

TDK DOLGOZAT

Kiss Péter



Nyugat-magyarországi Egyetem

Simonyi Károly Műszaki, Faanyagtudományi és
Művészeti Kar

Informatikai és Gazdasági Intézet

Virtuális gazdasági környezet szimulátor

Szerző: Kiss Péter – GAIN I. évfolyam – kissp@gain.nyme.hu

Konzulens: Bencsik Gergely – PhD hallgató – bencsikg@inf.nyme.hu

2014

Tartalomjegyzék

1.	Bevezetés.....	1
2.	A feladatról és az elérendő célokról.....	3
2.1.	Célok.....	3
2.2.	Játékosítás (Gamification).....	4
2.3.	Létező elgondolások, kapcsolódó ötletek.....	5
3.	Rendszerterv.....	7
3.1.	Szimulátor (mag).....	7
3.2.	Adatbázis.....	7
3.3.	Webes felület.....	8
3.4.	Szintek közötti kommunikáció.....	8
4.	Tervezési szakasz.....	9
4.1.	Ellátási lánc a tervezett rendszerben.....	9
4.2.	Entitások.....	9
4.3.	Folyamatok.....	11
4.4.	Alapvető versenyhelyzet.....	12
4.5.	IT specifikáció.....	12
4.5.1.	Kitermelő vállalat.....	12
4.5.2.	Feldolgozó vállalat.....	14
4.5.3.	Vevő.....	15
4.6.	Absztrakciós szint.....	15
5.	Konkrét példa: fejszegyár (demó program).....	17
5.1.	Tervezési fázis.....	18
5.2.	Megvalósítás.....	20
5.2.1.	Vállalat (Company) osztály.....	20
5.2.2.	Vevő (Customer) osztály.....	20
5.2.3.	Megrendelés (Order) osztály.....	20
5.2.4.	Számla (Bill) osztály.....	21
5.2.5.	Tárgy (Stuff) osztály.....	21
5.2.6.	Nyersanyag megrendelés (Resource_order) osztály.....	21
5.2.7.	Globális tömbök.....	21
5.3.	A program futása.....	21
5.4.	Szimulációs eredmények.....	22

5.4.1.	Szimuláció #1	23
5.4.2.	Szimuláció #2	23
5.4.3.	Eredmények	24
6.	További funkciók megvalósítása	25
6.1.	Különböző eloszlások szerinti igénygenerálás	25
6.2.	Port figyelés a kommunikáció megvalósításához	26
6.3.	Belső struktúra bővítése	27
6.3.1.	A két irányú bővítés	27
6.3.2.	Versenyhelyzet okozta problémák és azok megoldása	28
6.3.3.	Automatikus példányosítás	31
6.4.	Paraméterezési lehetőségek	33
7.	Összefoglalás	36
	Irodalomjegyzék	37

1. Bevezetés

Napjainkban egyre jobban elterjedté váltak a kisebb vállalatok körében is a vállalatirányítási rendszerek, melyek használata nagyban segítheti a vállalkozás megfelelő működését. A piac folyamatosan szélesedik, a felhasználók igényei megállíthatatlanul növekednek, így fontos, hogy számukra megfelelő alapot biztosítson az informatikai világ és kielégítse az igényeket. Ennek érdekében is történt, hogy egyre több ilyen vállalatirányítási rendszer került és kerül napjainkban is piacra, közte a piacvezető cégek védjeggyével (Microsoft, SAP, stb.), azonban nem szabad megfeledkeznünk a feltörekvő vállalatokról sem és a napjainkban egyre elterjedtebbé vált „open source”, azaz nyílt forráskódú szoftverek gyártóiról.

Intézetünk oktatási tematikájában is szerepelnek ezen irányzat elemei egy fő tárgy és egy teljes szakirány formájában. Feladatom elvégzése egy fontos részét képezi egy olyan rendszer megtervezésében, megvalósításában és majdani bevezetésében, amely a jövőben segíteni fog a hallgatók számára az egyetemen a vállalatirányítási rendszerekkel kapcsolatos tudásuk gyakorlati szintű felhasználására.

A tervezett rendszernek, amely egy virtuális vállalatirányítási környezetként fog működni, a feladata, hogy a hozzá kapcsolódó vállalatirányítási rendszereket összekapcsolja mintegy hálózatba, ezzel biztosítva az egymás közötti kommunikációt, alapot (egy világot) teremtsen számukra a kereskedésre és különféle eseményeket generáljon, amelyre a hallgatóknak az elméleti oktatás során szerzett ismeretük alapján kell megfelelően reagálniuk.

A feladatomat egy intézeti tárgy keretein belül kezdtem el és folytatom jelenleg is. Első lépésben a virtuális környezet több szinten történő megtervezését végeztem el, a valós életből vett információk és folyamatok felhasználásával, majd ezeket vittem át az informatika világába és ezek segítségével megvalósítottam egy olyan demó programot, amely egy szerveren futva, emberi beavatkozás nélkül a virtuális környezet alapját adva működik és moduláris jellegű, hogy a jövőben bővíthető legyen.

A további munkám során a demó rendszert tovább bővítettem, melynek során az egyszerű példát kibővítettem a korábbi absztrakt elgondolás szerint, így a rendszer indításakor paraméterezhetjük a programot, a benne szereplők közti kapcsolatokat definiáló struktúrákon, a résztvevők számán túl egészen az igénygenerálás eloszlásáig. Ennek értelmében a program indulása a paramétereknek megfelelően dinamikusan és automatikusan történik.

Jelenleg már két fő csatlakozott a projekthez, így az egyik személy kezdett neki a háttérben működő adatbázis megtervezésének, illetve a másik pedig a felhasználói interakciót lebonyolító webes felületnek. Közös feladatunk az, hogy ezek a részek külön-külön működjenek, illetve még a félév során megoldjuk ezek kommunikációját, integrációját.

2. A feladatról és az elérendő célokról

Az ötlet, aminek a megvalósításának először tervezési részét végeztem, egy összetett vállalatirányítási környezet létrehozása. Ennek a környezetnek a sajátossága, hogy ez tulajdonképpen egy világot definiál, amelyben minden egyes szereplő egy vállalatot vagy éppen személyt testesít meg az első elképzelés alapján.

A környezet egy független rendszerként képzelhető el, melybe a külön gépeken futó vállalatirányítási rendszerek úgymond becsatlakoztathatóak, függetlenül attól, hogy milyen rendszerek ezek. Első lépésben az intézet által is támogatott Microsoft Dynamics NAV élvez előnyt, de nem szabad elzárkózni semmilyen nemű nyílt forráskódú rendszertől.

Az elérés az első fázis során lokálisan lenne megvalósítva, azaz egy szoftverként futna a számítógépen, amely szerverként funkcionálna, és ehhez kapcsolódnának be, mint világba a kliensek. Később ez a rendszer változna, az egészet webes környezetben szeretnénk megvalósítani, azaz szerveren futna a “környezet” és az úgymond vállalatok (hallgatók) böngészőn keresztül érnék el azt, a hozzájuk tartozó részhez kapcsolódó virtuális felületet.

A megvalósítás nem teljes egészében egyszerre történik, hanem kisebb szintekre osztva készül el mindig az újabb verzió. Első körben egy kisebb szimulátor készült el a tervezési lépések után, azután pedig a bővítés történt meg, paraméterezhetőség, illetve ezzel párhuzamosan készül(t) az adatbázis és a webes felület, melyeket összekapcsolunk. További lépéseknek tekinthetők egy éles vállalatirányítási rendszer becsatlakoztatása, az automatikus entitások kivétele és emberi interakcióval helyettesítése, illetve a webes felület továbbfejlesztése. Ezek a lépések azonban további tervezést igényelnek, így célszerűbb lépésenként fejleszteni, bővíteni, mert így nyomon követhetővé válik a célok felé haladás.

2.1. Célok

A környezet létrehozásának elsődleges célja, hogy egy új elképzelés útján a hallgatók számára egy olyan rendszer kialakítása, melynek használatával a Vállalatirányítási rendszerek című tárgy gyakorlatát új tematika és megoldások szerint abszolválhatják. Ez alatt azt kell érteni, hogy mindenki egy önálló vállalatként kezd ebben a virtuális vállalati világban és működésüket fenntartva, egymással kereskedve a félév végéig minél jobb eredménnyel zárják a céget. Nyilván, ha valaki teljes csőddel zár, vagy éppen oda jut a félév során, a tárgy gyakorlata sikertelennek tekinthető.

A környezet egyik sajátossága lenne, hogy megfelelő eloszlás szerint generál eseményeket, amikre a hallgatóknak kell reagálniuk a legjobb tudásuk szerint. Például, a hallgató vállalata kerékpárokat gyárt és érkezik egy 200 db kerékpárról szóló rendelés, akkor nekik kell mérlegelniük, hogy teljesíthető-e a megrendelés, elkészíteni a terveket, végrehajtani a folyamatokat, számlákat elkészíteni, nyersanyag rendeléseket készíteni, stb., minden ehhez kapcsolódó feladat ellátása. Mivel a hallgatóknak feltétel a tárgy felvételéhez a Számvitel, illetve a Pénzügy tárgyak, így nem jelenthet problémát nekik egy számla, így felhasználhatják a gyakorlati részben a más tárgyak során elsajátított ismereteiket.

A rendszerben a hallgatók vállalatok, azonban lesznek különféle úgymond “NPC”-k (Non Player Character), amely kifejezés a virtuális játékokból ered. Ezek a szereplők lesznek a bankok, különféle szolgáltatók (áram, víz, gáz), melyek nélkül a vállalataink nem működhetnek. Amennyiben a vállalkozás növekedni szeretne, szeretné a termelést növelni, amelyhez hitel kell például, akkor fel kell keresnie az illetékes NPC-t és megtenni a szükséges lépéseket. Fontos megemlítenem, hogy ezek a tényezők csak a rendszer alapfunkcióinak működése után lesznek megtervezve, hiszen első sorban a teljes ellátási lánc kialakítása a cél, ahol a hallgatók a különféle szinteken tevékenykedhetnek.

A feladataim közé tartozott első körben a rendszer elképzelésének kibővítése, részletezése, majd a szükséges tervezési lépések elvégzése, melynek során kialakult a rendszerterv, a rendszerben szereplő entitások, azok kapcsolatai, funkciói. Elkészült egy demó program, amely egy egyszerűsített példát mutat be, annak automatikus (emberi interakció nélküli) működését, melynek futási eredményeit fel lehetett használni a paraméterek finomhangolására. Így egy adott struktúrára egy szimulátor készült el.

Jelenleg a feladatom a rendszer bővítése az absztrakt elképzelés alapján, kezdőfelület, melynek segítségével paraméterezhetővé válik a szimulátor. Emellett pedig a kapcsolati interfész elkészítése, melyen keresztül az adatbázissal fog kommunikálni a szimulátor az események alapján.

2.2. Játékosítás (*Gamification*)

Napjainkban nagy kihívást jelent a fejlesztők számára olyan szoftverek elkészítése, melyek folyamatosan naprakészek és segítik a felhasználó szakmai fejlődését, miközben érdekessé is teszi a számukra azt. E célból jött létre az a fogalom vagy irányelv, amit játékosításnak hívunk és napjainkban egyre inkább elterjedté válik, köszönhetően nagy sikerének.

Az embernek a vérében van a versengésre és a játékra való hajlam, így ha bármiben, amiben megtalálja ezeket az elemeket, azok felkeltik a figyelmét és érdekessé válik a számára. Ezt használja ki manapság sok ilyen ötlet. A projekt megvalósítása is ezt az irányt veszi alapul, a hallgatók versenyszellemére épít, azzal ösztönzi őket, hogy versenyezzenek a többi hallgatóval, hogy jobb jegyet szerezzenek, jobb eredménnyel zárják a félévet és közben pedig elsajátítják a gyakorlatok során azokat a szakmai ismereteket, amiket a tárgy során meg kell szerezniük.

