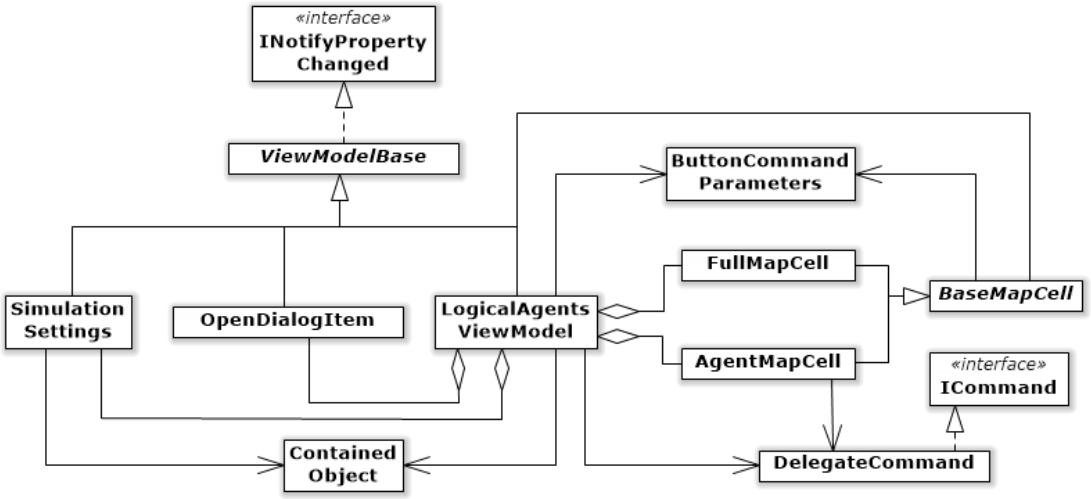
A szimulációban részt vevő ágensek ismereteit összegyűjti, kiszűri a releváns tartalmat, rezolúcióval új információkra próbál szert tenni (`MergeAgentMaps()`). Az összesített adatokat visszajuttatja (`UpdateCell()`) az ágenseknek a szimuláció folytatásához.



3.11. ábra. Szekvenciadiagram az időzítő lejárta által kifejtett hatásához

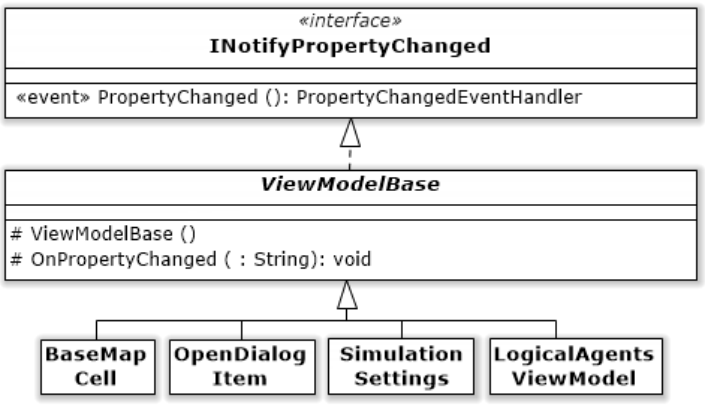
3.4.3. A `ViewModel` névtér

A névtérbeli osztályok (3.12. ábra) feladata, hogy megfelelő kapcsolatot építsenek ki a modell és a felhasználói felület között. Ehhez szükséges, hogy az alkalmazás ablakaihoz eljussanak a megjelenítendő adatok és a megváltozásukra vonatkozó jelzések, valamint a felhasználói tevékenység a megfelelő hatást váltsa ki a programban.



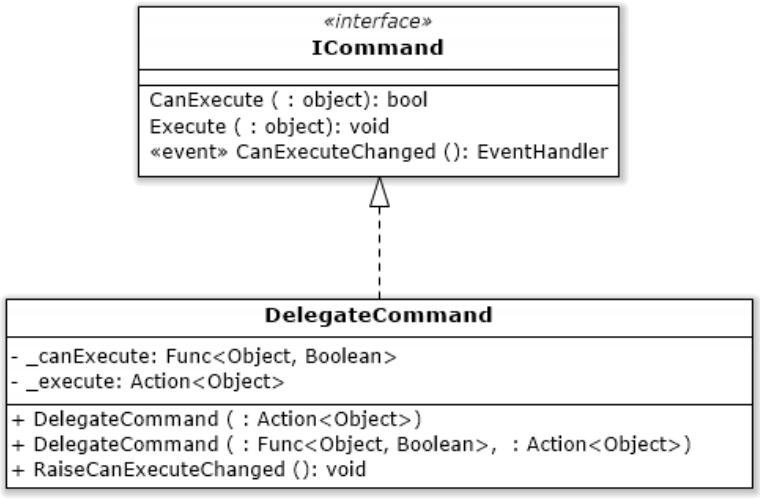
3.12. ábra. A `ViewModel` névtér

A `ViewModelBase` (3.13. ábra) osztály szerepe, hogy megvalósítsa a nézetmodell réteg azon osztályainak alapkövetelményét, amik valamilyen módon a nézet adatforrásaként funkcionálnak. Ebből származtatjuk a következő osztályokat: `BaseMapCell`, `SimulationSettings`, `OpenDialogItem`, `LogicalAgentsViewModel`. Az említett elvárás pedig az `INotifyPropertyChanged` interfész implementálása, ugyanis az itt megkövetelt esemény kiváltódása esetén automatikusan bekövetkezik a felület kapcsolódó részének frissítése.



3.13. ábra. A `ViewModelBase` osztály

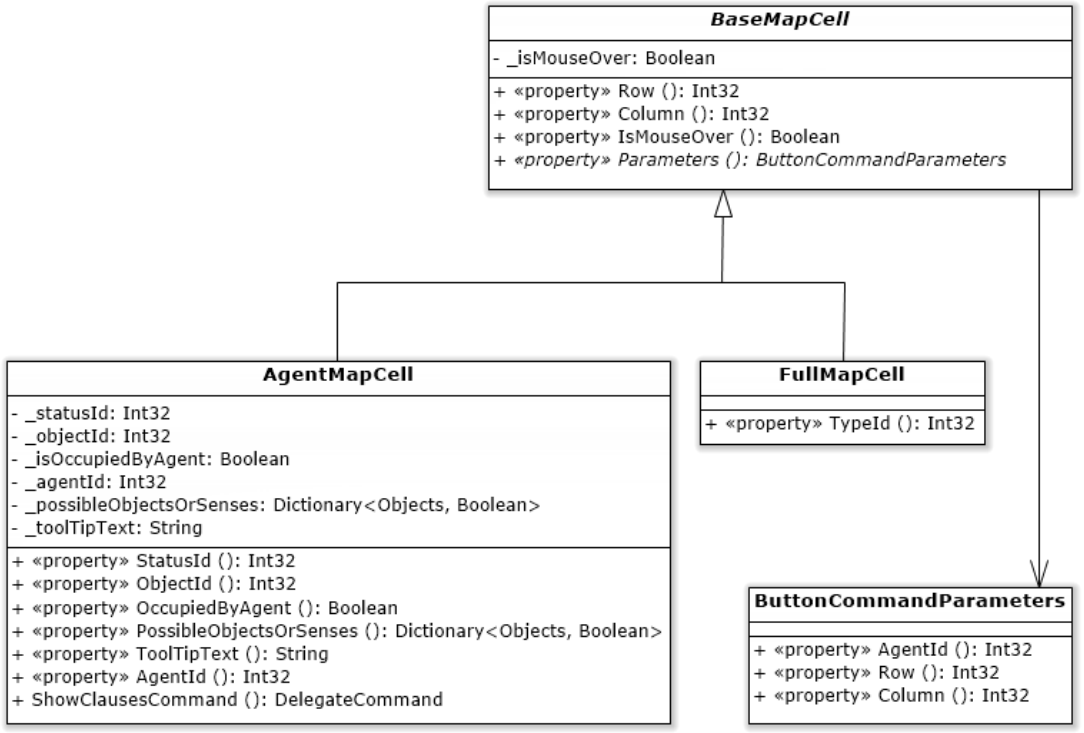
A nézettel való együttműködés másik alappillére, hogy az `ICommand` interfészt megvalósító parancsok a példányosítás után közvetlenül a felhasználói tevékenység által kiváltott eseményekhez rendelhetők, és azok bekövetkezésekor automatikusan végrehajtódnak, ezáltal a nézetbeli code-behind eseménykezelők mellőzhetők. Erre a célra hoztuk létre a `DelegateCommand` osztályt (3.14. ábra), mely a parancshoz társított tevékenységen (`_execute`) kívül annak végrehajthatóságával (`_canExecute`) is foglalkozik, így az ezt leíró logika a nézetmodell keretein belül lesz meghatározva. A végrehajtás feltételének elhagyása esetén a parancs mindig végrehajtódik.



