Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Kőhalmi István

Szakterület-specifikus kártyajáték szabályrendszer létrehozása többszintű szkriptnyelvek segítségével

Tudományos Diákköri Konferencia

KONZULENS

Dr. Mezei Gergely

BUDAPEST, 2014

Tartalomjegyzék

Összefoglaló

Az utóbbi években nagyon népszerűvé vált az interaktív szórakozás területén az a lehetőség, hogy valaki saját tartalmat készíthessen. Több műfajban jelentek már meg olyan keretrendszerek, amik az embereket képessé teszi a világuk saját tetszésük szerinti újraformázására.

A dolgozat célja, hogy egy ilyen keretrendszer fejlesztésének a hátterébe nyújtson betekintést. Bár a Magic: The Gathering játék döbbenetes sikere rengeteg, egymástól merőben eltérő iteráció megjelenését eredményezte, néhány alapfogalom változatlan maradt a gyűjtögetős kártyajátékokkal (a továbbiakban CCG) kapcsolatban. Ha sikerül megtalálni egy közös szótárat, az jó alapját adhatja egy szöveges szakterületi nyelv kialakításának.

Ez a nyelv több absztrakciós réteget foglalna magába, így helyette egy olyan nyelvcsaládot mutatok be, amelynek tagjai közösen alakítják ki a végső terméket: egy játékmotort, ami képes a kártyajáték változó szabályrendszerét kezelni. Ez a nyelvcsalád deklaratív és imperatív nyelveket is tartalmaz, a játék struktúrájának és viselkedésének leírásához.

Abstract

In recent years, the ability to create one's very own content in the field of interactive entertainment has become extremely popular and sought-after. Creative frameworks related to multiple genres have been published to the public, allowing people to reshape their respective world to their own liking.

The purpose of this paper is to provide an insight into the making of one such framework. While the astounding success of *Magic: The Gathering* introduced numerous, highly distinctive entries to the collectible card game (abbreviated as CCG from now on) genre [1], several core aspects remain the same. Finding a common glossary provides a good starting point to expand upon and formulate a text-based domain-specific language.

This language spans multiple levels of abstraction, therefore it needs to be reintroduced as a set of single-tier languages that form the final product – an engine which can handle the dynamic rule set of a typical CCG game – together. This group of languages contains both declarative and imperative members describing the game's structure and behaviour respectively.

1 Bevezetés

Ebben a fejezetben röviden bemutatom a témaválasztás folyamatát, a témával kapcsolatban felmerült problémákat. Kiemelem a gyakran előforduló kifejezéseket, végül egy általános képet nyújtok a dolgozat további fejezeteiről.

1.1 A téma bemutatása

Dolgozatom témája, a szakterület-specifikus kártyajáték szabályrendszer létrehozása többszintű szkriptnyelvek segítségével, olyan területeket érint, mint a szöveges szakterületi nyelvcsalád kialakítása, nyelvfelismerés, a szakterület absztrakciós szintekre való bontása, a szintek egymáshoz való viszonyának vizsgálata, kódgenerálás, fordítás.

Első körben olyan szkriptnyelv fejlesztése volt a cél, ami leír valamilyen konkrét játékot és annak szabályait: játéktereppel, eseményekkel, céllal (példa: szkriptnyelv a *Legend of Grimrock* [2] című játékhoz). Ezzel szemben a nyelvcsalád, amit a dolgozatban bemutatok, nem egyetlen konkrét játékot ír le, hanem eszköztárat kínál arra, hogy a szakterület keretein belül ezt meg lehessen tenni.

A gyűjtögetős kártyajátékok világa, mint szakterület ehhez az elváráshoz jól illeszkedik, mert az elmúlt húsz év során kialakult egy könnyen elhatárolható alapkoncepció, amit könnyen testre lehet szabni, továbbá némileg mélyíti a feladatot, hogy a játékmenet alaptermészete a játékszabályok kikényszerített módosulása.

A megoldás során első lépésként meg kell határozni a szakterület határait. Mivel a szabályokat lehetne akár helyben rögtönözni is, megszorításokat kell tudni adni, amely keretek között a játéktervezés megvalósítható.

A nyelvnek (felismerhetően) a szakterülethez tartozónak kell lennie. Ezt az a körülmény teszi nehézkessé, hogy a gyűjtögetős kártyajátékok elvont szabályrendszerét leginkább általános célú programozási nyelvvel lehetne leírni, csak néhány kártyajáték-specifikus parancsot tartalmaz (tipikusan a kártyák mozgatásához kapcsolódóan).