Ezek az igények vásárlások vagy éppen megrendelések lehetnek. A megrendeléseket nyilvánvaló módon el kell bírálni, hogy teljesíthetők e, a vásárlási igények pedig az esetleges raktárkészlet alapján érkeznek.

2.3. Létező elgondolások, kapcsolódó ötletek

A Google segítségével próbáltam már meglévő megoldásokat találni, melyek az elképzelésünkhöz hasonlóan van kialakítva vagy legalábbis hasonlóságokat mutat ez irányba. A keresésem során a “*Virtual Business System*”, “*Virtual Business Environment*”, *Virtual Business System for Students*”, “*Virtual Business Environment for Students*” kulcsszavakat használtam.

Találtam olyan megvalósított projekteket, amelyek egy virtuális környezetet biztosítanak a vállalkozás számára (hasonlóan a *Second Life*-hoz) [1][2], melynek segítségével bekapcsolódva a virtuális világba egy vállalatot lehet szimulálni, az emberek karakterekként szerepelnek, és így mint egy virtuális meeting tudnak kommunikálni. Ez nem sokban tér el egy szimpla konferencia beszélgetéstől, azonban imitálni, vizualizálni próbálja a vállalati légkört, a helyszínt.

A másik találat, amely némely részben kapcsolatban áll(hat) a témámmal az a *Virtual Business Online* [3][4], amelynek két részének is utánanéztem: *Virtual Business Personal Finance Online* és *Virtual Business Management Online*. A lényegük, hogy a felhasználó egy személyt vagy egy vállalatot szimbolizál és alakíthatja annak sorsát. A vállalatnál a megszabott lehetőségeknek megfelelően kialakíthatja a gyártósorát, illetve a menedzsment és eladó osztályát. Ezeken kívül van még 3 része (*Restaurants, Sports and Entertainment, Retailing*), de ezekkel nem foglalkoztam.

Ezek a megoldások más célt szolgálnak, mint amit mi tűztünk mi magunk elé, hiszen nem a meglévő vállalatirányítási rendszerek számára kínálnak egy virtuális környezetet, amibe becsatlakozhatnak, hanem az üzleti kommunikációt kívánják segíteni vagy a tanulók számára

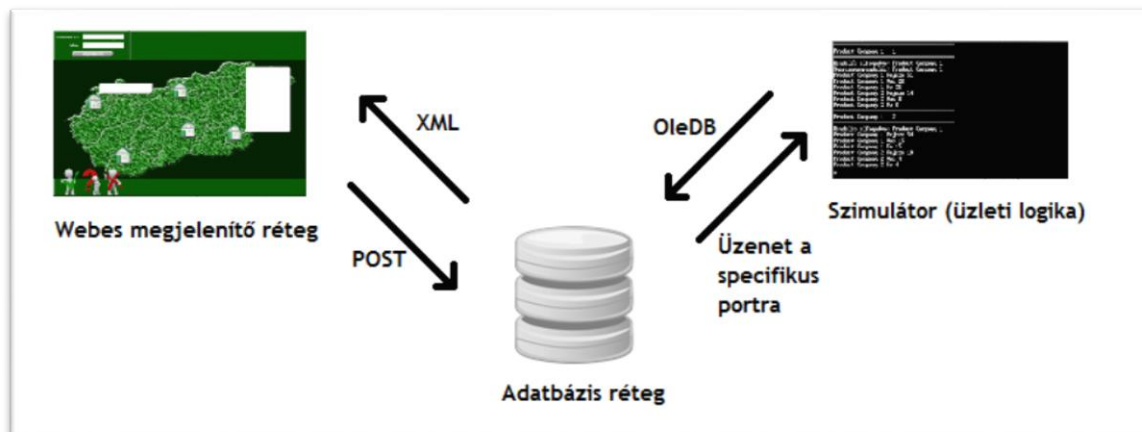
egy olyan online világot kínálnak, melynek segítségével egy vállalat gyártásának osztályát kialakíthatják, változtathatják kedvük szerint, bővíthetik vállalatukat. Itt nem nyílik lehetőség a számlázással, rendelés-feldolgozással kapcsolatosan az ismeretek szélesítése és az elméleti oktatás során szerzett tudás felhasználására.

Az ötlet ezért is egyedülálló, hiszen ötvözi az egyetemi oktatás során szerzett tudás felhasználását és a virtuális világ kínálta lehetőségeket.

3. Rendszerterv

A rendszer egy három részből (vagy szintből) álló architektúra alapján lehet elképzelni (1. ábra). Ennek értelmében a korábbiakban említettek alapján:

1. Szimulátor (a mag)
2. Webes felület
3. Adatbázis



1. ábra - Rendszerterv, architektúra

A szintek egymástól teljesen elkülönülnek, közöttük különféle technológiákkal valósul meg a kommunikáció olyan formában, hogy az a legmegfelelőbbben támogassa annak célját.

3.1. Szimulátor (mag)

A központi részt a szimulátor adja, amely már a tervezési fázis végén, a fejlesztési fázis legelső állomáspontján már demó verzióban elkészült. Ez a rész felelős az üzleti logikáért, vezérli az eseményeket. Szimulációs módban automatikusan dolgozza fel a generált megrendeléseket, további megrendeléseket készít az ellátási láncon felfutóan. Induláskor az egyedeket is automatikusan generálja a megadott résztvevőszám és struktúra alapján, a megrendeléseket az eloszlásnak megfelelően generálja. A standard módban a megrendeléseket kiküldi és a felületen leadott „válasz” alapján reagál és teszi meg a további lépéseket.

3.2. Adatbázis

Az adatbázis réteg a közvetítő szerepet látja el és emellett az adatok hosszú távú tárolására szolgál. A rendszer indulásakor a leíró adatokat tartalmazó táblák feltöltődnek a létrejött egyedek adataival, illetve futás közben az eseményeket naplózza is. Emellett a megrendelések állapota változik, számlák készülnek, kerülnek elfogadásra, ezeket mind az adatbázisban tároljuk el.

A másik fontos szerepe, hogy a két másik réteg közvetlenül nem kommunikál, így neki kell a kommunikációs csatorna szerepét ellátnia. Ez azért is fontos, mert csak akkor adja tovább a „jelet” ha az már az adatbázisba beíródik, így nem veszhet el adat, illetve létre nem jött adatról nem küld tovább értesítést. A szimulátorral egy specifikus porton keresztül küld üzenetet, melyre reagálva a szimulátor módosít a megfelelő adatokon a megfelelő táblában. A felület pedig hasonlóképpen értesíti, hogy változás volt, létrehoz egy XML fájlt, amit a felület felhasznál a megjelenítés során.

3.3. Webes felület

A webes felület a felhasználók számára a kommunikációs platform, itt érik el majd a világot, illetve tudnak interakcióba lépni a rendszerrel. Itt látják a megrendeléseiket bejelentkezés után, látják a többiekét, azonban egymás „titkos” adatait nem. Az adatokat, melyeket az adatbázis által generált XML fájlból olvas ki, elhelyezi a helyükre és csak az adatbázis értesítése esetén frissíti őket. A felhasználó a megrendeléseit látva dönthet a sorukról, elfogadja vagy elutasítja őket. Ezek az interakciók POST metódussal jutnak el az adatbázis felé, aki üzenetet küld a mag felé és így feldolgozásra kerül a reakció és beindítja a további funkciókat.

3.4. Szintek közötti kommunikáció

A kommunikáció a megfelelő szintek között közvetlenül történik, azonban a webes felület és a szimulátor (mag) közvetlenül nem kommunikálnak. A tervek szerint, induláskor elindul a mag, paraméterezi a felhasználó az induló felületen a programot és elindítja. Ekkor létrejönnek az egyedek (automatikusan), beillesztődnek a megfelelő adatok az adatbázisba, az oldal elérhetővé válik.

A működés során igények generálódnak, melyeket elküld a mag az adatbázisnak, beillesztődnek a megfelelő adatok a táblákba és ezt az adatbázis érzékeli, jelez a webes felületnek (ha be van jelentkezve a felhasználó), hogy változás történt, ennek megfelelően változnak a dolgok, amelyhez az adatokat az adatbázisból XML fájlok segítségével emel ki. A webes felületen a felhasználó reagál a megrendelésre és az adatokat a POST metódus segítségével adja át az adatbázis számára, amely eltárolja az adatokat, módosít azokon a megfelelő helyeken.

Miután az adatbázis frissült, a megrendelések változásának megfelelően üzenetet küld a magnak, amely az erre a célra fenntartott portot figyelve megkapja az üzenetet és annak megfelelően reagál, új megrendelést generál vagy éppen az elutasított megrendelést küldi el egy másik potenciális felhasználó számára.

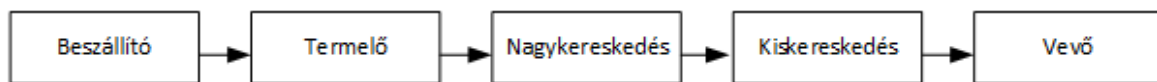
4. Tervezési szakasz

A tervezés során több lépésen végighaladva jutottam el a való életben szereplő elemektől egészen az informatika világában használatos elemek és fogalmak felé. A következő fejezetben részletezem ezeket a lépéseket.

4.1. Ellátási lánc a tervezett rendszerben

A rendszerben résztvevő vállalatok mind-mind más szerepet töltenek be, különböző szinten helyezkednek el a rendszeren belül. Ez az elosztás az ellátási láncnak megfelelően történik (2. ábra), melynek értelmében különböző szinteket lehet megkülönböztetni [5].

Ennek értelmében egy olyan rendszer adható meg, amely több tekintetben is rugalmas. A résztvevők több szinten helyezkedhetnek el, nem fontos, hogy egyes szinteken azonos mennyiségű résztvevő legyen, de nyilvánvalóan fontos, hogy ne legyen semelyiken egyetlen egyed.



2. ábra - Ellátási lánc

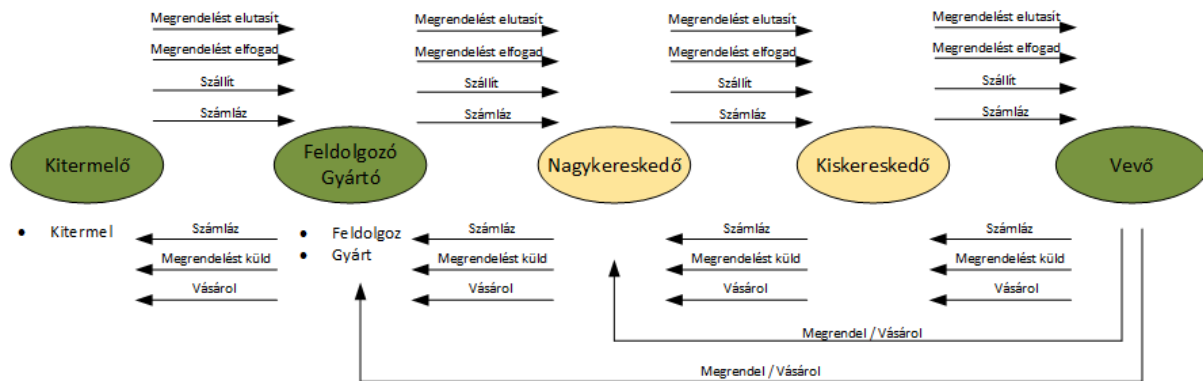
Fontos, hogy a résztvevőket megfelelően definiáljuk a feladatuknak megfelelően, azaz el legyenek különítve a szerepeik alapján. A vevői oldalon generálódó igények visszafele hatnak a rendszerben, azaz egy generált igény egy másik igényt generál, mégpedig a másik kereskedés felé, ami igényt generál(hat) a termelő vállalat felé, ami szintén igényt generálhat a beszállító felé.