3.14. ábra. A `DelegateCommand` osztály

A felület két oldalán elhelyezkedő térképek hasonlóságai és különbségei miatt a `BaseMapCell` osztály (3.15. ábra) biztosítja öröklődés segítségével, hogy a konkrétan megjelenítendő térképtől függetlenül az egyes cellák kezelni tudják a hozzájuk tartozó sor- és oszlopindexet (`Row`, `Column`), továbbá jelezni tudják, ha az egérmutató felettük található (`IsMouseOver`). Absztrakt tulajdonságként vettük fel a `Parameterst`, amit a származtatott osztályok eltérő módon implementálnak aszerint, mi tekinthető minimális információnak a cella beazonosíthatóságához. (A tulajdonságot az egérpozíció szerinti cella-kiemelésre használjuk fel.) A `ButtonCommandParameters` osztály példányaiban tárolható ez a minimális információ. Ennek felvételére azért volt szükség, mert a parancsok egyetlen paramétert képesek kezelni, ami akár összetett típus is lehet, a részfeladatban pedig néhány primitív típus együttes továbbítására volt szükség.

Mivel a teljes térkép nem változik meg egy szimuláció futása során, és az egérmozgáson túl felhasználói tevékenység sem köthető ahhoz a területhez, ezért a származtatott `FullMapCell` osztályt (3.15. ábra) elegendő csupán a cella anyagát kifejező `TypeId` tulajdonsággal kiegészíteni.



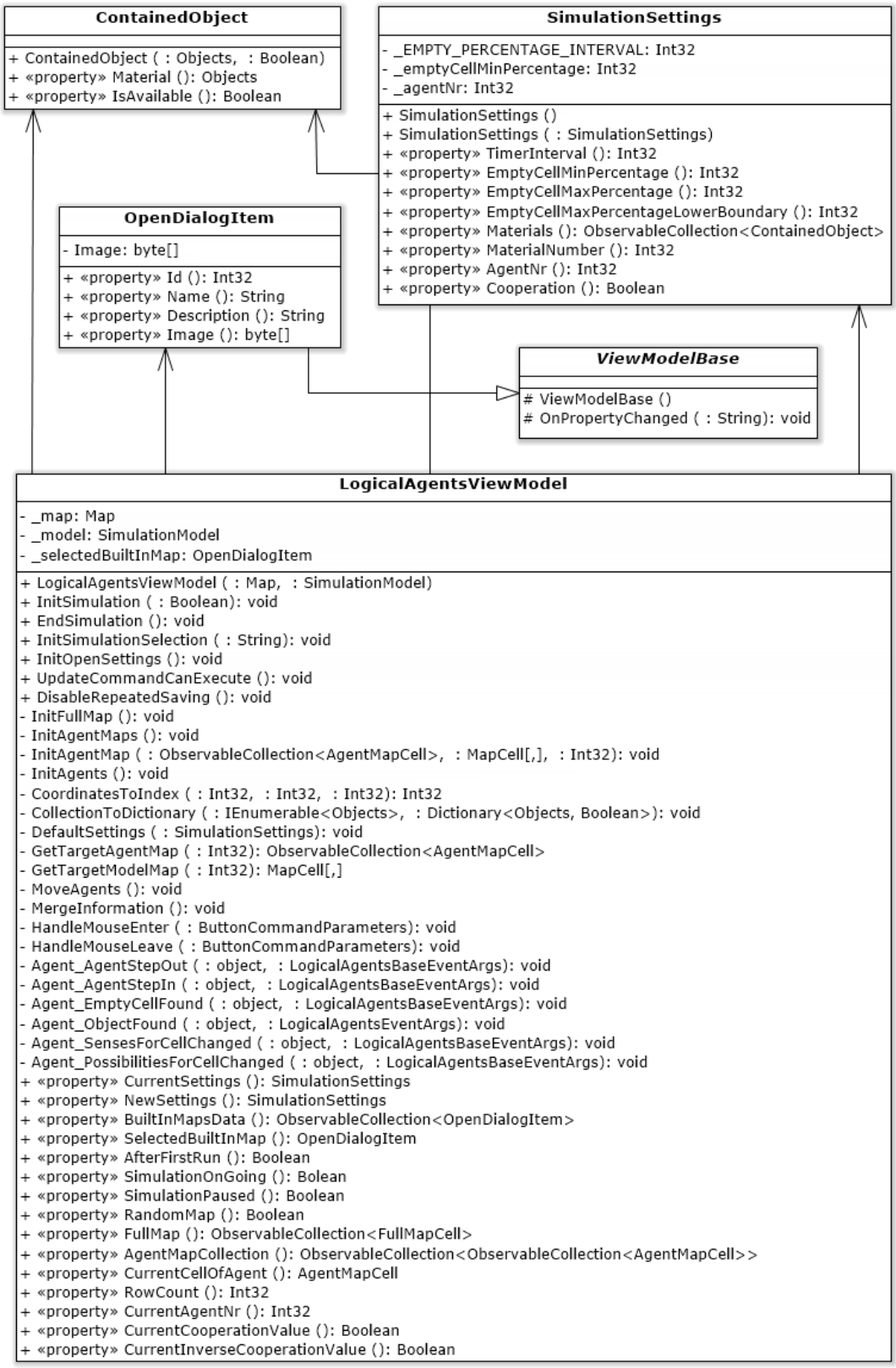
3.15. ábra. Az `AgentMapCell` osztály

Az ágensek ismeretei alapján megjelenített térképek lényegesen több funkcionalitással rendelkeznek, ebből adódóan a háttérben használt `AgentMapCell` osztály (3.15. ábra) is összetettebb. Mindenek előtt a tulajdonságoknak használni kell a `PropertyChanged` eseményt a bekövetkezett változások jelzésére. Kezelni kell az ágensek bejárása szerinti cellastátuszt (`StatusId`), a már felfedezett objektumok fajtáját (`ObjectId`), illetve azt is, jelenleg tartózkodik-e ott ágens (`OccupiedByAgent`). Számon kell tartani a lehetséges anyagokat és az érzékelők által jelzett anyagokat. Ez egyetlen tulajdonság segítségével lett megvalósítva (`PossibleObjectsOrSenses`), ugyanis a két eset kizárja egymást, de reprezentáció szempontjából nincs köztük különbség. Ráadásul ez teszi lehetővé, hogy a cellához rendelt tooltip szövegét is azonos módon konstruáljuk meg cellastátusz szerinti szétválasztás alapján.