A fejlesztés során felmerültek az *ANTLR* használatából kötődő megszorítások is, főleg a nyelvfelismerő eszköz generálásának módja miatt. Az *ANTLR*-ről bővebben a második fejezetben írok.

1.2 Szótár

1.2.1 Gyűjtögetős kártyajátékok

Körökre osztott stratégiai játékok, amiket előre konstruált paklival lehet játszani.

- **CCG:** collectible/customizable card game
- **Game:** a játék maga
- **Player:** játékos
- **Card:** kártya
- **Token:** olyan asztalon maradó lap, ami nem volt induláskor a pakliban, valamilyen szabály miatt jön játékba. Fizikailag megjelenhet kártyalapként, vagy más, apró helyettesítő tárgyként (kavics, korong, stb.).
- **Group:** valamilyen logika szerint összefüggő kártyák halmaza
- **Deck:** előre összeállított kártyacsoport, általában nem ismert a lapok sorrendje
- **Hand:** a kézben tartott lapok csoportja
- **Zone:** a játékterep egy kiemelt része, ami kártyák csoportosítására alkalmas
- **Attribute:** a játék állapotát határozzák meg
- **Turn:** a játék időbeosztásának alapegysége, a játékosok felváltva cselekedhetnek
- **Phase:** a kör több részre osztható, bennük más-más cselekvések vannak megengedve, a sorrendjük kötött
- **Draw:** húzás
- **Discard:** lapdobás
- **Roll:** kockadobás
- **Play:** kijátszás

1.2.2 Szakterületi nyelvek

- **DSL:** Domain-specific language
- **Grammar:** adott szöveg-alapú bemenetre meg lehet mondani, hogy egy adott nyelvre illeszkedik-e a nyelvtani szabályok alapján
- **Terminal symbol:** a formális nyelvtan illesztése során használt elem, a szintaktikailag azonos tokenek osztálya.
- **Nonterminal symbol:** a formális nyelvtan illesztése során használt elem, szintaktikailag azonos csoportosításokat jelölő osztály.
- **Token:** olyan karaktersorozat, amely a nyelv szintaktikája szerint önálló jelentéssel rendelkezik.
- **Lexer:** a beolvasott karaktersorozatot értelmezhető részekre (token) bontja.
- **Parser:** a feladata a token-sorozat és a nyelvtani szabályok alapján felépíteni a forrás szintaktikai struktúráját, a konkrét szintaxisfát (CST), majd abból a feldolgozás szempontjából érdektelen elemek eltávolításával előállítani az absztrakt szintaxisfát.
- **LL(*) parser:** a szabályokat fentről lefelé próbálja illeszteni, a szabályon belül a nemterminális szimbólumok behelyettesítését mindig balról jobbra végzi. A parsernek nincs szüksége véges számú token előismeretére ahhoz, hogy dönteni tudjon.
- **AST:** abstract syntax tree – a forrás szintaktikai konstrukcióját reprezentáló gráf, ebben a fában már csak a modellalkotáshoz elengedhetetlen nyelvi elemek jelennek meg.
- **Compiler:** karakterlánc transzformációt végez, a forrásnyelv általában magasabb absztrakciós szinten használatos, mint a célnyelv.

1.3 A dolgozat tartalma

A második fejezetben a feladat részletesebb ismertetése a cél, bemutatom a szakterületet és az ahhoz kötődő elterjedt kifejezéseket, ezt követi egy rövid ismertető az *ANTLR* nyelvfelismerő eszköztárról és a felhasználásának módjáról. A szakterületi

nyelvet átfogó nyelvcsalád bemutatását követően az elkészült szintaxisfa forráskóddá alakításának elméleti hátteréről írok.

A harmadik fejezetben a nyelvek feldolgozásának részletes, hierarchikus leírása olvasható.

A negyedik fejezetben a tervezett játékszerkesztő rendszer grafikus kezelőfelülete áll a középpontban.

Az ötödik fejezetben egy már létező példajáték segítségével mutatom be a rendszer komponenseinek együttes működését.