A rendszerünk tulajdonképpen nem egy egyszerűsített ellátási lánc, hiszen nem 3 konkrét lépcsőfokból áll (Beszállító – Vállalat - Vevő), hanem különböző mennyiségben helyezkedhetnek el a belső szinteken egyedek, így az ellátási láncok hossza 3-tól akármeddig terjedhetnek, persze a realitás határain belül.

4.2. Entitások

A rendszerben az elosztási lánchoz hasonlóan különböző szinteken helyezkednek el az entitások és a szintjüknek megfelelően végeznek különböző feladatokat (3. ábra). Az elsődleges megvalósítási fázisban, mikor még demó verzióban fog futni a szimulátor, mindent a számítógép vezérel, azaz nem lesznek külső entitások, csupán belsők, melyeket a rendszer vezérel. Ezzel a cél a környezet működésének demonstrálása és az esetleges hibák felfedezése.

Illetve első körben még csak a gyártó, vevő szintekre szorítkozunk, később ez a struktúra bővítve lesz, egészen az infrastrukturális szintig, azaz az alapvető nyersanyagok szolgáltatásáig, illetve áram, víz, stb.



3. ábra - A rendszer entitásai és azok folyamatai

Ez alapján megkülönböztetünk három fontos résztvevőt, melyek a demó rendszerben fognak elsőként szerepet játszani.

- **Kitermelő vállalat:** A nyersanyagokat megadott intenzitással képes kitermelni a világból és szolgáltatni a további felhasználás céljából. Például: vasat kitermelő vállalat napi 1 tonna intenzitással képes kitermelni a vasat, amit már kész eladni a további felhasználásra egy feldolgozó/gyártó vállalatnak.
- **Feldolgozó/gyártó:** A gyártó vállalat, amely a kitermelő vállalatoktól megvásárolja a kitermelt nyersanyagokat és feldolgozás után, mikor elkészül a profilja szerinti terméke, eladja azt a vevőnek, illetve teljesíti a megrendeléseket. A rendelkezésre álló nyersanyagokból állítja elő a termékét megfelelő intenzitással, például percenként egy termék.
- **Vevő:** A legalsó szintet képviseli. Innen érkehetnek a vásárlási igények főként (köztes gyártók is vásárolhatnak), megrendeléseket küldenek a gyártó cégek számára, melyek dönthetnek arról, hogy elfogadják azt vagy elutasítják. Emellett egyszerű vásárlás is lehetséges, amennyiben nem tervezett mennyiségről van szó és van a gyár raktárában a kívánt mennyiségben a termékből.

A későbbiekben köztes szintek is lesznek, melyek a demó leegyszerűsítése miatt nem kerülnek még be, ezek a kis- és nagykereskedők, de ezek csak a gyár és a vevő között helyezked(het)nek el, így a tesztelési fázishoz első fázisban nem tartottam fontosnak őket, persze később bekerültek a rendszerbe.

4.3. Folyamatok

Az egyes entitások tevékenységeit fontos részletezni, hiszen ezek adják meg az egyes elemek közti kapcsolatot és a köztük végbemenő interakciókat, ezért a következőben ezeket fogom leírni.

- **Megrendelés küldés:** Az egyes entitások küldhetnek egymásnak megrendeléseket, melyeket a megszabott intenzitásnak megfelelően generálnak a potenciális megrendelők. Ezek a megrendelések tartalmazzák a mennyiséget a termékből és a határidőt, melyet teljesítenie kell annak, aki elfogadja azt. Mivel több potenciális gyártó is van, ezért véletlenszerűen kapja az egyik, amennyiben elfogadja az, akkor nincs további folyamat, amennyiben visszautasítja, akkor a fennmaradók közül kapja meg egy másik vagy elveszik.
- **Megrendelés elfogadás:** A gyártó vagy termelő vállalat kapja, dönthet róla, hogy elfogadja a megrendelést, amennyiben teljesíthető az. Ez egy üzenetet képvisel a megrendelő felé, hogy a határidőig meg fogja kapni a kívánt mennyiséget a termékből.
- **Megrendelés elutasítás:** Amennyiben a megrendelést határidőre nem teljesíthetőnek értékeli a gyártó, akkor elutasítja azt, ez egy szimpla válasz a megrendelésre, ami arra sarkalja a vevőt, hogy másik gyártóhoz forduljon.
- **Szállítás:** A két fél közötti anyagmozgatást reprezentálja. A “kereskedő” vállalatok raktárainak tartalmán (termék mennyiségén) változtat.
- **Számlázás:** Ez a lenyomata annak, hogy a megrendelő vagy vevő és a termelő vagy gyártó között tranzakció volt. Ennek során a két fél a megrendelés adatait eltárolja a későbbi nyomon követés céljából
- **Kitermelés:** A kitermelő vállalat megfelelő intenzitással termeli ki az alapvető nyersanyagokat. Ennek az intenzitása az adott anyagtól függhet, mennyiség/idejében adható meg és folyamatos, a kitermelt anyag elraktározásra kerül.
- **Gyártás:** A kitermelő vállalatoktól vásárolt/megrendelt anyagokból gyártja a terméket. Ennek során az alapanyagok felhasználásra kerülnek és létrejön a késztermék, ami a raktárba kerül.
- **Vásárlás:** A megrendelés feldolgozása során, mikor a vállalat látja, hogy jelenleg nem megrendelésre gyárt és van a raktárban szabadon a kívánt mennyiség a termékből, akkor nem a megrendelés raktározódik el és kezd el arra gyártani, hanem egyből elfogadja és fel is dolgozza azt, megkezdődik a szállítás, számlázás, stb.

4.4. Alapvető versenyhelyzet

A versenyhelyzet elveszhet olyan esetben, ha egy adott szinten a rendszerben egyetlen ember helyezkedne el, azaz egyetlen hallgató töltene be egy adott ágazatban szerepet (pl. kerékpár készítés), ekkor monopolhelyzetbe lenne, az összes igény hozzá érkezne. Ennek a helyzetnek az elkerülése végett a rendszert valamilyen szinten önállóra készítjük, úgymond intelligensre, hogy ha látja a rendszerben, hogy egy adott terméket csak egy vállalat gyárt, akkor automatikusan létrehoz másik NPC vállalatot, amely fenntartja vele a versenyt. Persze itt nem azt kell elképzelni, hogy minél inkább elnyomni akarja majd a hallgatót, hanem megadott valószínűséggel ő kapja a megrendelést, nem a hallgató vállalata, illetve ha a hallgató elutasítja a megrendelést, akkor automatikusan megkapja az.

4.5. IT specifikáció

Fontos a való életből vett elemeket vagy entitásokat az informatika nyelvére lefordítva lejegyezni, ennek eredményeként egy olyan vázlatot vagy specifikációt kapunk eredményül, melynek segítségével ezeket a dolgokat könnyebben át lehet vinni egy informatikai környezetbe. Ennek fényében a fejezet első részében az entitásokat, mint résztvevőket fogom úgymond definiálni, illetve az azokhoz kapcsolódó függvényeket.

Az entitásokhoz tartozó alapvető adatokat változóknak fogjuk eltárolni, a folyamataik pedig lesznek a hozzájuk tartozó függvények. A teljes specifikáció alatt ragaszkodni fogok az angol nyelvű leíráshoz.

4.5.1. Kitermelő vállalat

A legalapvetőbb vállalat, ami az ellátási láncban az elején helyezkedik el. A legfontosabb alapadatok, amelyeket ezekkel a vállalatokkal kapcsolatban el kell tárolni:

Cégnév - Company_name: szöveges **string** változó, a vállalat teljes neve

Helyszín vagy telephely - Location: szöveges **string** változó, a vállalat telephelyének címe

Raktár - Warehouse: egy lista, melynek elemei **termék objektumok**, melyek tartalmazzák a nevet és a mennyiséget, így több terméket is lehet tárolni a listában, mint egy raktárban.

Típus - Type: egész szám **int** változó (ez alapján lesznek majd kategorizálva a vállalatok aszerint, hogy kitermelő vagy feldolgozó/gyártó)

Termék - Product: szöveges **string** változó, a vállalat portfóliója alapján a termék, amit gyárt (például fejszegyár)

Bankszámlaszám - Bank_account_number: egész szám **int** változó.

“Számlatömb” – Bill_list: egy lista, melynek elemei **számla objektumok**, tartalmazzák a két résztvevő számlaszámát, nevét, a megrendelés megfelelő adatait.

Megrendelés lista – Order_list: egy lista, melynek elemei **megrendelés objektumok**, tartalmazzák a már elfogadott rendeléseket, amelyekre éppen gyárt a vállalat és határidőre teljesíti azokat.

Foglaltsági dátum – Busy_till: egy **Date** típusú változó, melyben el van tárolva az a dátum, melytől a vállalat nem megrendelésre, hanem raktárra gyárt már, fontos a rendelésfeldolgozás során, figyelembe kell ezt venni a kalkulációkor.

Minimális raktárszint – Minimal_inventory_level: egy **int** típusú változó, ha bármelyik nyersanyag szintje ez alá esik, automatikusan nyersanyag megrendelés generálódik a kitermelő vállalat(ok) felé.

A kitermelő vállalatnak a legalapvetőbb folyamatait különálló függvényekként lehet leírni, definiálni:

Kitermelés/Gyártás: A legalapvetőbb funkciója a vállalatnak. Ez egy olyan folyamat, ami folyamatosan fut, azaz folyamatos a kitermelés, azonban figyelnie kell arra, hogy a megrendelésre gyártás van e folyamatban, ennek függvényében történik az intenzitás alapján a termelés az alapvető vagy a megrendelési raktárba. Ennek az intenzitása például 1 tonna/óra. Ennek eredményeül az adott raktár tartalmának számértékét növeli 1 óra alatt eggyel. Vizsgálja, hogy ha megrendelés alatt zajlik a kitermelés, hogy megvan e a megfelelő mennyiség, akkor szállítja azt és visszaáll az alapvető raktárba való termelésre, illetve nullázza az aktuális megrendelő változóit.

Rendelés elfogadás: Megvizsgálja, hogy mi a határidő, kiszámítja, hogy mennyi anyagot képes a vállalat kitermelni addig, hozzáadja a már az alapvető raktárban lévő mennyiséget és amennyiben a kapott érték legalább annyi, mint a megrendelés, akkor elfogadja azt, az alapvető raktárból átkerül minden a megrendelési raktárba és a kitermelés során oda kerül a közben termelő is.

Rendelés elutasítás: Amennyiben nem teljesíthető a megszabott határidő, akkor elutasítja azt és a vállalatán belül nem történik változás.

Szállítás: Mikor megvan a megfelelő mennyiség, akkor átküldi a megadott mennyiséget a megrendelőnek és megkezd a számlázást is egyben.

Számlázás: A termelő és a megrendelő közötti tranzakciót naplózza. Eltárolja a fájlban a két kereskedő cég adatait és a határidőt, mennyiséget. Ezt mindkét fél eltárolja.

Nyersanyag rendelés: A minimális raktárszint jelzőt minden gyártási folyamat ellenőrzi a program az adott vállalatnál és amennyiben az alá esett bármelyik nyersanyag, úgy megrendelést készít a kitermelő vállalat számára, hogy a raktárkészletet ismét egy optimális szintre emelje.

Rendelések teljesítése: A vállalat, ha elfogad egy megrendelést, de nem azonnal teljesít, akkor elraktározza azt, majd amikor elkészül a rendelési mennyiség, akkor a tárolt megrendelések közül az esedékest teljesíti.