A korábbi osztályokat felhasználva a teljes alkalmazás nézetmodelljeként a `LogicalAgentsViewModel` (3.16 ábra) osztály funkcionál. Tárolja a szimuláció térképét (`_map`) és a szimuláció modelljét is (`_model`). A konstruktor a szimuláció beállításait alaphelyzetbe állítja, példányosítja a parancsokat, és ahol szükséges, megadja hozzájuk a végrehajthatóság feltételeit is. A feltételek későbbi megváltozását jelezni kell (`RaiseCanExecuteChanged()`), ezáltal a parancskötés másik oldalán található nézetbeli vezérlő aktívvá vagy inaktívvá válik. A parancsok többségéhez tartozik egy hasonló elnevezésű esemény is, mely segítségével az alkalmazáskörnyezet tudo-

mást szerezhet az elvégzendő tevékenységről. Nem szükséges eseményt használni, ha a parancs alapján elvégzendő tevékenység csak a modellre van hatással, mert ez mindössze a modell alkalmas metódusának meghívásával is megoldható. A nézetmodell az alábbi parancsokat teszi elérhetővé:

- *<parancs neve>*: *<parancshoz kapcsolódó tevékenység>* (*<kiváltandó esemény neve, ha van>*)
- `PauseSimulationCommand`: a szimuláció megállítása és folytatása (`PauseSimulation`)
- `ManuallyStepSimulationCommand`: a szimuláció kézzel történő léptetése
- `JumpToTheEndCommand`: a futó szimuláció gyors lejátszása (`JumpToTheEnd`)
- `RestartSimulationCommand`: aktuális szimuláció újraindítása (`RestartSimulation`)
- `NewRandomSimulationCommand`: új szimuláció indítása véletlenszerűen generált térképen (`NewRandomSimulation`)
- `NewSimulationCommand`: új szimuláció indítása egy kisorsolt tárolt térképen (`NewSimulation`)
- `SelectNewSimulationCommand`: párbeszédablak megnyitása mentett térképek közüli választáshoz (`SelectNewSimulation`)
- `StartSimulationCommand`: új szimuláció indítása a párbeszédablakban kijelölt térképpel (`StartSimulation`)
- `SaveCurrentMapCommand`: az aktuális térkép elmentése (`SaveCurrentMap`)
- `OpenSettingsCommand`: a beállítások ablak megnyitása (`OpenSettings`)
- `SaveSettingsCommand`: az új beállítások elmentése (`SaveSettings`)
- `MergeInformationCommand`: ágensek közötti információ-megosztás végrehajtása
- `MouseEnterCommand`: az egérmutató egy cella fölé lett mozgatva
- `MouseLeaveCommand`: az egérmutató el lett mozgatva a celláról
- `ShowClausesCommand`: egy ágens térképén a megkattintott cella klózainak megmutatása (`ShowClausesForAgentMapCell`)



3.16. ábra. A `LogicalAgentsViewModel` osztály

Minden szimuláció indításakor az alsóbb rétegekre támaszkodva inicializálódnak a nézetbeli függőségi tulajdonságok forrásai. Az elrendezést az ágensek száma (`CurrentAgentNr`) és az együttműködésük (`CurrentCooperationValue`) határozza meg. Fel kell építeni a szimuláció térképét (`FullMap`), és létre kell hozni az ágensek ismereteit tárolni képes térképeket is (`AgentMapCollection`). A modellhez hasonlóan itt is egyetlen térkép jön létre, ha együttműködnek az ágensek, egyébként külön gyűjtemény (`ObservableCollection<AgentMapCell>`) kell ágensenként.

Az ágensek számának dinamikus változása miatt nem oldható meg már a konstruktorban, hogy az egyes példányok eseményeihez eseménykezelőket társítsunk, így ez is az inicializálás részévé válik. Egy ágens egy kilépési és egy belépési eseménnyel jelzi a mozgását (`Agent_AgentStepOut()`, `Agent_AgentStepIn()`), ez alapján kell a cellák `OccupiedByAgent` tulajdonságát karbantartani. Saját bejárásból adódóan illetve információ-megosztás következtében objektumok helyzetét határozhatja meg (`Agent_ObjectFound()`), és üres cellákat találhat (`Agent_EmptyCellFound()`). Külön esemény tartozik az érzékelőkhöz (`Agent_SensesForCellChanged()`) és a cellán elképzelhető anyagokhoz (`Agent_PossibilitiesForCellChanged()`). Fontos, hogy az eseménykezelőkben ne csak a cella státuszát és típusát állítsuk be, hanem a modelltől lekérve aktualizáljuk az érzeteket és a lehetőségeket is (`CollectionToDictionary()`). Itt a vizsgált lista vagy halmaz elemei alapján anyag-logikai érték párok jönnek létre, melyek minden egyes megváltozása a cella tooltip szövegének frissítését is megköveteli.

A fő nézetmodell kezeli a párbeszédablakokhoz tartozó adatkötések mögötti információkat is. A beállítások ablakhoz a `SimulationSettings` osztály egy példánya tartozik. A szimuláció minden beállítható paraméteréhez egy-egy tulajdonság kapcsolódik. A szabad cellák arányát kifejező intervallum minimális nagysága konstansként került megadásra (`_EMPTY_PERCENTAGE_INTERVAL`). Ennek betartására az alsó érték megváltozása szükségessé teszi a felső érték azonnali ellenőrzését, esetleges korrigálását. A lehetséges anyagok megadásához külön osztályt (`ContainedObject`) használunk, mely egyszerűbbé teszi a felülethez kötés megvalósítását. A beállítások módosításakor elképzelhető, hogy az értékek megváltoztatása (`NewSettings`) ellenére a felhasználó mégsem menti azokat, ezért a dialógusablak megnyitásakor egy független másolat készül (*copy constructor*), ami vagy felülírja a beállításokat, vagy elvetésre kerül, utóbbi esetben változatlanul rendelkezésre állnak a jelenlegi beállítások (`CurrentSettings`).

A térképkiválasztó ablak a beépített térképek listáját jeleníti meg. Az adatokat a perzisztencia rétegtől kell elkérni, de ezekből a modell `Map` osztálya előbb kiszűri a

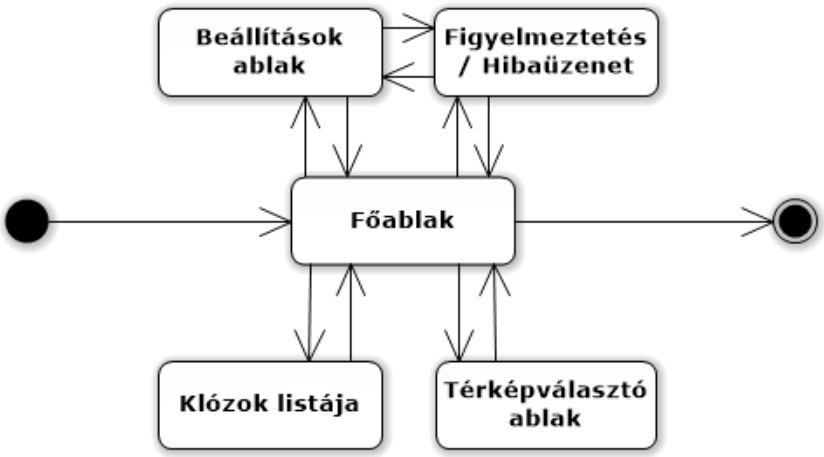
lényeges információkat, végül a nézetmodell teszi őket elérhetővé (`BuiltInMapsData`) az `OpenDialogItem` osztály segítségével. A megkapott információk közül az azonosítóból (`Id`) képezzük az elnevezést (`Name`), és a választott térkép (`SelectedBuiltInMap`) is ez alapján kerül beazonosításra a betöltéskor. A listában való kijelölés tudja megváltoztatni a betöltendő térképet, ezzel összhangban kell megjeleníteni a betekintő képet (`Image`), ezért itt is jelezni kell a változás eseményét.

3.4.4. A `View` névtér

Az MVVM architektúrából adódóan a nézet már csak korlátozott mértékben tartalmaz programkódot. A nézet egy ablakát két fájl segítségével lehet leírni, a *.xaml* kiterjesztésű szolgál a felület felépítésének leírására, a hozzá tartozó *.xaml.cs* fájlban az eseménykezelők írhatók meg. A két állomány együttesen valósítja meg az ablak parciális osztályát.

Esetünkben az alkalmazásban csak parancsokon keresztül történik a kommunikáció, ezért nincsenek code-behind eseménykezelők. Az ablakokat felépítő vezérlőkből előálló struktúra megadásától különválasztottuk az adatkötések és a megjelenés stílusának leírását, ehhez egy erőforrásfájlt használtunk fel (*StyleDictionary.xaml*). Így áttekinthetőbb kódot kaptunk, ráadásul a definiált stílusok egymásra építhetők, ezáltal az ismétlődő formázásokat is elég egyszer megadni.

Ha az alkalmazás ablakai alapján definiáljuk a program állapotait, akkor a 3.17. ábrán látható átmenetek lehetségesek:



3.17. ábra. Az alkalmazás állapotdiagramja

A továbbiakban a nézet létrehozásához használt bonyolultabb megoldások kerülnek bemutatásra.