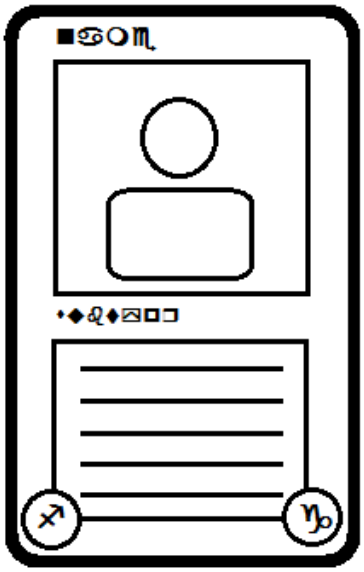
A hatodik fejezetben összesítem az elért eredményt és felvázolom a lehetséges továbbfejlesztési irányokat.

2 A feladat kifejtése

Ebben a fejezetben részletesebb képet festek a feladatról és a felmerülő problémák megoldásáról.

2.1 Gyűjtögetős kártyajátékok, mint szakterület

Mikor nevezhetünk egy kártyajátékot CCG-nek? Általában véve ezeknek hasonló a megjelenésük, stílusuk: a kártyalapok előlapján illusztráció látható, a lap tetején a kártya neve olvasható, opcionálisan megjelenhet a kártya szabálymódosítását taglaló szöveg, kiegészítve színesítő szöveggel. Ezen felül különböző statisztikák és szimbólumok osztályozhatják a kártyát. Az egy pakliba tartozó lapok hátoldala mindig egyforma. Elvárt tulajdonság a gyűjthetőség és cserélhetőség biztosítása (kártyák csoportosítása szettekbe megjelenésük ideje alapján), valamint az, hogy a játékszabályok stratégiai játékélményt tudjanak biztosítani (nem szerencsejátékok) [1][4].



1. ábra Általános CCG kártyalap

Az első CCG-t Richard Garfield tervezte és 1993-ban látott napvilágot a *Wizards of the Coast* gondozásában, *Magic: The Gathering* néven. A játék keresztezte a baseball-kártyák gyűjtési ösztönét a kártyajátékok könnyű terjedési képességével. A játék ezen felül tartalmazta a szabályok változtatásának lehetőségét, amit a *Cosmic Encounter* című társasjáték inspirált [1].

A játékot azóta rengeteg új iteráció megjelenése követte, amelyek más és más játékmechanikával operálnak. Megmaradt azonban néhány központi fogalom, amit nagyon nehéz lenne lecserélni úgy, hogy még mindig CCG-ről legyen szó. A játékok alapeszköze a kártya, így az olyan cselekedeteket, amik egy kártyalap helyzetét befolyásolja, a szakterület folyamataként vehetjük számba.

A kártyalapoknak két oldala van, a hátoldaluk azonos, hogy a pakliban ne lehessen felismerni az egyes lapok helyzetét. A lapokat tehát lehet fel- és lefordítani, valamint 4 irányba forgatni (ennél több helyzet nem jellemző). Ezen kívül a kártyákat csoportokba lehet foglalni, a csoportok között mozgatni lehet őket. Ez utóbbinak speciális esetei a laphúzás, ami a pakli legfelső lapját a kézbe helyezi; a lapdobás, amely egy lapot a játéktéren kívülre helyez; valamint a kijátszás, ami a kézből az asztalra mozgat egy lapot, valamelyik kitüntetett csoportba.

A játékosok viszonya a lapokhoz kétrétű: egyrészt a lapnak lehet a tulajdonosa, másrészt lehet az irányítója. A tulajdonos az, akinek a paklijából származik a lap, az irányító pedig az, aki használhatja. Az egyes lapok láthatóságát (melyik oldala van felül) elsősorban a tulajdonosi viszony határozza meg. Az asztalon lévő lapokat általában mindenki látja, a paklikban lévőket senki. A kézben lévő lapokat a tulajdonos látja. Ezt írhatja felül az irányító viszony, mivel az irányító akár felveheti a kezébe is a kártyát, így takarva azt el a tulajdonosától.

A játékosokhoz és kártyákhoz lehet tulajdonságokat rendelni, aktuális értékük írják le a játék állapotát. Gyakran előforduló játékos tulajdonságok az életpontok száma, a különböző rendelkezésre álló erőforrások értéke, állapotok megléte. Kártyákhoz kötődő tulajdonság a kártya neve, kijátszási költsége, típusa, altípusa, egyéb statisztikák. A példák alapján látszik, hogy a tulajdonságok típusa többféle lehet, ezeket a következő fejezetben tisztázom.