4.5.2. Feldolgozó vállalat

Hasonló a helyzet a kitermelő vállalathoz, annyi eltéréssel, hogy ez már nem a kiinduló szintet képviseli, így amennyiben megrendelést kap, nem csak a saját készleteit kell megvizsgálnia az elfogadás vagy elutasítás előtt, hanem feljebb kell lépnie az ellátási láncon és megrendelésként lekérdezni, hogy határidő előtt mennyi alapanyagot képes megszerezni a gyártáshoz, mindezt visszafelé számolva időben, hogy elegendő ideje maradjon a gyártásra is.

Megrendelés mérlegelése és megrendelés küldése: Ez a kitermelő vállalathoz képest összetettebb. Megvizsgálja, hogy mennyi van meg az adott termékből készen a raktáron, mennyihez elegendő nyersanyag van a raktárban és mennyi idő alatt képes abból készterméket előállítani, ezt levonja a határidőből, majd a megrendelés darabszámából ezeket levonva, kapunk egy hiányt, amihez nyersanyagot kell szerezni. Ennek fényében kiszámolja, hogy mennyi időre van szükség a maradék darabok legyártásához, majd, a végére maradó időt, mint határidő, mellécsatolva az igényelt nyersanyag mennyiség, elküldi megrendelésként a beszállító kitermelő vállalatnak, ami nyilván akkor tudja azt fogadni, ha képes határidőre elkészülni.

Amennyiben rendben van, megtörténik a megrendelés, amint megvan a nyersanyag átkerül a gyárba, ott elkészül a termék és megy a megrendelőnek. Ha viszont nem tudja teljesíteni,

elutasítja a megbízást, amennyiben van másik beszállító, annál is próbálkozik, ha nincs, akkor a vevőtől érkező megrendelést elutasítja.

4.5.3. Vevő

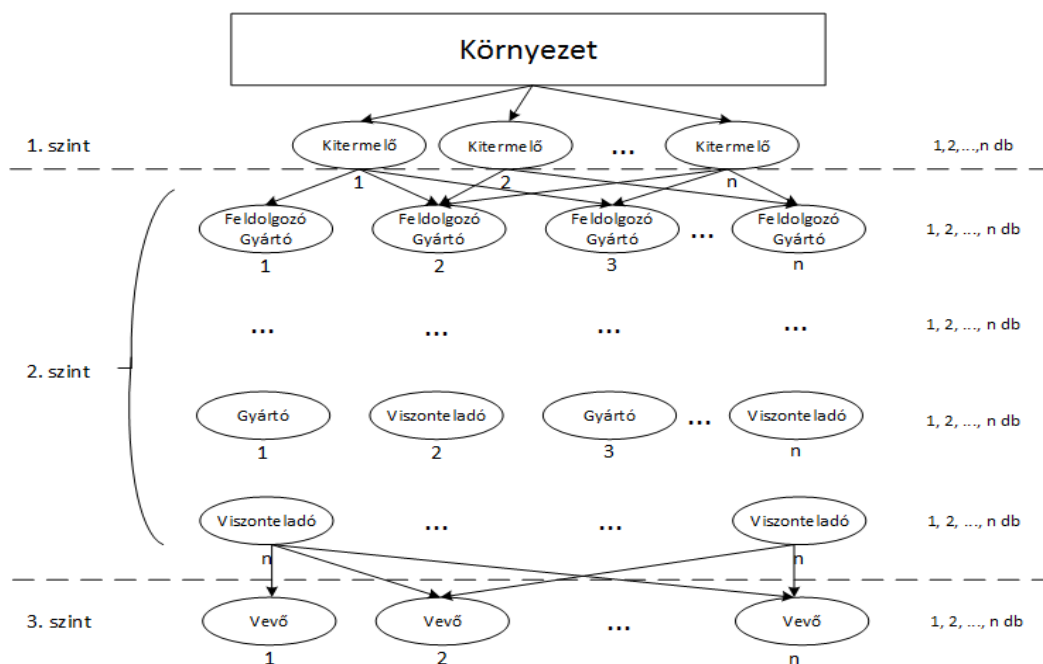
Az előre definiált intenzitásokkal generálja a vásárlási és megrendelési igényeket. A rendszerben több ilyen vevő is lesz, ezek külön-külön bírnak ezzel a tulajdonságukkal. A vevőről csak néhány adatot tárolunk el, hogy később be tudjuk azonosítani, hogy kivel történt a kereskedelem és hogy a számlázás megfelelően működjön.

- **Vevő neve** - Customer_name: szöveges **string** változó
- **Számlaszám** - Bank_account_number: egész szám **int** változó
- **Számlatömb** – Bill_list: egy lista, **számla objektumokkal** feltöltve

Hasonlóan a gyártókhoz megrendelést generál annyi különbséggel, hogy nem végez rajtuk tovább „megmunkálást”. Ezek folyamata ugyanúgy történik, mint a fentebb leírt esetben.

4.6. Absztrakciós szint

A rendszert fontos, hogy első körben absztrakt módon definiáljuk, hogy átláthatóvá tegyük azt és egyszerűbben megérthetővé. Az elemek nincsenek semmilyen szinten konkretizálva, hiszen így még nincs megszabva a köztes szintek száma, az egyes vállalatok közti lehetséges kapcsolat, hiszen ezekkel már elveszteni a terv az értelmét.



4. ábra - Absztrakciós ábra

Az absztrakciós ábra (4. ábra) az ellátási láncot reprezentálja, azonban több szinten. Nem csupán egy láncot mutat be, hanem egy, kettő vagy akár több láncot egymás mellett, azok végigfutását és kapcsolatait. Emellett mutatja azt, hogy az egyes entitások, a láncok elemei hogyan vannak kapcsolatban egymással, milyen kölcsönhatások vannak az egyes szintek között.

- Az első szinten nyilván a kitermelő vállalatok helyezkednek el. Ezek teljesen egyedülálló kategória, hiszen a környezetből ezek a gyárak “szedik ki” a nyersanyagokat, alapanyagokat, melyeket onnantól kezdve akár különböző ellátási láncon végighaladva készülnek el a késztermékek, amelyeket a vevőknek értékesítünk.
- A második szinten vannak a feldolgozó és gyártó vállalatok. Az ábrán látható, hogy 1, 2, ..., n db ilyen vállalat helyezkedhet el az egyes szinteken, illetve az ilyen szintekből is több lehet. Például a fát kitermelik, lesz belőle gerenda, ami egyik irányban haladva lefele a láncon egy feldolgozó üzemén áthaladva léceket ad eredményül vagy falapokat, amelyeket akár az egyszerű vevő is megvásárolhat. Egy másik láncban haladva feldolgozzák a fát és további megmunkálások során különféle bútorokat készítenek belőlük. Így a két “útvonal” eltérő szintekkel rendelkezik formában és mennyiségben is. Az egyik eltérő résztvevő ezen a szinten a viszonteladó, az a vállalat, aki megveszi a készterméket vagy félkész terméket és azt kínálja a vevők számára, például megveszi a fejszéket, kalapácsokat, ásókat, kapákat és szerszámboltként (mint vállalat) kínálja ezeket, akár egy nagykereskedés.
- A harmadik szintet a vevők képviselik, akik az egyes láncok végén keletkező végtermékekre az igényforrásokat, azaz akik a terméket megvásárolják. Ez az utolsó szint, hiszen itt ér véget az ellátási lánc.

5. Konkrét példa: fejszegyár (demó program)

Az absztrakt környezet önmagába foglalja a környezetből kitermelő vállalatokat, feldolgozókat, vevőket. Ez az első koncepció alapján létrejött rendszer, ami az ellátási láncot adja, és mint egy hálózat épülhet ki az egyes résztvevők között. A feladatom elkészítése során egy egyszerűbb példával próbálom szemléltetni azt, hogy két kitermelő vállalat által produkált anyagot egy feldolgozó gyár miként hasznosítja, és mint készterméket szolgáltatja az igények szerint, melyeket a vevők generálnak a definiált eloszlás szerint.

Az általam egyszerű példának vélt ellátási lánc egy fejszékét gyártó vállalatot vesz alapul, melynek mindkét oldalról fel kell állítani a láncban résztvevő további részeit. Ennek értelmében szükség van egy kitermelő vállalatra, ami megfelelő intenzitással kitermeli a fejsze gyárunknak a vasat a környezetből, illetve egy fát kitermelő vállalatra, amely a faanyagot szolgáltatja. Ezeket azért így jellemzem, mert a környezetből történik a kitermelés, megadott intenzitással képesek ezek a vállalatok úgymond a semmiből előállítani nyersanyagot (a virtuális környezet lényege). A kitermelők feldolgozzák az ércet és a nyersfát és már feldolgozható állapotban szolgáltatják azt. A raktár kapacitásuk végtelen, így nem kell törődni azzal, hogy leálljon-e a termelés vagy sem.

A következő szinten helyezkedik el az a gyár, amely a fejszékét készíti ebből a két alapanyagból. A mennyiségi arányok adottak, ennek fényében történnek a megadott mennyiségű fejszéhez az alapanyag rendelések. A gyártási intenzitás itt is adott lesz, azaz megadott idő alatt képes a virtuális gyárunk egy fejszét elkészíteni, majd raktározni azt, hogy teljesítse a vevők igényeit.

A vevők szintje a legutolsó lépcsőfok, itt generálódnak az igények, amelyek megrendelés formájában jutnak el a gyárba, melyek hatására az ellátási láncban felfutva generálód(hat)nak újabb igények a feljebb elhelyezkedő résztvevők felé.

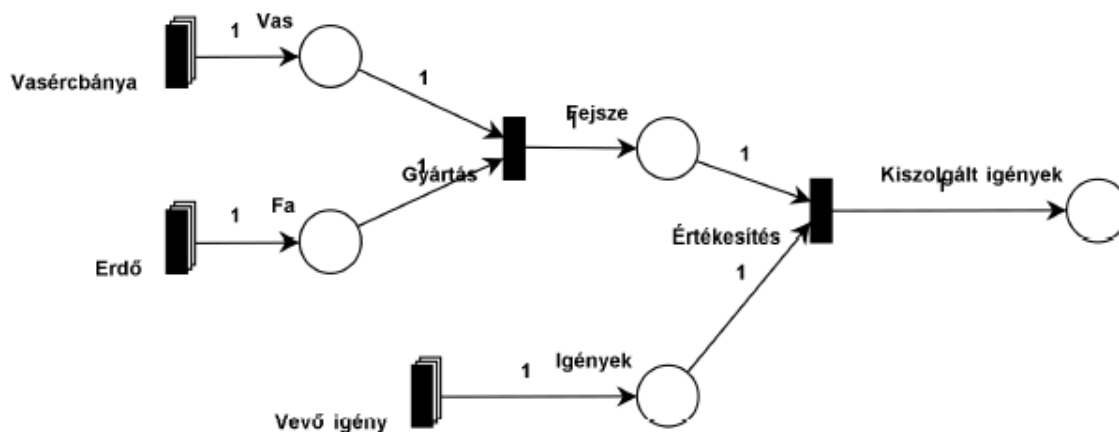
Fontos megjegyezni, hogy ez a rendszer nagyon stilizált, hiszen nem csak fára és vásra van szükség való életben egy fejsze elkészítéséhez, hiszen a gyár nem csak nyersanyagot rendelhet, hanem a gépek meghibásodása esetén alkatrészre is szüksége lehet, a fejszenyel lakkozásához is szükség van alapanyagra, azonban ezektől eltekintünk annak érdekében, hogy egy átlátható és megvalósítható környezetet kapjunk alapul. Illetve a másik fontos tényező pedig az, hogy ebben a rendszerben minden automatizált, azaz az NPC-k a résztvevők, ők kommunikálnak

minden nemű közbeavatkozás nélkül, reprezentálják a tervezett rendszer működését még emberi interakció nélkül.

5.1. Tervezési fázis

Az előző részben az absztrakt példából kiindulva, konkretizálva a demo rendszer résztvevőit, elkészült a fejszés példa, ami tekinthető egy elsődleges kiindulási struktúrának, emberi résztvevők nélküli rendszernek.