Az ágensek térképeit két egymásba ágyazott `ItemsControl` vezérlővel jelenítjük meg. A forrásként (`ItemsSource`) megadott `ObservableCollection` gyűjtemények által a tartalmazott elemek darabszámához (vagyis egyrészt a térképekhez, másrészt az azokon belüli cellákhoz) automatikusan igazodik a nézet.

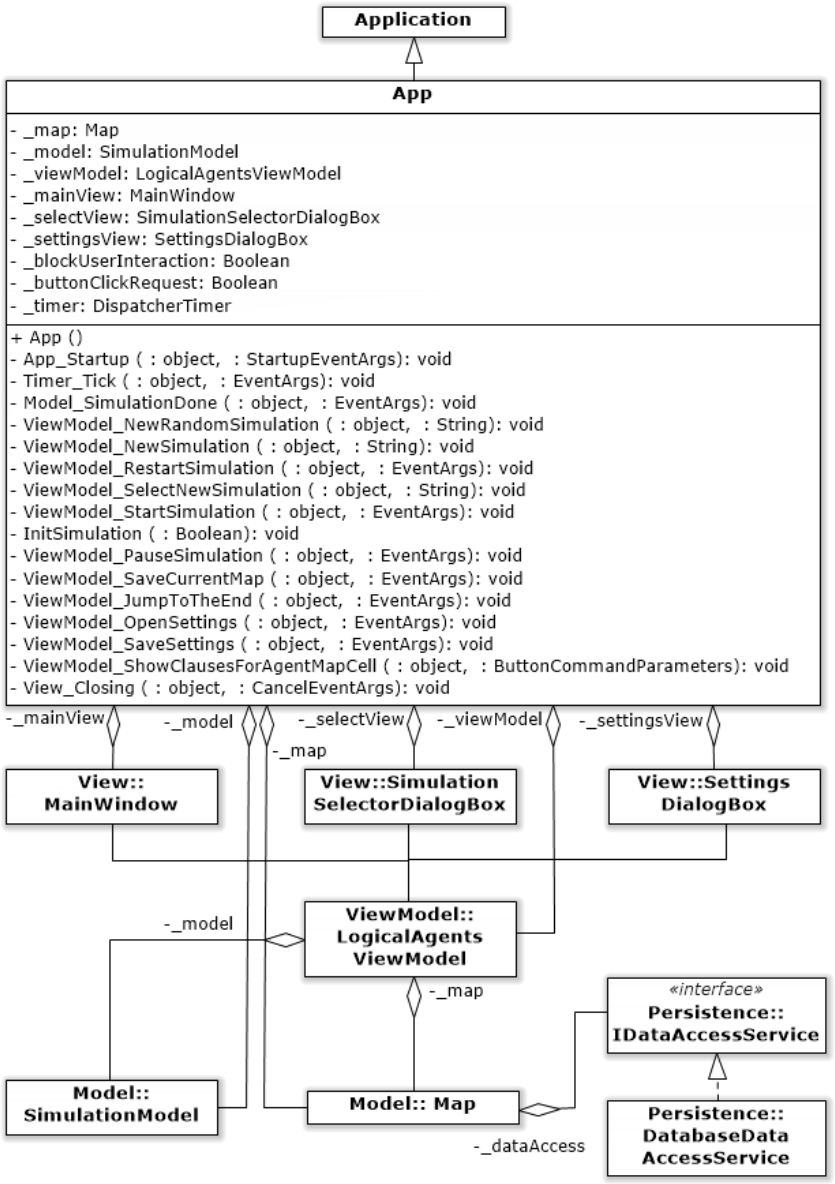
A belső elrendezés elemeihez gombokat használtunk. Az alapértelmezett gombstílus egérpozíció-függő összetevői nem írhatók felül közvetlenül, ezért saját sablonokat (`ControlTemplate`) készítettünk, ezek adják a cellák színét. A gombokon lehetőség van a háttér elé képet helyezni, így jelennek meg az anyagok ikonjai.

A felületet animációk (`Animation`, `Storyboard`) segítségével tettük látványosabbá. Ezek alapvető jellemzői a céltulajdonság (`TargetProperty`), a változás értékei (`From`, `To`), és az időtartam (`Duration`, `RepeatBehavior`). Animációt rendeltünk a tartalmazott anyagok kikövetkeztetéséhez (`Pulse`) és a gombon való kattintáshoz (`ScaleX`, `ScaleY`).

Az egérrel pillanatnyilag mutatott cella kiemeléséhez (lila keret) hozzáadtuk a projekthez az *Expression.Blend.Sdk* csomagot, ami elérhetővé teszi WPF-es alkalmazásokban a `System.Windows.Interactivity` névteret. Ennek segítségével lehetőség (`Interaction.Triggers`) van eseményekhez (`EventTrigger`) parancsvégrehajtást hozzárendelni (`InvokeCommandAction`). Esetünkben az egérmozgás eseményeit használtuk fel (`MouseEnter`, `MouseLeave`) a térképeken belüli azonos pozíciójú cellák együttes kijelöléséhez. Az itt használt parancskötés specialitása, hogy a parancs nem a gomb `DataContext` tulajdonságon keresztül érhető el, ami `AgentMapCell` típusú, hanem az alkalmazás nézetmodelljén keresztül. Emiatt módosítani kell a kötés jellemzőit. A `RelativeSource` tulajdonság segítségével az alapértelmezett forrást a hierarchiát tekintve egy szülő vezérlőtől (`Mode=FindAncestor`) vesszük át, erre a legkényelmesebb választás maga az alkalmazásablak (`Window`), ami biztosan a `LogicalAgentViewModel` tulajdonságait köti.

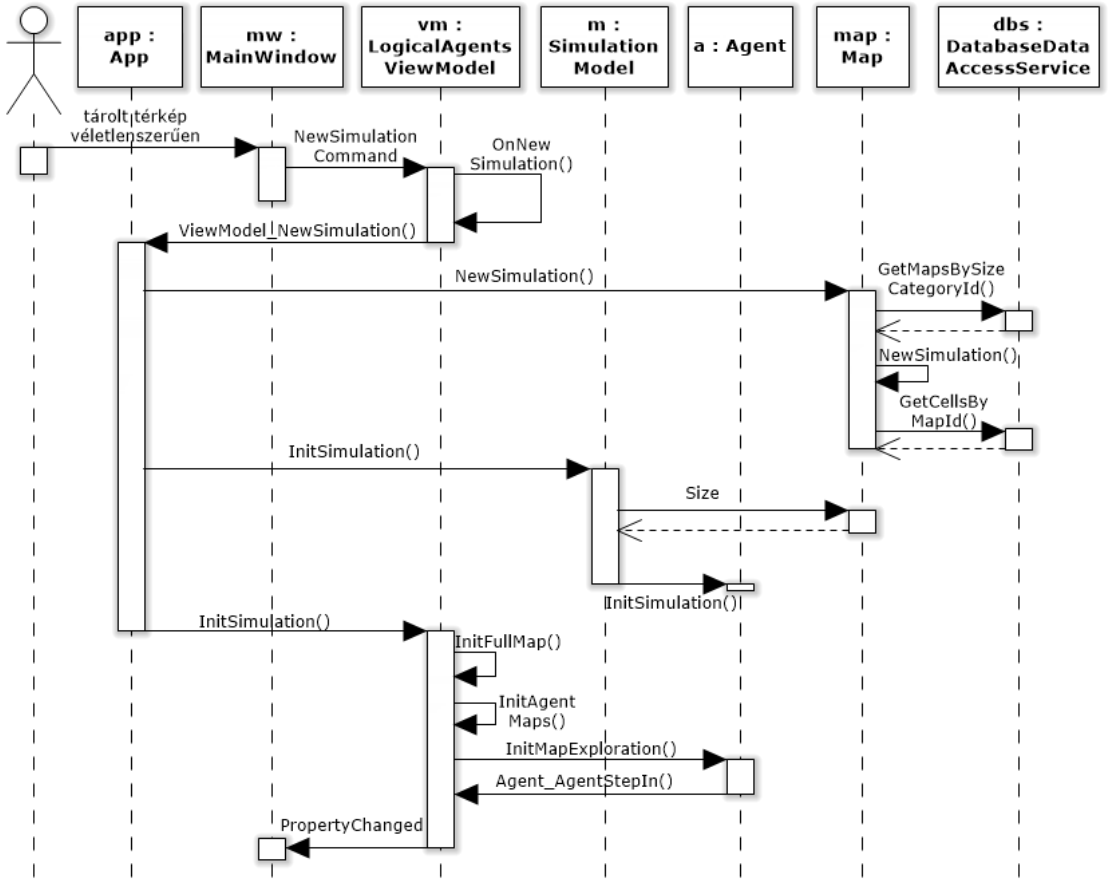
3.4.5. Az alkalmazáskörnyezet

A projektben az alkalmazáskörnyezetet az `App` osztály testesíti meg, amit az *App.xaml* és *App.xaml.cs* fájlok írnak le. A környezet alakítja ki az alkalmazásrétegek közti függőségi viszonyokat a program elindításakor (`App_Startup()`). Ezen felül a teljes alkalmazásra kiható eseményekhez is biztosít eseménykezelőket. Ilyen esemény többek között a szimuláció szüneteltetése, amikor az időzítőt kell kezelni, vagy egy új szimuláció indítása, amikor a főbb osztályokat az egymásra épülésük szerint alulról felfelé haladva inicializálni kell, vagy pedig a beállítások megnyitása, amikor az alkalmazás ablakai közti váltást kell megoldani. Az osztály metódusainak pontos listáját a 3.18. ábra mutatja a rétegek közti kapcsolatokkal együtt.



3.18. ábra. Az `App` osztály

A szimuláció tárolt térképen történő elindítása az alkalmazáskörnyzet által felépített hierachia minden rétegére hatást gyakorol:



3.19. ábra. Szekvenciadiagram tárolt térképen indított szimulációhoz

3.5. Implementációs megoldások

3.5.1. Térkép generálása véletlenszerű felépítéssel

A térképek sokféleségének biztosítására (ezáltal a szimuláció univerzálisabb bemutatására) készült eljárás, ami a szimulációnak kedvező módon véletlenszerű térképet generál.

Bemenő paramétere a térkép méret-kategóriája szövegesen megadva, ebből véletlenszerűen kerül meghatározásra az előállítandó térkép adatbázisban tárolt határértékek közti pontos mérete. Az algoritmus (lásd pszeudokód) a bal felső sarokból indul ki, azt a cellát garantáltan bejárhatóvá teszi (5. sor). A haladási irányok szerinti létező szomszédokból egy gyűjteményt képez (6-10. sor). Ezekből a lehetséges folytatási cellákból véletlenszerűen választ egyet (12. sor), majd annak létező szomszédaiból is egyet (18. sor). Ha ez utóbbi még nem felfedezett cella, mindkét kiválasztott bejárhatóként lesz megjelölve (20-21. sor), különben felfedezetlen marad a második, az első pedig objektumot fog jelenteni. A folytatási pontok gyűjteménye úgy módosul, hogy az ebben a lépésben kiválasztott törlődik (13. sor), az esetleges új szabad végcella felfedezetlen szomszédai hozzáadásra kerülnek (22-27. sor). A gyűjteményen alkalmazandó műveletek alapján a `LinkedList` generikus típusra ([8]) esett a választás, hiszen beszúrni elegendő az utolsó helyre, a ciklusmagonkénti törlés pedig ilyenkor átláncolással megoldható az elemek egyesével történő átmásolása helyett.

Az algoritmus lényegi része pszeudokód formájában:

```
1:  procedure GenerateMap
2:      for all i, j do
3:          map[i, j] ← UNKNOWN
4:      end for
5:      map[0, 0] ← EMPTY
6:      Empty(endPoints)
7:      for all posin AdjPositions(0, 0) do
8:          map[pos.row, pos.column] ← OBJECT
9:          Add(endPoints, pos)
10:     end for
11:     while Size(endPoints) > 0 do
12:         first ← RandomElement(endPoints)
```

```
13:         REMOVE(endPoints, first)
14:         EMPTY(choices)
15:         for all posinADJPOSITIONS(first.row, first.column) do
16:             ADD(choices, pos)
17:         end for
18:         second ← RANDOMELEMENT(choices)
19:         if map[second.row, second.column] = UNKNOWN then
20:             map[first.row, first.column] ← EMPTY
21:             map[second.row, second.column] ← EMPTY
22:             for all posinADJPOSITIONS(second.row, second.column) do
23:                 if map[pos.row, pos.column] = UNKNOWN then
24:                     map[pos.row, pos.column] ← OBJECT
25:                     ADD(endPoints, pos)
26:                 end if
27:             end for
28:         end if
29:     end while
30: end procedure
```

A szabad cellák kettesével bővülése a tömbösödés megakadályozására, a szabad rész járatszerűvé válására szolgál, ahol a bővítés bekövetkezésének valószínűsége is az ismeretlen cellák irányába nagyobb.

Az algoritmus által érintetlenül hagyott cellák végül az objektumok halmazát bővítik, így az algoritmus garantáltan egy bal felső sarokból kiinduló, összefüggő bejárható részt eredményez. Az ellenben nincs biztosítva, hogy minden objektum rendelkezzen szabad szomszédos cellával, ezek a szabad celláról nem érzékelhető, más objektumok által teljesen körülzárt pontok felfedezetlenek maradnak a bejárás végeztével.

Az alap algoritmus futási idejét vizsgálva kijelenthető, hogy optimális esetben a szabad cellák p arányát tekintve $p\frac{n^2}{2}$ iterációs lépés elegendő. Az általános eset bonyolult valószínűségszámítási elemzést igényel, mivel a járatbővítés sikeressége adott ponton függ a térkép méretétől, az algoritmus előrehaladottságától (mennyi felfedezetlen cella van még) és a pozíciótól is (széleken kevesebb a lehetséges szomszéd).

Az alkalmazás során a döntő tényező azonban az, hogy a keretfeltételeknek való megfelelési kényszer miatt hányszor kell újrakezdeni a generálás ciklusát. Kijelenthető, hogy minél nagyobb a szabad cellák aránya, annál több próbálkozás kell. A

sarkok közelébe való eljutás, vagyis a terjedés irányának megkötése is csökkenti egy generálás sikerességének valószínűségét.

A sarokhoz közeli szabad cella biztosítása kapcsán három megközelítés merült fel. A legkevesebb változtatást az igényelte volna, ha a generálás során számon lett volna tartva, hogy sikerült-e mind a négy sarokba eljutni a szabad cellák hozzáadásával. Ez azonban olyan szigorú megkötés, aminek bekövetkezési valószínűsége a véletlenszerű generálást megtartva csekély. Megoldható lenne az is, hogy az algoritmus a négy sarok mindegyikét kiindulópontnak tekintse, ekkor az összefüggőség lenne még külön ellenőrizendő. Ez hat cella-pár közti (tranzitivitás miatt csökkenthető) útkeresési feladatot jelentene, úgyhogy nem is ez a lehetőség lett kiválasztva. Végül a hatékonyság, a rugalmasság érdekében enyhítve lett az elvárás, a sarok helyett végtelen normában tekintett egy vagy két távolságra levő szabad cella keresése által. Ez térképmérettől függetlenül néhány lépéses lineáris keresés keretében megvalósítható.

3.5.2. Térkép bejárása

Fontos megemlíteni, hogy a térkép szabályos alakja miatt nem gráfként kerül reprezentálásra (például csúcsmátrix vagy éllista formájában), hanem kétdimenziós tömbként, ahol az indexek magukban hordozzák az információt, milyen irányok mentén léteznek szomszédok, és azokra hogyan lehet hivatkozni.

A megvalósítás további sajátossága, hogy az ágens kezdetben nem rendelkezik a térkép alakjával, csak egy annak megfelelő tárolóhellyel, ahová lépésenként folyamatosan jegyzi fel a felfedezés információit. A külön csúcsszínezés ([4]) kiváltható a térkép nyújtotta publikus információk, a saját felfedezés adatai és az ágensek közötti információ-megosztás segítségével. Ezek alapján meghatározható, mikor szükséges a visszalépés, továbbá ez eredményezi, hogy több együttműködő ágens diszjunkt módon osztja fel a szabad cellákat egymás között. Noha a bejárással egy cellán többször is áthalad az ágens, vagyis új információk szerzése szempontjából felesleges lépéseket is tesz, képessé válik magától megállapítani, mikor végzett az elérhető rész feltérképezésével. A visszalépési pontok ráadásul azt is jelentik, hogy az adott cella teljes környezete onnan nézve felfedezésre kerül, és ez alkalmat ad megkísérelni következtetéseket levonni.