A fejszés példát végiggondolva és felhasználva az Elektronikus szolgáltató rendszerek című tárgy során felhasznált tudásomat a Petri-hálókról, elkészítettem az egyszerűsített hálót a fejszés elképzeléshez.



5. ábra - Petri háló a fejszés példához - 1. verzió

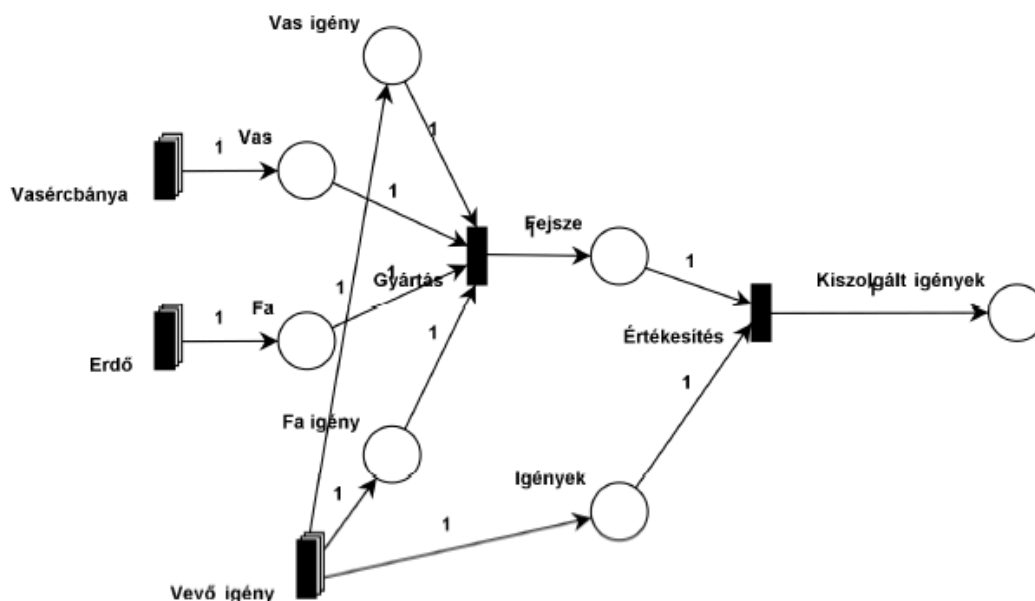
Az ábrán látható (5. ábra), hogy a kétféle nyersanyagot két különböző forrással jelöltem, ezek a forrás tranzícióim. Megadott intenzitással (a kitermelésnek megfelelően) az alapanyagokat küldik a kitermelő gyárak raktárainak jelölt helyekre (Vas, Fa), ahol felhalmozódik a nyersanyag. A fejszgyár egy tranzícióként van jelölve, ami ha van raktáron már nyersanyag, a megfelelő arányoknak szerint (jelen esetben 1-1 arány) felhasználja azokat és a megfelelő tranzíció (Gyártás) létrehozza a terméket, a fejszét, ami a gyár raktárában tárolódik (Fejsze). Ekkor kerül a képbe a Vevő igény, mint tranzíció. Ez azt kívánja reprezentálni, hogy megfelelő intenzitással igényeket generál, amelyek tokenekként jelennek meg (Igények). Így a kapott igények és fejszék 1-1 arányban, a tokenjeik alapján egy újabb tranzíción keresztül (Értékesítés) a tokenek elvándorolnak a végső helyre (Kiszolgált igények).

Fontos megjegyezni, hogy itt az él súlyok 1 értéket kaptak, mert könnyebb az átláthatóság szempontjából. Az egyszerűség kedvéért termel ki 1 egységnyi nyersanyagot időközönként a kitermelő vállalat, ami lehet például 1 kilogramm vas és 1 gerenda vagy teljesen más mértékegység

szerint, illetve ezt a mértékegységet felhasználva készülnek el a fejszék is. Azonban a kitermelési intenzitás változtatható és választható, hogy a vas például lassabban legyen kitermelve, mint a fa.

A vevő igényének a súlyozása változtatható, azaz, nyilván ha a vevő igénye 5 db fejsze, akkor 5 igény token keletkezik, és azokat kell kielégítenie a gyárnak, ha közben érkezik egy 2 db fejszés igény, akkor újabb 2 token keletkezik az igények körében. Ez dinamikus és a vevői igénytől függ. Első elgondolásban az intenzitás mellett a “vevő” 1-5 skálán generálhat darab szempontjából igényeket.

Az első példát továbbgondolva pedig az ellátási láncon visszafutó hatást is próbáltam szemléltetni, ennek fényében pedig a vevői igény generálása során nem csupán az igényekhez kerül token, hanem a fejszegyárhoz is generálódik igény. Ennek fényében csak akkor történik fejsze gyártás, ha tényleges vevői igény is létrejön, azaz nem gyárt raktárkészletre a vállalat, de ez azt is eredményezheti persze ilyen tekintetben, hogy ha sok igény érkezik, akkor feltorlódhatnak az igények, illetve szélsőséges esetben a rendszer sose kerül olyan állapotba a kezdeti üres járat miatt, hogy nem kell gyártania. Ennek érdekében fontos ebben az esetben az igény generálásának az intenzitását megfelelően beállítani.



6. ábra - Petri háló a fejszés példára - 2. verzió

A jelentősége a második elgondolásnak (6. ábra), hogy mutatja a felfele futó igényeket, azaz, hogy a késztermékre generált igény miként generál a lánc másik vége felé is hasonlóan igényt az alapanyagokra. Csak akkor történik nyersanyag rendelés a kitermelő vállalatok raktárából, ha a fejszegyár maga is kapott igényt már a vevőktől.

5.2. Megvalósítás

A megvalósítás során, mint említettem, történtek eltérések a tervezés fázishoz képest, de voltak olyan elemek, amelyek megmaradtak eredeti formájukban. A fejlesztést Microsoft Visual Studio 2010-el kezdtem el a .NET keretrendszerben C# nyelven.

5.2.1. Vállalat (Company) osztály

A tervezési fázisban elképzelték alapján került megvalósításra, azonban összevonásra került a kitermelő vállalattal. A kettő közötti megkülönböztetést a típus változó adja. A legfőbb metódusai közé tartozik a gyártás, a rendelések mérlegelése és elfogadása, illetve elutasítása. Új adattagot vettünk fel, a marketing indexet, amelynek a későbbiekben lesz majd szerepe, a sok elutasított igény alapján csökken az értéke, ezzel csökkentve a valószínűségét, hogy ismét megrendelést kapjon. Egyik fontos változója a foglaltságot jelző adattag, ennek szerepe, hogy mutassa, melyik időpontig gyárt már a rendszerben lévő rendelésre a vállalat, azaz innen számolhat a következő megrendelés határidejéig az elbíráláskor, hogy képes e addig legyártani az igényelt terméket.

5.2.2. Vevő (Customer) osztály

A rendszerben a vevő generálja az igényeket úgymond és küldi őket egy véletlenszerűen választott feldolgozó vállalat felé. A legfontosabb folyamata a rendelések kezelése. Miután elküldi a megrendelést, várja annak eredményét. Amennyiben pozitív választ kap, a jelenlegi állapot alapján nem tesz semmit, ha negatív választ kap, akkor a potenciális vállalkozások listájáról, akik kielégíthetik a megrendelését lehúzza azt, amelyiktől negatív visszajelzést kapott és továbbküldi egy másik lehetséges vállalatnak. Ez a folyamat addig folytatódik, ameddig nem kap pozitív választ, avagy elfogynak a lehetséges vállalkozások, akkor a megrendelés elveszik a rendszerből és egy erre a célra kialakított listába kerülnek.

5.2.3. Megrendelés (Order) osztály

Az alap elgondolásban nem szerepelt egy külön entitás, a megrendelés. Ez az osztály azért született meg, mert ezt passzolgatják egymás között az NPC egyedek, tartalmazza a megrendelés adatait, minden megrendelés egy külön objektumként jelenik meg a rendszerben. Adattagjai a megrendelés határideje, a két fél, akik között áthalad (változik a vállalat, miután visszakapja a vevő negatív elbírálással) és a megrendelésben szereplő anyag/tárgy és annak mennyisége.

5.2.4. Számla (Bill) osztály

Jelen pillanatban ez az objektum még csak nyomon követési célokat szolgál, azaz naplózza, hogy melyik cégek között történt adás-vétel, későbbiekben ennek fontos szerepe lesz a folyamatok szempontjából.

5.2.5. Tárgy (Stuff) osztály

A megvalósítás egyszerűsítése érdekében érdemes volt létrehozni ezt az osztályt. Két adattagja van, a mennyiség és a tárgy neve. Ennek az osztálynak a lényege, hogy összesíteni lehet a rendszerben lévő tárgyak/anyagok listáját és a megrendeléseknél és raktározás során ilyen objektumokban tudjuk tárolni a mennyiségeket. Alapértelmezetten minden vállalatnak megvan minden ilyen objektum, de csak amivel kereskednek, azok fognak a későbbiekben 0-nál nagyobb értéket kapni a mennyiség adattagban.

5.2.6. Nyersanyag megrendelés (Resource_order) osztály

Ez az osztály egy segédosztály, mivel a demo verzió során a kitermelő vállalatainknak végtelen raktárkészletük van, így a megrendelést nem kell elbírálni, így másként kell kezelni azt.

5.2.7. Globális tömbök

A globális tömböket azért hoztam létre, hogy a rendszerben lévő entitások példányait nyomon lehessen követni, elérni őket és megfelelően felhasználni. Ezek segítenek a véletlenszerű kiválasztás során. Az alábbi ilyen tömbök lettek létrehozva:

- **global_customer_list:** A rendszerben lévő vevőket tartalmazó lista.
- **global_company_list:** A rendszerben lévő összes vállalkozást tartalmazza, beleértve a kitermelő és a feldolgozó vállalatokat is.
- **global_producting_company_list:** A rendszerben lévő összes feldolgozó vállalat listája. Fontos, mert amikor a vevő igényt generál, innen választ, hiszen nem nyersanyagot fog rendelni, hanem készterméket.
- **global_lost_order_list:** A rendszerben való vándorlás során az összes potenciális vállalkozáson végigfutva sem lett kielégítve, így elveszett és itt eltároltam.

5.3. A program futása

A program futásának lépései az alábbi módon követik egymást:

1. Az inicializációs folyamat lefut, létrejönnek a vállalatok, az anyagok/tárgyak példányai, a vevő(k) és a létrejött példányokból feltöltődnek a megfelelő globális tömbök. A kimeneti szöveges fájlban „fejlécként” megjelennek a feldolgozó vállalatok paramétereikkel.
2. Elindul a „végtelen” ciklus, melynek futása reprezentálja a folyamatos időintervallumot, persze a szimulációs folyamatok során ez diszkrét időintervallumra volt szűkítve.
3. A véletlen szám generátor segítségével legenerálódik az igény egy véletlenszerűen választott vállalatnak, egy véletlenszerűen választott vevőtől, véletlenszerű mennyiségi megrendelésről és elküldésre kerül, megindul az elbírálás. Az elbírálás eredménye naplózásra kerül a szöveges fájlba.
4. Miután a megrendelés elfogadásra került vagy bekerült az elveszett igények listájába, lezajlik a gyártás az adott napra minden vállalatnál, illetve a vállalatok, ha a minimális nyersanyag készlet szint alá esik, akkor nyersanyag megrendelés keletkezik. A nyersanyag megrendelés szintén eltárolódik a napló fájlba.
5. A „nap” végén legutoljára pedig minden feldolgozó vállalat raktárának tartalma kiíródik a fájlba és ezzel lezárult egy ciklus.

5.4. Szimulációs eredmények

A programot szimulációs célból két eltérő paraméterezés szerint futtattam le (1. táblázat), melynek az eredményeit egy hétre, hónapra és évre nézve is bemutatom.