3.5.3. Rezolúció

A dolgozat korábbi részeiben szó esett a rezolúció elméleti hátteréről, a `Literal` és a `Clause` osztályokról, valamint a `Resolution` statikus osztály által nyújtott műveletekről. Ebben a szakaszban ezen információkra támaszkodva a rezolúció feladathoz igazított implementációja kerül bemutatásra.

Klózok megalkotására az alábbi helyzetekben kerül sor:

Ha az ágens belép egy cellára, vagyis az egy szabad cella, ezt a tudást $\neg O(x, y) \quad (O \in \{fa, kő, vas, arany\}, \quad x, y \in [0, n-1])$ alakú klózok felvétele által rögzíti.

Ehhez a lépéshez tartozik továbbá, hogy az érzékelők alapján is megfelelő klózok keletkezzenek. Minden érzékelt anyaghoz egy diszjunkció jön létre, amikben a literálok száma a nem szabad szomszédos cellák számával egyezik meg, és amik kizárólag a hivatkozott anyagban különböznek. Pl.: $fa(4,1) \vee fa(3,2)$, $vas(4,1) \vee vas(3,2)$ (3.20. ábra). Egy darab ilyen klóz-objektum lesz példányosítva, és a kapcsolódó cellák klózhalmazai erre a példányra fognak mutatni.

3.20. ábra. Példa szituáció

Másik fontos szituáció, ha az ágens egy lépése által bebizonyosodik, hogy egy cellán valamilyen objektum található. Ez esetben az eddigi bejárás során pontosított `PossibleObjects` halmaz alapján keletkeznek a klózok, az összes nem lehetséges anyagból – egy üres cellához hasonlóan – egy negált literálból álló klóz lesz, a lehetséges anyagokból pedig csak az adott cellára vonatkozó diszjunkció jön létre, pl.: $fa(3,2) \vee vas(3,2)$.

Ha a rezolúcióval levezethető az üres klóz, az azt jelenti, hogy az adott cellára kikövetkeztethető volt, a lehetséges anyagok melyike található ott. Ezt a tudást is rögzíteni kell klózok formájában. A kikövetkeztetett anyag miatt az előző példát folytatva $vas(3,2)$ keletkezhetne, és ez együtt jár a $\neg fa(3,2)$ hozzáadásával is.

A következtetés sikerességének fontos tényezője, hogy az ágens ne csak megalkotni tudjon klózokat, hanem azokat meg is változtassa, ha olyan információ bir-

tokába jut, ami ezt indokolttá teszi. A megoldott feladatban ez akkor fordul elő, ha egy üres cellára belépve található ott nem negált literált tartalmazó klóz, hiszen mindez azt jelenti, hogy az érzékelés pillanatában megfogalmazott feltételezés erre az üres cellára már nem érvényes. Belátható, hogy a módosítandó klózok nem lehetnek egységklózok, mivel ez akkor fordulna elő, hogy az ágens érzékel valamit, ami csak egyetlen szomszédos cellán lehet, de az a szomszédos cella üres, ez viszont ellentmondás. Így az érintett klózok legalább két literált tartalmaznak, ami azt jelenti, nem csak az üres cellára vonatkoznak, ezért teljesen törölni nem szabad őket, hanem az érvénytelen literálok törlése a feladat. A klózok mint objektumok tárolási módjából adódik, hogy az üres cellából hozzáférve a példányhoz a módosítás minden kapcsolódó cellában érzékelhető lesz.

A klózokat tároló `HashSet<T>` generikus típus ([9]) választásának oka, hogy minden beszúrást egy tartalmazásra vonatkozó keresés előz meg, amit a beszúrás műveletigénye szempontjából figyelembe kell venni. A `HashSet` az elemeket a hash-értékük szerinti kosarakban tárolja, így a keresés $\mathcal{O}(1)$ idejűvé válik. Ráadásul az érvénytelen literálok törlése duplikátumokat okozhat, amik törlésére is fel kell készülni, ez szintén egy konstans idejű művelet.

A rezolúciót a `ResolutionMethod()` függvény vezérli, ami a rezolvens-képzésnél a `SearchForResolvents()` metódusra támaszkodik. A kiindulási klózhalmazt szisztematikusan vizsgálja át, de próbálja a felesleges összehasonlításokat kizárni (lásd pszeudokód). Inicializáló lépésként a kiindulási halmazt önmagával veti össze, a keletkező rezolvenseket külön gyűjti (5. sor). Innentől iterációba szervezve (6. sor) hívódik a `SearchForResolvents()` függvény (8. sor). A hatékonyabb számítás érdekében arra a mindig igaz tulajdonságra alapoz, hogy az első aktuális paraméter halmazából már nem lehet új rezolvenst képezni, így csak az elsőt a másodikkal kell összehasonlítani (16-17. sor), ugyanis a második halmazba nem fog negált literált tartalmazó klóz kerülni. A rezolvensek tárolására a harmadik aktuális paraméter szolgál (26. sor). A függvény azonnal igazzal tér vissza, ha levezethető volt az üres klóz (24. sor), hamis értéket vesz fel, ha elfogynak az összehasonlítandó klózok (35. sor). A függvényhívás után az említett tulajdonság fenntartásához az előbbi felparaméterezés szerinti első két halmaz összevonásra kerül (10. sor), és a következő lépésben az újonnan keletkezett rezolvensek halmazával lesz összehasonlítva (11. sor).

A klózok megalkotásából adódik, hogy negált literál csak egységklóz formájában fordulhat elő. (Ebből következik, hogy a rezolvens-képzés nem hoz létre olyan klózt, amiben negált literál található.) Erre a megfigyelésre építve a komplemens literálpár

keresésének négyzetes műveletigénye egy összehasonlításra redukálható (18. sor), amennyiben a két klóz közül nem pontosan egy tartalmaz negált literált. Ráadásul a klózok ezen speciális alakja miatt azt sem kell külön ellenőrizni, hogy pontosan egy komplemens literálpárt tartalmaznak-e, tehát ha komplemens literálpárt találnuk, azonnal lehet rezolválni (22. sor), és ezzel a két klóz vizsgálata véget is ért (27. sor).

Az algoritmus lényegi része pszeudokód formájában:

```
 1: function Resolution(clauses)
 2:     set1 ← clauses
 3:     Empty(set2)
 4:     Empty(set3)
 5:     emptyClause ← SFR(set1, set1, set2)
 6:     while ¬emptyClause and Count(set2) > 0 do
 7:         Empty(set3)
 8:         result ← SFR(set1, set2, set3)
 9:         emptyClause ← emptyClause or result
10:         set1 ← Union(set1, set2)
11:         set2 ← set3
12:     end while
13:     return emptyClause
14: end function

15: function SFR(set1, set2, target)
16:     for all clause1 in set1 do
17:         for all clause2 in set2 do
18:             if ContainsNeg(clause1) ≠ ContainsNeg(clause2) then
19:                 for all literal1 in clause1 do
20:                     for all literal2 in clause2 do
21:                         if Complementary(lliteral1, lliteral2) then
22:                             resolvent ← Compute(clause1, clause2)
23:                             if EmptyClause(resolvent) then
24:                                 return true
25:                             else
26:                                 Add(target, resolvent)
27:                                 break
28:                             end if
29:                         end if
30:                     end for
31:                 end for
```

```
32:             end if
33:         end for
34:     end for
35:     return false
36: end function
```

A rezolúció számításigénye miatt nem lenne optimális az algoritmus túl gyakori végrehajtása, ezért az ágensek jól meghatározott helyeken próbálkoznak csak a következtetéssel:

- minden objektum első felfedezésekor
- egy objektum későbbi érzékelésekor
- a bejárás szerinti visszalépéskor (minden szomszédos még ki nem következtetett objektumra)
- a bejárás végén az összes még ismeretlen objektumra (sőt, ezt iterációba szervezve, hiszen amíg sikerül következtetni, elképzelhető, hogy az új információ másik következtetés sikerességét eredményezi)

3.5.4. Ágensek közötti információ-megosztás

A feladat különlegessége abban rejlik, hogy az ágensek által gyűjtött, akár több lépésben finomított információkból kiszűrjük a hasznosakat. Általánosságban ez annyit jelent, hogy egy adat mikor értékesebb egy másiknál, de a klózokat vizsgálva olyan kérdések is felmerülnek, mint hogy van-e köztük ellentmondás, mikor állít többet egy klóz a másiknál, részhalmazi viszonyban vannak-e, vagy legfeljebb nemüres metszettel rendelkeznek.

Megoldásként az ágensek által használt módszerből indultunk ki: minden egyes klóznak és a literáljainak később meg lehet a szerepe, kivéve az üres cellákra nyilvánvalóan lehetetlen objektum-tartalmazást feltételezőknek. Emiatt a klózok kérdése nem folyamatosan kiértékelendő feladat, hanem előfeltétele, hogy az összes ismeret alapján minden cellához státusz legyen rendelve, mely folyamat során az előforduló klózokat egyetlen halmazban egyesítjük. Innen eliminálásra kerülnek az előbb említett érvénytelen literálok, klózok. Annak érdekében, hogy minél több információt

lehessen visszajuttatni az ágenseknek, minden még kérdéses cella esetében alkalmazzuk a rezolúciót, ráadásul így egy helyen kell csak futtatni az eljárást, amit az ágensek később külön-külön úgyis megtennének.

Az így kapott eredményt az ágensek tudásának eredőjeként kezeljük, amiről továbbfelhasználásra alkalmas független másolatokat kell készíteni, tehát mintha egy ágens maga rögzítette volna ezt az összes adatot. Ez úgy oldható meg, ha minden egyes klózról mély másolat (*deep copy*) készül, ami az összes literáljában meghivatkozott cella klózai közé bekerül.

Érdekesség, hogy a cellákban tárolt információk eredője más-más módon számítható ki. Például a cellastátusz az azt leíró felsorolási típus egésszé konvertált értékének maximumaként kapható meg, az érzékelt és a kikövetkeztetett anyagok teljes mértékben átvehetők, ha még nincs erre vonatkozó ismeret, a lehetséges anyagok egyfajta metszetképzésből adódnak, végül a klózok esetében a fent kifejtett kumulatív módszer volt célravezető.

3.6. Tesztelés

3.6.1. Unit tesztek

Az alkalmazás tesztelésére a modell réteg `Map` és `Agent` osztályaihoz egységteszteket készítettünk, mely fehérdoboz tesztelésnek tekinthető, ugyanis a programkód alapján fogalmaztuk meg az objektumok megváltozására vonatkozó ellenőrzéseket. Ennek érdekében hozzáadtuk a projekthez a *Moq* csomagot, mely mock objektumok létrehozásában nyújt segítséget.

A `Map` osztály `IDataAccessService` függőségének feloldásához az adatbázist is helyettesíteni kellett. Megadtuk a tesztelésnél használni kívánt tesztadatokat (pl. `_cellData`), majd ezeket felhasználva elkészültek az adatbázistáblák mock objektumai (pl. `_cellMock`). Az adatbázis-elérés mockolásához leválasztottuk annak interfészét (`ILogicalAgentsEntities`), és beállítottuk (`Setup`), mi legyen a visszaadott eredmény a különböző kérésekre. Mindezzel a `DatabaseDataAccessService` osztály példányosíthatóvá vált a konkrét adatbázis használata nélkül (`_entityMock.Object`).

A térkép egységtesztjében két területre fókuszáltunk. Egyrészt ellenőriztük, hogy az adatbázis-szerkezetnek megfelelően tárolt adatok helyesen képeződnek-e le a modellbeli osztályokra. Ehhez a tesztadatok alapján felépítettük a szimuláció térképét, majd cellánként összehasonlítottuk azok anyagát az elvárt értékekkel. Másrészt az ágensek felé biztosított publikus metódusokat teszteltük. Ez a térkép határainak ellenőrzését, a szabad és objektumokat tartalmazó cellák elkülönítését és az érzékelők jelzéseinek vizsgálatát jelentette.

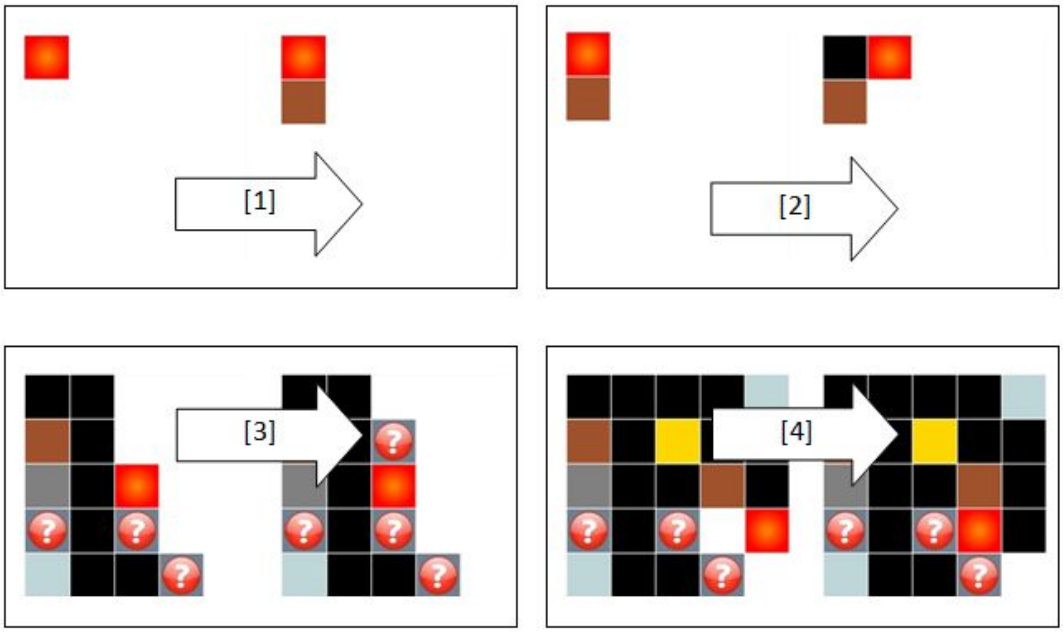
A térkép sikeres tesztjei alapján úgy döntöttünk, az ágens tesztelésénél arra alapozunk, hogy a térkép működése helyes. Az ágens vizsgálatánál az összegyűjtött információk időbeli megváltozásaira összpontosítottunk, megfogalmaztuk az elvi elvárásokat és leellenőriztük azokat a teszttérképen való futtatással. A teszttérképet úgy alkottuk meg, hogy a bejárásával az ágens minél több különböző szituációba (3.21. ábra) kerüljön, amiket tesztesetként felhasználhatunk:

[1] : az ágens (1,1) felfedezi a tőle lefelé (2,1) található objektumot (a táblázat a (2,1) cellát jellemzi)

[2] : az ágens (1,1) átlép a jobbra (1,2) található szabad cellára (a táblázat az (1,2) cellát jellemzi)

[3] : az ágens (3,3) felfedezi a tőle felfele (2,3) található objektumot (a táblázat a (2,3) cellát jellemzi)

[4] : az ágens (4,5) átlép a jobbra (4,4) található szabad cellára (a táblázat a (4,3) cellát jellemzi)



3.21. ábra. Az ágens pozíciója a tesztesetekben

Kép	Status		PossObj		Clauses		DeducedObject	
	régi	új	régi	új	régi	új	régi	új
[1]	UNKNOWN	OBJECT	1	1	1	5	NOTHING	WOOD
[2]	UNKNOWN	EMPTY	1	0	1	4	NOTHING	NOTHING
[3]	UNKNOWN	OBJECT	2	2	4	7	NOTHING	NOTHING
[4]	OBJECT	OBJECT	2	2	9	11	NOTHING	NOTHING

3.1. táblázat. Tesztesetek

A 3.1. táblázat egy-egy sorában az látható, hogy a képeken illusztrált szituációkban milyen értékeket vettek fel (*régi*) a vizsgált cella tulajdonságai az ágens lépését megelőzően, majd azt követően hogyan módosultak (*új*). (A *PossObj* oszlopban a `PossibleObjects` tulajdonság mögötti, a *Clauses* oszlopban a `Clauses` tulajdonság mögötti gyűjtemény elemszáma lett felüntetve.) Az itt szereplő tulajdonságoknak és azok értékeinek ellenőrzésén túl az egységteszt a szituációra jellemző módon a már meglévő és az újonnan keletkező klózok szerkezetét is külön vizsgálta.