		Termelési sebesség	Kezdeti késztermék mennyiség	Kezdő vas mennyiség	Kezdő fa mennyiség
Szimuláció #1	Vállalat 1	3	10	10	10
	Vállalat 2	2	10	20	20
Szimuláció #2	Vállalat 1	5	20	20	20
	Vállalat 2	4	30	10	10

1. táblázat - Szimulációk paramétereit

A vállalatok, amint a nyersanyag szintjük a minimális limit alá esett, rendelést adtak le a kitermelő vállalatoknak, amelyek azonnali teljesítéssel kiszolgálták őket, hiszen azok raktármennyiségei végtelenek voltak.

5.4.1. Szimuláció #1

A két feldolgozó vállalat gyártási sebessége **2** és **3**, az generált igény **1-10**ig terjedő mennyiségű lehet. A kezdeti értékek az első vállalat esetében: **10** fejsze, **10** vas, **10** fa. A második vállalat esetében: **10** fejsze, **20** vas, **20** fa. A kitermelő vállalatok raktártartalma végtelen, a feldolgozó vállalatok termelése folyamatos, minimális készlet szint alá esés alkalmával (nyersanyag) megrendelést küld, és **20-20** nyersanyag érkezik a raktárába.

A szimulációt 365 periódusban, azaz 365 napot reprezentálva futtattam. Az eredmények azt mutatták, hogy az igények túlnyomó része elfogadásra került, némelyek csúsztatva (azaz nem azonnal kiszolgálva) és csak kevés került egy vállalat által elutasításra/átpasszolva, elenyésző pedig elveszett.

	Rendelést kapott	Rendelést elfogadott	Rendelést elutasított	Rendelést tárolt és később teljesített
Vállalat 1	190	203	26	74
Vállalat 2	175	147	46	73
Rendelés elveszett	15			

2. táblázat - Szimuláció #1 eredményei

Jól látható (2. táblázat), hogy az a vállalat, amelyik nagyobb mennyiséget gyárt naponta, annak kevesebb (majdnem fele annyiszor) kellett elutasítania a megrendeléseket, többet is tudott így elfogadni (köztük azokból is bőven, amelyeket a másik vállalat előtte elutasított).

5.4.2. Szimuláció #2

A két feldolgozó vállalat gyártási sebessége **4** és **5**, az generált igény **1-10**ig terjedő mennyiségű lehet. A kezdeti értékek az első vállalat esetében: **30** fejsze, **10** vas, **10** fa. A második vállalat esetében: **20** fejsze, **20** vas, **20** fa. A kitermelő vállalatok raktártartalma végtelen, a feldolgozó vállalatok termelése folyamatos, minimális készlet szint alá esés alkalmával (nyersanyag) megrendelést küld, és **20-20** nyersanyag érkezik a raktárába.

Az eredményen látható (3. táblázat), hogy mindösszesen 2-vel növelve a napi termelést, hasonló megrendelési intenzitás mellett a megrendeléseknek már a 100%-a elfogadásra kerül, sőt a megrendelések minden esetben egyből eladásig jutottak, hiszen mindig volt a raktárban megfelelő késztermék mennyiség és azonnal ki tudta elégíteni a megrendelést. Ezen adatok fényében egyértelmű, hogy a rendszerben egyetlen megrendelés sem került az elveszett megrendelések listájába.

	Rendelést kapott	Rendelést elfogadott	Rendelést elutasított	Rendelést tárolt és később teljesített
Vállalat 1	187	187	0	0
Vállalat 2	178	178	0	0
Rendelés elveszett	0			

3. táblázat - Szimuláció #2 eredményei

5.4.3. Eredmények

Az eredmények mutatják, hogy az egyes paraméterek kisebb változtatása, illetve a kezdeti értékek variálása is változtatni tudja a folyamatok eredményeit.

A két szimuláció eredményei is jól mutatják, hogy csak a kezdeti nyersanyag és késztermék mennyiségek voltak megváltoztatva, illetve a gyártási sebességek eltérőek az előzőhöz képest. Mégis, az egyik esetben 100%-ban eladás módján történt az értékesítés, még csak eltárolni se kellett a megrendeléseket, míg a másik esetben voltak elutasítások, sőt kettős elutasítások, melynek eredménye a megrendelés elvesztése volt és ez a megrendelések kb. 4%-át érintették.

Fontos megjegyezni, hogy a jelenlegi álláspont szerint nem teljes még a való élet szimulálása, hiszen szállítási idő nincs, a kitermelő vállalatok kapacitása végtelenre van választva, csak egy forrásból érkeznek igények, a folyamatokban nem keletkezhet hiba, azaz nincs a termelés során leállás, selejt stb. Ezeknek a megoldása a rendszert már túl bonyolulttá és nehezen kezelhetővé tette volna, illetve túl összetett és egyben felesleges lett volna megvalósítani ilyen mélységekben.

6. További funkciók megvalósítása

Az első félév során a tervezési lépéseken végighaladva eljutottam egy kezdeti demó programhoz, amely már szimulátorként egy kezdő alapot biztosított a további munkálatokra. A jelenlegi félév során folytattam tovább a fejlesztést, melynek során több lényeges mérföldkőhöz is elérkeztem.

6.1. Különböző eloszlások szerinti igénygenerálás

Az első fázisban mikor a szimulátor elkészült, az igények generálása egyenletes eloszlás szerint történt. Ehhez a .NET keretrendszer által szolgáltatott Random osztályt használtam fel, amely egyenletes eloszlás szerint generálta számomra az igényeket a megadott intervallumon belül, illetve egyenletes eloszlás szerint választott a paraméterként átadott globális listákból. Ennek eredményéül pseudo-random módon került eldöntésre, hogy mekkora mennyiségből, melyik vállalat, milyen határidővel kapjon megrendelést.

Fontos viszont megjegyezni, hogy ez az eloszlás nem a legmegfelelőbb számunkra, hiszen csak a megrendelési mennyiséget tekintve, hogy 1-10 terméket rendel egy vállalat, nem ugyanakkora a valószínűsége az intervallum egyik, illetve másik végén lévő értéknek, ha a valós világot vesszük alapul (bútorgyár, 1 székrenyt vesznek vagy 10-et, mind a kettő lehetséges, de az előbbinek a valószínűsége reálisabb).

Ennek fényében fontos volt, hogy a való élethez sokkal inkább igazodó eloszlásokat is implementáljuk a programba, ezzel növelve a „világ” létjogosultságát és erősíteni a valósággal a kapcsolatot. A programban három eloszlás szerint lehet az igénygenerálást beállítani, amelyek a következők [6]:

- Poisson
- Exponenciális
- Normál

Az egyenletes eloszlás csupán azért volt felhasználva az előző fázisban, mivel még csak a belső rendelés feldolgozás rész volt a legfontosabb, illetve a tervezési lépések, ezért csak az egyszerű véletlen szám generátort használtam. Azonban az élet egy sztochasztikus folyamatként fogható fel, ezért nem megfelelő azt egyenletes eloszlással reprezentálni és kevésbé használható fel az igénygenerálásra, illetve a sorban állási modellek esetében se szokás egyenletes eloszlást alkalmazni, sokkal inkább a Poisson eloszlást szokták előszeretettel használni.

Az eloszlások, amiket felhasználtam nem saját kezűleg lettek implementálva, egy már korábban elkészült *MathNet.Numerics* [7] csomag *Distributions* osztálykönyvtárát használom fel. Ezen belül lévő *Poisson*, *Exponential* és *Normal* osztályokat használom fel, azoknak a *Sample* (minta) metódusát, ami a típustól függően kért paraméter mellett (*Poisson* => λ , *Exponenciális* => ráta, *Normál* => átlag, szórás) generál nekünk egy véletlenszámot az eloszlásnak megfelelően. A paraméterek változtatásával változtathatunk az eloszlásfüggvényen, így a kapott számok intervalluma módosítható, azonban ezeket a jelenlegi fázisban fixáltuk, hogy nagyságrendben 1-10 közötti értékeket kapjunk, néhány kiugró eredménnyel.

A felhasznált eloszlások eredményeül olyan kimagasló (kiugró értékű) megrendelések is születhetnek, ahol pl. 10 nap határidőt kap a vállalat 1 termék legyártására vagy pont az ellentéte, 1 napot kap 10 vagy több termék legyártására. Ezek a jelenségek általában vagy könnyű elfogadást vagy totális elutasítást vonnak maguk után, hiszen ha épp megrendelésre gyárt a vállalat, biztosan nem tud egy 15 termékre szóló megrendelést másnapra teljesíteni, ha napi 3 terméket készít. De ezek a jelenségek viszont a való életre vezetnek vissza a rendszert, hiszen megtörténik az életben is, hogy ilyen esemény történik egy vállalkozás életében és a vezetőknek kell mérlegelni, hogy melyik megrendelés a fontosabb, melyik vagy éppen mindkettő teljesíthető e. Pont itt jön a képbe a hallgatói interakció, miszerint a hallgató fogja eldönteni azt, hogy elfogadja-e a megrendelést ilyen esetben vagy sem, itt már csak a saját ismereteire esetleg megérzéseire támaszkodhat.

6.2. Port figyelés a kommunikáció megvalósításához

Az architektúrában szereplő három résztvevővel való kommunikáció megvalósításához, azon belül is az adatbázissal való kapcsolattartást szolgálja egy port figyelő szál a programban. Ez azért fontos, mert ezen a porton tud üzenetet küldeni az adatbázis a szimulátor (mag) felé, hogy változás történt és ez alapján reagálhasson.

A kommunikáció TCP-n keresztül valósul meg, a szimulátornak az 5555-ös portot tartom fenn, hogy azon tudjanak a „kliensek” kommunikálni. Az adatbázis küld egy szöveges üzenetet, amiben benne vannak a paraméterek elválasztva egy megfelelő karakterrel. A szöveg feldolgozódik és az értékek átadódnak a megfelelő metódusnak, ami a paraméterek alapján elvégzi a szükséges műveleteket.

Ez a szemlélet már az integrációs folyamat részét képezi, itt már a három szint egymással kommunikálva dolgozik, illetve itt már szeretnénk, ha a webes felületen tudna a felhasználó a megrendelésekre reagálni, azaz az automatizmust innen már kiiktatnánk, és emberi interakcióra támaszkodnánk. Fontos megjegyezni, hogy nem végleges ez, mivel szeretnénk a továbbiakban is akár szimulátorként használni a programot, ezért egy kapcsoló fog bekerülni a felületre, ahol ki lehet választani, hogy normál üzemmódban menjen a program vagy éppen a szimulációs módban. Ezek a dolgok még fejlesztés alatt állnak jelen pillanatban és zajlik az integráció a három réteg között.

6.3. Belső struktúra bővítése

Az első fázisban, amikor még csak a demó programról beszéltünk, akkor egy fix struktúrája volt a rendszernek és csak a paramétereket lehetett állítani a vállalatokon belül. Ekkor pontosan két kitermelő vállalat volt, amelyek különböző nyersanyagokat termeltek, két feldolgozó vállalat, akik egymás konkurenciái voltak és egyetlen vevő. Napi egy igényt generált a vevő, amely aztán ment a feldolgozókhoz és lettek elbírálva.

Ez a rendszer még nem igazodott az absztrakt elgondoláshoz, csupán egy lehetőséget mutatott be a variációkból és programkód szinten kellett hozzányúlni, ha változtatni akartunk. Ezen túllépve jött a következő fázis, amely szerint a megadott paraméterek alapján lehet két irányban bővíteni a rendszert.

6.3.1. A két irányú bővítés

A rendszer két irányba való bővítéséhez első sorban el kellett dönteni, hogy milyen módon történjen ez, illetve, hogy milyen szintekre osszuk ezt fel, hiszen az ésszerűség határain belül kell maradni. Ennek értelmében függőleges irányban a bővítés 2 lépcsőre lett bontva:

- Feldolgozó vállalatokon között lévő kapcsolatok alapján létrejövő szerkezet
- A feldolgozók és a vevők közötti szintek szerkezete

A feldolgozó vállalatok közötti kapcsolatokat úgy kell elképzelni, mint egy fa gráfot, hiszen minden egyes vállalat egy belső szintet képvisel, alatta vagy nincs már másik feldolgozó vagy felette nincsen, avagy lehetséges, hogy köztes szinten helyezkedik el, ilyenkor mindkét irányban van feldolgozó alatta és fölötte is. Fontos megjegyezni, hogy ez a szerep az ellátási láncban való elhelyezkedés miatt adódik, valakitől kapja a nyersanyag(ka)t, valakinek pedig szolgáltatja a félkész terméket vagy készterméket.

A feldolgozók és a vevők közötti szintek a nagykereskedők és a kiskereskedők. Ezek opcionális jellegűek, mivel azok nélkül is elindítható a program, de hozzá is lehet őket adni, ilyenkor a számukat kell csak megadni (minimum 2 a verseny miatt). A nagykereskedő az, amit először hozzá lehet adni, de nem szükséges utána a kiskereskedőt is hozzáadni, azonban ha kiskereskedőt akarunk hozzáadni, annak szükséges feltétele, hogy a nagykereskedőket is hozzáadjuk. Ez azért is egyértelmű, mert ha nem lennének a nagykereskedők csak a kiskereskedők, akkor igazándiból csak névben lenne eltérés, hiszen nincs másik kereskedő a rendszerben, akik úgymond alattuk helyezkednének el.

6.3.2. Versenyhelyzet okozta problémák és azok megoldása

A legnagyobb problémát az jelentette, hogy a felhasználó számára szabadságot adjunk abban, hogy milyen értékeket szeretne paraméterül adni a programnak a résztvevők szempontjából. Azonban egy elég nagy problémába ütköztünk, amely az alapvető verseny helyzetét borította fel.

Első elképzeléseim alapján úgy gondoltam, hogy a felhasználó megadhatja, hogy ő szeretne a rendszerben példának okáért 3 kitermelő vállalatot, 10 feldolgozót, 3 nagykereskedőt, 1 kiskereskedőt és 8 vevőt. Ezzel nem is lenne gond, csak meg kell szabni a programban, hogy ki kivel kereskedik, hiszen a termék fix és a termék definiálja az adott vállalat ellátási láncban elfoglalt helyét.

A problémát az adta, hogy mindenhol teljesülnie kell a lokális versenynek és ekkor az egész rendszerre elmondható, hogy korrekt, mert így globálisan is teljesül a verseny. Ha legalább két résztvevő van egy adott ponton az jó, mert akkor ők egymással versenyhelyzetet alkotnak, így mindenhol minimum két résztvevőre van szükség. Ez az elképzelés a kitermelő vállalatoknál a legegyszerűbb.

A kis- és nagykereskedéseknél is még rendben van, hiszen alapvetően nem túlbonyolítva a rendszert, minden ilyen jellegű cég minden terméket árusít, azaz a vevő bármelyik termékért az azt gyártó cégeket és az összes kereskedést is potenciális beszerzőhelynek tekinti. Ez azért jó, mert így mindenkinek ugyanakkora esélye van kapni megrendelést egy adott termékre, illetve a lokális verseny is sértetlen, minden gyártó egy terméket kínál, a verseny megvan, a nagy- és kiskereskedők mindent kínálnak, a saját szintjükön belül egyenlő esélyekkel indulnak, így ott is adott a verseny.

A legnagyobb gondot és fejtörést a feldolgozó vállalatok adták. Ha mindenki ugyanazt gyártaná, nem lenne probléma, hiszen mindenki egymással verseng és le van tudva a dolog. Azonban ha már több terméket akarunk gyártani, akkor a lokális verseny megtartása érdekében egy oszthatósági problémába ütköztünk, melynek a következő (a versenyt és a korrektséget nem sértő) megoldásai voltak (lásd 4. táblázat):

2 vállalat	Egy terméket gyárt mind a kettő.
3 vállalat	Egy terméket gyárt mind a három.
4 vállalat	Egy terméket gyárt mind a négy VAGY két termék két-két vállalattal. Egyforma esélyek, mert mindenkinek, ha az ő terméke lett kiválasztva, 50% esélye van, hogy rendelést kapjon, így korrekt a rendszer és mindenhol megvan a lokális verseny is.
5 vállalat	Egy terméket gyárt mind az öt.
6 vállalat	Egy terméket gyárt mind a hat VAGY két terméket gyárt három-három vállalat VAGY egy késztermék és két félkész termék és minden csomóponton két-két vállalat.
7 vállalat	Egy terméket gyárt mind a hét.

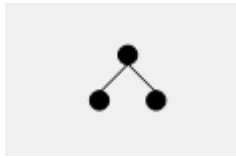
4. táblázat - Oszthatósági lehetőségek

A fenti példákön látható, hogy az egy termék az egész rendszerben elv az mindig működik, csak éppen semmi ellátási láncbeli problémát nem erősít fel, azaz ha a láncon felfut egy megrendelés egészen a nyersanyagig, így nem túl célszerű. Illetve ennyi azonos terméket gyártó vállalat mellett győznie kellene a kitermelő vállalatnak is, hogy mindet ellássa nyersanyaggal és emellett a megrendelések nagy eséllyel sose vesznének el, mert mindig lenne egy olyan vállalat, aki teljesíteni tudná azt.

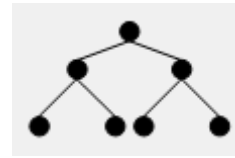
Az oszthatósággal való probléma megoldás jó lett volna, azonban a prímszámokkal nem tudunk mit kezdeni, olyan esetben mindig egy termék lenne, ami az előbb felvetett problémát adná. Emellett ha valamelyik nagyobb prím többszöröse jön elő (pl. $2 * 13$), akkor például két féle termék lenne és mindből 13 gyártó, ami megint csak elveszi a rendszer lényegét.

Ennek eredményeül a konzultációk során megállapodtunk a konzulenssemel, hogy le kell szűkíteni mindenképpen a felhasználói szabadságot ezen a téren, azonban mégis beletenni valamilyen szintű lehetőséget. Így jutottunk arra, hogy konkrét struktúrákat alakítunk ki, amelyeket kiválaszthat a felhasználó és azokat paraméterezve adhatja hozzá a rendszerhez. Így lett a végére két konkrét struktúra:

- 2 szintű 3 csomópontos fa struktúra (7. ábra)
- 3 szintű 7 csomópontos fa struktúra (8. ábra)



7. ábra - 2 szintű 3 csomópontos fa struktúra



8. ábra - 3 szintű 7 csomópontos fa struktúra

Mindkét esetben egy kiegyensúlyozott szabályos fáról van szó. Minden csomópont alatt két olyan csomópont helyezkedik el, amellyel kapcsolatban áll, így lokálisan hasonlóak az esélyek, hiszen az adott terméket ugyanannyian gyártják, bármely csomópontot is vesszük alapul és mindenkinek ugyanannyi termékre van szüksége a saját termékének az előállításához.

Gondolkodtam még egy olyan lehetőségen is, hogy a 8. ábrához hasonlóan legyen a struktúra, annyi különbséggel, hogy a legalsó levél szinten középen a két csomópont az egy, azaz a 2. szinten mindkét gyártónak szüksége van arra a termékre a sajátjuk elkészítéséhez. Azonban akkor annak a gyártónak előnye lenne, kétszer akkora valószínűséggel érkezne neki megrendelés, hiszen két termékbe is felhasználják az ő termékét, így ezt valahogy orvosolni kell.

Ekkor gondoltam arra, hogy a gyártáskor a nyersanyag felhasználásnál egy súlyozást alkalmaznánk, például ha abból a termékből kettő kell, amelyet csak ennek a terméknek a gyártásához használunk fel, akkor abból, amely két termékhez is kell, abból csak egyet. Ezáltal a mennyiség útján egyenlítenénk ki azt, hogy mekkora eséllyel kap megrendelést az a gyár. Végül ezt az elgondolást félre tettük, mivel ez a szimulációs rész nem a végső cél így felesleges túlbonyolítani az, de ha a későbbiekben szükség lesz rá, akkor van már egy felhasználható ötletünk ez ügyben.

A problémát végül ezekkel az előre definiált struktúrákkal oldottuk meg, bár csökkentve az előre tervezett felhasználói szabadságot. A felhasználó ezeket a struktúrákat adhatja hozzá a számára kiválasztott mennyiségben (egy struktúra => egy termékstruktúrát jelent, két ilyen struktúra között nincs kapcsolat, maximum a kitermelő vállalatok lehetnek közösek) és a kiválasztott résztvevőszámmal a csomópontokon. Ezeket egy listában láthatja, törölhet belőlük, adhat hozzá újabbakat.

6.3.3. Automatikus példányosítás

A második komoly lépés a rendszer belső struktúrájának bővítésével az volt, hogy a példányosítás a paraméterek átadása után automatikusan történjen, és ne kelljen a termékstruktúrát előre meghatározni, hanem minden automatikusan menjen.

A program mikor a paramétereket megadtuk, az OK gomb lenyomásával elindul, azaz a háttérben ekkor kezdődnek meg a példányosítások. Ez a folyamat két részre osztva történik, mi szerint szét kell venni a két struktúrát, amely közül a felhasználó választhat, hiszen ezek két különböző termékstruktúrát definiálnak, így külön kell őket kezelni.

A felhasználó megadott struktúrái egy listában helyezkednek el, amit megkap az erre a célra létrehozott *Constructing* osztály és elkezd soronként felolvasni azt. A lista tulajdonképpen egy *DataTable* táblázat, amit egy *DataGridView* komponenssel jelenítünk meg. A tábla három oszlopot tartalmaz:

- struktúra megnevezése
- struktúrák száma
- a struktúrán belül a csomópontokon lévő egyedek száma

Ezeket az adatokat sorról-sorra kiolvassa a program és feldolgozza. E célból két különálló metódust hoztam létre a két struktúra felépítéséhez, melyek mindig az adott struktúrához kapcsolódó paramétereket. A két metódus alapjaiban véve egyezik, azonban a 3 szintű 7 csomópontos struktúráé egy kicsit bővebb a másikhoz képest.

A metódusokat próbáltam minél inkább hatékonyra készíteni, minden fontos lépést egybesűríteni, illetve rekurzív megoldásra törekedni, így a programkód is sokkal átláthatóbbá válik. Az így elkészült metódus egy három egymásba ágyazott *for* ciklus, melynek első szintje a struktúrák számának megfelelően léptet, a második a struktúrán belül a csomópontokon léptet (itt van az első eltérés a két metódus között, egyik 3 másik 7 lépés), a harmadik pedig aszerint léptet, hogy a felhasználó a csomópontonként hány darab egyedet szeretne.

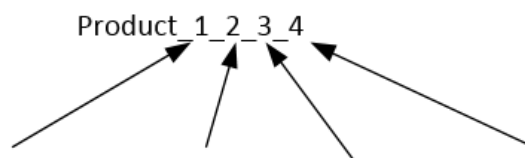
Ekkor legbelül történik a példányosítás, ahol létrejönnek a vállalatok az adott csomóponton (3. ciklus), amelyek ugyanazt a terméket gyártják, majd lép a következő csomópontra (2. ciklus) és ott is létrejönnek a vállalatok és így tovább. Annyit még fontos megjegyezni, hogy a termékek is ekkor jönnek létre, hiszen hozzá kell adni a gyár raktárába, mint lehetséges termék,

illetve az első csomóponton lévő vállalatoknak a két alatta elhelyezkedő vállalat termékét, amiből el fog készülni a sajátja.