Ezen kitüntetett pontok közti időben a lehetséges anyagok halmazának változásait is megfigyeltük két tipikus helyzetben. Egyrészt, amikor az objektum másik

szögből való megközelítése nem szűkíti a lehetőségek körét, másrészt, amikor viszont csökken a lehetséges anyagok száma.

A bejárás utolsó lépését is felhasználtuk a teszteléshez. A befejezés előtt megkísérelt rezolúció által kifejtett hatásokat vizsgáltuk:

- Ha a cellához tartozó kikövetkeztetett anyag megváltozik, akkor az utolsó lépés előtt csak ismeretlen lehet.
- Az ilyen sikeres következtetés csak a cellához tartozó klózok számának növekedésével együtt történhet, ráadásul minden új klóz egyetlen negált literálból áll.
- A szimuláció végén nincsenek duplikált klózok.
- Minden üres cellára igaz, hogy az ottani érzetek mindegyike valamely szomszédos objektumot tartalmazó cella kikövetkeztetett anyaga (ugyanis a teszttérkép teljesen felfedezhető).

3.6.2. Tesztelés a fejlesztés során

A fejlesztés során a kód egyes részein elhelyeztünk extra ellenőrzéseket szintén egyfajta fehérdoboz tesztelés gyanánt. Ezek az ellenőrzések a végső verzióban nem maradtak benne, egyrészt, mivel megvalósításuk műveletigénye jelentős, másrészt olyan szituációkra vonatkoznak, melyek a tervezési szándék szerint nem következhetnek be, és a százas nagyságrendű véletlenszerű szimuláció ezt be is igazolta, amennyire lehetséges. A vizsgált szempontok:

- Egyetlen anyag érzékelése után történő objektum-felfedezéshez meghívott rezolúció mindig sikeres.
- Több lehetséges anyag esetén csak legfeljebb egy következtethető ki rezolúcióval.
- Ha egy cellát sikerül kikövetkeztetni, akkor onnantól a `DeducedObject` tulajdonsága változatlan marad.
- A szimuláció végén nincs téves következtetés a használt térképhez képest.

- Nem fordul elő, hogy az érzetek száma nagyobb, mint a lehetséges objektumot tartalmazó szomszédos cellák száma.
- Ha egy objektum csak egyetlen szabad celláról érzékelhető, és ott az érzékelők több különböző anyagot is jeleznek, akkor a szimuláció végén az objektum kikövetkeztetetlen marad.
- Az ágensek közti információ-megosztás ellentmondásmentes adatokon történik.
  - egy cella nem lehet egyszerre szabad és objektumot tartalmazó
  - szabad cellához tartozó, különböző ágensek által érzékelt anyagok halmaza nem különbözhet
  - adott cella esetén a lehetséges anyagok halmazai vagy egymás részhalmazai, vagy megegyeznek
  - egy cellára két ágens nem következtethet ki különböző anyagokat

3.6.3. Felhasználói esetek szerinti tesztelés

A felhasználói esetek tesztelése egy integrációs teszt volt, a programegységek együttműködése adta eredményt vizsgáltuk feketedoboz megközelítéssel.

- Szimuláció inicializálása
  - Minden adatbázisbeli térkép látható és elindítható.
  - A tárolt térképek méret szerint két diszjunkt halmazban érhetők el.
  - Csak a mellékelt térképeknek van valós betekintő képe.
  - A méretkategória szerinti választás csak megengedett méretű térképet ad.
  - Random-generált térképek csak összefüggő bejárható résszel keletkeznek.
  - A random-generált térképek biztosítják a négy különböző kezdőpozíciót.
- Beállítások kezelése
  - A szabad cellák aránya érvényre jut, szemmel látható mértékkel csökkenthető, szigorú esetben a generálás ellehetetleníthető.
  - A tizenöt százalékos különbség nem kerülhető ki.
  - A be nem jelölt anyagok nem szerepelnek a kapott térképen.

  - Az ágensek darabszáma és az együttműködési beállítása a megfelelő elrendezést eredményezi.
  - A sebesség megadása csak a felkínált keretek között lehetséges.

- Szimuláció vezérlése
  - Az ágensek a kezdőpozíciójukra térnek vissza.
  - Menteni csak random-generált térképet lehet.
  - Sikeres mentést követően a mentés funkció rögtön elérhetetlenné válik az adott térképen, nem lehet többszörös mentést készíteni.
  - Az újraindítás megtartja a jelenlegi térképet.
  - A szimuláció megállítása, folytatása, befejeződése módosítja a gombok kattinthatóságát.
  - A gyorsbillentyűk működnek, azonban ha a megegyező gomb inaktív, a billentyű-kombinációknak sincs hatása.

3.6.4. Értékelés, megjegyzések

A fentiekben felsorolt szempontokkal, tesztesetekkel szemben támasztott elvárások minden esetben maradéktalanul teljesültek.

A tesztelés nem tartalmaz skálázhatóságra, terhelésre vonatkozó részeket, mivel az ezt befolyásolni képes paraméterek korlátosak. A használható térképek mérete nem haladja meg a 100 cellát. A rezolúció egy cella meghatározott környezetéből gyűjt össze klózokat, melyek számosságát behatárolja a szomszédos cellák és a szimulációban kezelt anyagok száma. A beállítások legfeljebb négy ágens használatát engedélyezik, így a köztük történő információ-megosztás adatmennyisége sem lehet tetszőlegesen nagy.

Hatékonysági szempontok ellenben több helyen megfontolásra kerültek, ezek részletezése és a választott megoldások indoklása a dolgozat implementációt kifejtő részében található.

3.7. Továbbfejlesztési lehetőségek

Noha a megvalósítás során minél átgondoltabb implementációs döntések és megoldási módszerek használatára, valamint ezek minél tökéletesebb kivitelezésére törekedtem, a programmal kapcsolatban több továbbfejlesztési lehetőség is felmerült:

- az implementált rezolúciós eljárás hatékonyságának javítása
  - a klózok differenciáltabb tárolása
  - csak a következtetés szempontjából releváns klózok halmazba gyűjtése
- a rezolúció alkalmazásának gyakorisága
  - a környezet alapján heurisztikák segítségével eldönteni, megéri-e meghívni az eljárást
- alternatív bejárási stratégiák
  - meghatározott cél-anyag esetén az érdektelen területek részletes feltérképezésének elhagyása
  - részeredmény-, rezolúció-orientált mozgás egy objektum vagy objektumcsoport minél gyorsabb kikövetkeztetése érdekében
- a szimuláció lépéseinek tárolása, teljes oda-vissza léptetési lehetőség
- az adatbázis-elérés webszolgáltatásként való megvalósítása

Irodalomjegyzék

[1] Stuart Russell, Peter Norvig: Artificial Intelligence: A Modern Approach, Pearson, 2009, [1152], ISBN-013-604-259-7

[2] Pásztorné Varga Katalin, Várterész Magda: A matematikai logika alkalmazásszemléletű tárgyalása, Panem, 2003, [394], ISBN-963-545-364-7.

[3] Josh Smith: Advanced MVVM, 2013, [50], ISBN-098-578-454-7.

[4] Thomas H. Cormen: Introduction to algorithms, MIT Press, 2001, [1180], ISBN-026-203-293-7.

[5] Microsoft .NET keretrendszer letöltése, http://www.microsoft.com/en-us/download, 2016. május 7.

[6] Microsoft SQL Server 2014 Express és SQL Server Management Studio letöltése, http://www.microsoft.com/en-us/download/details.aspx?id=42299, 2016. május 7.

[7] The MVVM Pattern, http://msdn.microsoft.com/en-us/library/hh848246.aspx, 2016. április 23.

[8] A LinkedList osztály, http://msdn.microsoft.com/en-us/library/he2s3bh7(v=vs.110).aspx, 2016. április 30.

[9] A HashSet osztály, http://msdn.microsoft.com/en-us/library/bb359438(v=vs.110).aspx, 2016. április 30.