Az egyik fontos dolog, hogy minden nemű név egy indexstruktúra alapján generálódik, így egy hírtelenjében elég átláthatatlan, azonban annál hasznosabb megoldás szerint vannak a cégnevek és a terméknevek kialakítva. A program indulásakor létrehozok néhány változót, amiben a termék és a cég indexek vannak eltárolva, illetve számolom azt, hogy az egyes struktúra típusokból hány darabnál járunk. A termék és cég indexek kezdetben 1 értéket kapnak, a struktúrák pedig 0-át. Amikor létrejön egy vállalat, akkor ez az index eggyel nő. A termék index egy kicsit eltér ettől, az csak csomópont váltás esetén változik, hiszen az adott csomóponton minden vállalat ugyanazt a terméket gyártja. Amint lefut egy teljes struktúra lépése és ugrik a következőre, a struktúra számláló is lép egyet.

Az eredményül kapott nevek több információt is tartalmaznak (9. ábra), mint például a termék neve tartalmazza azt, hogy milyen típusú struktúrában van (2 szintű 3 csomópontos vagy 3 szintű 7 csomópontos), mi a struktúra indexe (hányadik ilyen típusú struktúra), hányadik csomópont (a struktúrán belül hányadik csomópont, fentről lefelé, balról jobbra haladva vannak számozva) és hogy globálisan hányadik sorszámú termék ez. Az első három paraméter az a termék helyzetének a beazonosításához fontos, az utolsó pedig egy nyilvántartás jellegű, hogy a rendszerben tudjuk követni, hogy hány féle termék szerepel.

Ezek után már csak a vállalatot be kell helyezni a globális tömbökbe, amelyek funkcionálisan csoportosítják őket (kitermelő, feldolgozó, kis-nagykereskedés, kaphat e megrendelést vevőtől, stb.). Emellett a termékeket is egy ilyen globális tömbbe töltjük, hogy elérjük őket, illetve egy másik tömböt is használok, amibe azok a termékek kerülnek, amelyeket rendelni tudnak a vevők, ez a kis- és nagykereskedők példányosítása során fontos, hogy milyen termékeket kell a raktárunkba helyezni majd, amiket árusítani fognak.



A termék melyik struktúra típus, a struktúra indexe, melyik node, hányas termék (globálisan)

9. ábra - Indexstruktúra a termék nevében

A vevők nyilvántartása is hasonlóképpen történik, azokat is indexelem, de őket csak a létrejöttük sorszámával a nevükben, illetve őket is egy tömbbe töltöm, hogy onnan könnyedén

elérjük. A kis- és nagykereskedők pedig ugyanígy, annyi különbséggel, hogy a terméklistájukat (raktár) fel kell tölteni az összes lehetséges termékkel, amit vevő meg tud rendelni.

Ezáltal, hogy így hozzuk létre az egyedeket, teljesen szabályosan jönnek létre, a nevük rengeteg információt hordoznak magukba, amely fontos a további műveletek számára és könnyedén tud a rendszer alkalmazkodni a különböző, felhasználó által adott paraméterekhez.

6.4. Paraméterezési lehetőségek

A program demó verzió utáni egyik legfontosabb fejlesztési lépése az volt, hogy ne csak a programkód szintjén lehessen megszabni azt, hogy milyen paraméterekkel működjön az, hanem egy a felhasználó számára is átlátható kezdő panel legyen (10. ábra), amelyen tetszés szerint beállíthatja, hogy milyen alapvető paraméterekkel szeretné a virtuális környezet szimulátort elindítani.

Fontos megjegyezni, hogy a program angol nyelven készül, így minden egységesen angol nyelven íródik, a felületen a szövege, illetve a programkódban használt kifejezések is követik ezt az irányt.

Start settings

Levels

☐ 3 - Extract, Manufacturer, Customer

☒ 4 - Extract, Manufacturer, Customer, Retailer

☐ 5 - Extract, Manufacturer, Customer, Retailer, Wholesaler

Customers: 8

Retailers: 12

Wholesalers: 2

Distributions

☐ Poisson

☒ Exponential

☐ Normal

Manufacturers inner structure

3-level-7-node structure

Count of structures: 1

Companies on each node: 3

2-level-3-node structure

Count of structures: 2

Companies on each node: 2

Type	Count	Companies on each node
3-level-7-node	1	2
2-level-3-node	2	2
3-level-7-node	1	3

OK

10. ábra - Kezdő panel

A panel négy jól elkülöníthető, ám egyaránt fontos részre osztható fel:

- Szélső szinteken lévő résztvevők számának beállításai
- Eloszlás beállítása
- Belső struktúrát hozzáadó rész
- Hozzáadott struktúrákat tartalmazó lista

Szélső szinteken lévő résztvevők számának beállítása

Alapvetően a rendszer három szintből áll: Kitermelő – Feldolgozó – Vevő. Kezdőállapotban a felületen ez a beállítás jelenik meg, így csak a vevők számát állíthatjuk, mivel a kitermelő vállalatok, azok a belső struktúrához igazodóan automatikusan generálódnak, így ahhoz nincs lehetőségünk hozzányúlni, illetve a köztes feldolgozókon sem tudunk állítani, hiszen azokat a belső struktúrák beállításával megadtuk, de ezekről a későbbiekben is említést teszek.

További két lehetőségünk van az alapvető beállítás mellett, két lépésben is bővíthetjük a szerkezetet a nagykereskedőkkel, illetve a kiskereskedőkkel. Ezeket külön-külön kell hozzáadni, más szerepet látnak el, ezért fontos, hogy először csak a nagykereskedő kerül hozzá és utána a kiskereskedő, vagy csak a nagykereskedő.

A lehetőségeket és a beállíthatóságot korlátoztam, hiszen csak azokat a paramétereket lehessen állítani, amelyeket szeretném hozzáadni a későbbiekben a struktúrához. Illetve az értékek minimumának a 2 értéket adtam meg, hogy létrejöjjön az adott lokális szinten a versenyhelyzet, mint ahogy azt már korábban említettem.

Eloszlások beállítása

A korábban már említett eloszlásoknak egy külön rész lett kialakítva. A megfelelő rádiógomb kiválasztásával tudjuk az adott eloszlásra állítani a programot.

Belső struktúrát hozzáadó rész

A korábbiakban említettem, hogy a tervezés és az automatizáció során felmerültek problémák azzal kapcsolatban, hogy a versenyhelyzet fenntartható maradjon, így két struktúra részletet választottam ki, melyeket fel lehet használni a belső struktúra kialakításához. Ezeket egy-egy gomb segítségével lehet hozzáadni egy listához, miután beállítottuk, hogy hány darab ilyen struktúrát szeretnénk hozzáadni és hogy hány darab egyed (vállalat) helyezkedjen el az adott „gráf” struktúra egy-egy csomópontjában.

A versenyhelyzet fenntartása érdekében itt is a minimális egyedszám a csomópontokon kettő, illetve maximalizáltam őket tízre, hiszen felesleges lenne szimulációs célok végett olyan hatalmas struktúrát létrehozni. Azonban azt, hogy hány darab ilyen struktúrát adunk hozzá a rendszerhez, az nincsen szabályozva, viszont érdemes az értelmes kereteken belül maradni.

Hozzáadott struktúrákat tartalmazó lista

A hozzáadott struktúrákat egy listában jelenítem meg. A lista önmagában nem módosítható, csupán a két gomb segítségével adhatunk hozzá újabb elemeket, illetve a lista sorának bal szélén lévő cellára vagy éppen cellákra kattintva (kijelölve) az egész sor(ok) kijelölődnek, ekkor a *Delete* (Törlés) gomb lenyomásával eltávolíthatóak a listából.

A felület beállítása után az *OK* gomb lenyomásával a beállított paramétereket átadja a felület a szimulátornak, amely feldolgozza azokat, aszerint állítja be a szükséges paramétereket (pl. eloszlás) és hozzá létre a belső struktúrát, példányosítja a vállalat objektumokat. A gomb alapvetően nem aktív, csak akkor válik használhatóvá, ha legalább egy lista elem van a belső struktúráknál, hiszen anélkül a rendszernek nem lenne értelme. Illetve ha hozzá is adtunk a listához, de töröltünk onnan, ez esetben ismét nem aktív állapotba kerül a gomb.

7. Összefoglalás

Napjainkban a vállalatirányítási rendszerek térhódítása folytán egyre nagyon igény van az ilyen jellegű rendszerek használatában jártas szakemberek képzésére. Fontos, hogy az oktatásban a hallgatók mindig naprakész és megfelelő tudáshoz jussanak és manapság egyre fontosabb, hogy ez a tudás minél figyelemfelkeltőbb módon jusson el hozzájuk.

Az általunk elindított projekt egy olyan egyedülálló rendszert testesít meg, amely szükségessé teszi a hallgatók számára azt, hogy a már korábban megszerzett tudásukra támaszkodjanak éles helyzetben a gyakorlati oktatás során, illetve, hogy mindezt egy olyan környezetben tegyék meg, ahol egymással versenyezve tevékenykedhetnek.

Az első fázisban megtörténtek a tervezési lépések, kialakult a rendszerterv, elkülönítésre került a három réteg, amelyek külön kerülnek megvalósításra, azonban végül ezek integrálva lesznek egy komplex rendszerbe. A tervezési részt lezárva elkészült a program demó verziója, amelyben egy fix ellátási lánc működése valósult meg szimulációs jelleggel.

A második fázisban elindultak a webes felület, illetve az adatbázis oldalról történő fejlesztési lépések, miközben a szimulátor (mag) fejlesztése is folytatódott. A rendszer az absztrakciós elképzeléshez igazodva bővült a belső struktúráját tekintve, az eredeti egyenletes eloszlás helyét felváltotta az exponenciális, Poisson és normál eloszlások. Elkészült a program kezdő felülete, ahol a felhasználó tetszés szerint beállíthatja a környezet induló paramétereit, így téve a rendszert rugalmassá. A fázis végére elkészül az új szimulátor verzió, amely az új tulajdonságokat megkapja, illetve beiktatásra kerül egy új mód, amelyben már a webes felülettel való integráció után kommunikáció útján a felhasználói interakciótól függően történnek az események, nem pedig automatizáltan.

További fejlesztési munkák várnak még, hogy a három réteg teljes mértékig összhangban működjön, illetve a következő mérföldkő, ami pedig egy éles vállalatirányítási rendszer (Microsoft Dynamics NAV) bekapcsolása lesz a rendszerbe, azonban ez a lépés már csak a harmadik fázisban fog megtörténni.

Irodalomjegyzék

- [1] 3D Explorer - <http://www.3dexplorer.com/> - Utolsó megtekintés: 2014.10.12.
- [2] VenueGen - <http://www.venuegen.com/> - Utolsó megtekintés: 2014.10.12.
- [3] Knowledge Matters - <https://www.knowledgematters.com/business-simulations/virtual-business-personal-finance/> - Utolsó megtekintés: 2014.10.12.
- [4] Knowledge Matters - <https://www.knowledgematters.com/business-simulations/virtual-business-management/> - Utolsó megtekintés: 2014.10.12.
- [5] Budapesti Corvinus Egyetem Vállalatgazdaságtan Intézet archívuma, 100. sz. Műhelytanulmány, HU ISSN 1786-3031 – <http://edok.lib.uni-corvinus.hu/294/1/Nagy100.pdf> - Utolsó megtekintés: 2014.10.20.
- [6] Prof. Dr. Závoti József – Matematika III. 5, Nevezetes valószínűség-eloszlások – http://www.tankonyvtar.hu/hu/tartalom/tamop425/0027_MA3-5/index.html – Utolsó megtekintés: 2014.09.19.
- [7] Math.Net – <http://numerics.mathdotnet.com/> – Utolsó megtekintés: 2014.10.25.