A játékmenet körökre van osztva. A játékosok a saját körükben cselekednek, amik felváltva követik egymást. Egy kör több fázisból áll, minden fázisnak megvan a maga szerepe a játék dinamikáját illetően. Nagyon gyakori fázis a húzás fázisa a játékos körének elején, valamint az erőforrások frissítésének fázisa. A fázisok sorrendje kötött.

A játéknak mindig van(nak) győzelmi feltétele(i), ami(ke)t általában valamelyik attribútum értékéhez kötnek, legyen az adott számú győzelmi pont elérése, az életpontok elvesztése, stb. Ehhez lazán kapcsolódnak olyan események, amelyek azonnali vereséget

jelentenek, például elfogynak a lapok a pakliból. Lehetőséget kell adni a döntetlenben való megállapodásra is.

Az előbbiek tehát azokat a kifejezéseket tartalmazzák, amik a CCG-kkel kapcsolatosan gyakran felmerülnek. Ha csupán ezekre hagyatkoznék, a keretrendszer nem lenne kivitelezhető, ugyanis a CCG-k természetüknél fogva óriási tervezői szabadságot engednek a szabályrendszerek tekintetében. Ennek fékezésére néhány megkötést kellett bevezetnem, hogy a játékok tervezése kezelhető keretek között menjen végbe:

- pontosan két játékos van
- a játék szimmetrikus a játékosok indulóállapotát tekintve
- a játékosoknak nincs kitüntetett szerepe (pl. az egyik csak támad, a másik csak véd)
- a győzelmi kondíciók mindkét játékos számára azonosak
- a fázisok a játékosok körén belül váltják egymást (vannak olyan CCG-k, ahol a fázisokon belül váltják egymást a játékosok)
- nincs lehetőség az ellenfél körében cselekedni

A,B: cselekvő játékos	Számok: fázis sorszáma											
Körökön belül fázisok	A1	A2	A3	B1	B2	B3	A1	A2	A3	B1	B2	B3
Fázison belül körök	A1	B1	A2	B2	A3	B3	A1	B1	A2	B2	A3	B3

2. ábra A kétféle ütemezés összehasonlítása egy háromfázisú játék esetén

2.2 Nyelvfelismerés

Az előző alpontban bemutatott fogalmak alapján előállítható a CCG szakterületi nyelve. Munkám során gyorsan kiderült, hogy egyetlen nyelv nem tudja az összes absztrakciós szintet egyszerre kezelni, minden szinthez és felhasználási területhez saját nyelvet célszerű rendelni.

A következő alpontokban röviden beszélek az ANTLR eszköztárról, majd az egyes nyelvek felelősségét és használatuk módját ismertetem.

2.2.1 ANTLR

Az ANTLR (ANother Tool for Language Recognition) olyan eszköz, aminek a segítségével szöveg-alapú szakterületi nyelveket lehet építeni: a nyelvtanuk alapján felismerő és feldolgozó osztályokat generál adott célnyelven, amit fordítás, vagy elemzés során fel lehet használni [5]. Az eszköz karbantartója Terence Parr, a University of San Francisco professzora.

A nyelvtant kétféle szabályból lehet felépíteni. A lexer szabályok neve nagybetűvel kezdődik és egyetlen terminális szimbólumot ad vissza. Az alábbi lexer szabály egész számokat ismer fel és INT típusú terminális szimbólumként tér vissza:

```
INT: '0'|('-')?(('1'..'9')('0'..'9')*);
```

A parser szabályok neve kisbetűvel kezdődik és nyelvi szimbólumok szintaktikailag összefüggő csoportját írják le. A parser szabályokhoz rendelhető újraírási szabály, ami a generált absztrakt szintaxisfát módosítja úgy, hogy könnyebben feldolgozható legyen (saját hierarchia jelölhető ki). Az alábbi rekurzív parser szabály olyan láncot keres, aminek az elemei elején pont áll, amit egy alfanumerikus szó követ:

```
path: ('.'head=ID)(rest=path)?
```

Egy ANTLR nyelvtan tartalmazhat csak lexer, vagy csak parser szabályokat. Ha mindkét típus megtalálható, a parser szabályoknak meg kell előzni a lexer szabályokat. A nyelvtannak kell tartalmaznia a parser generátor kimeneti nyelvét.

Az ANTLR LL reguláris parsert (LL(*)) alkalmaz [3], ami a nyelvtan kialakítása során két főbb megkötést von maga után:

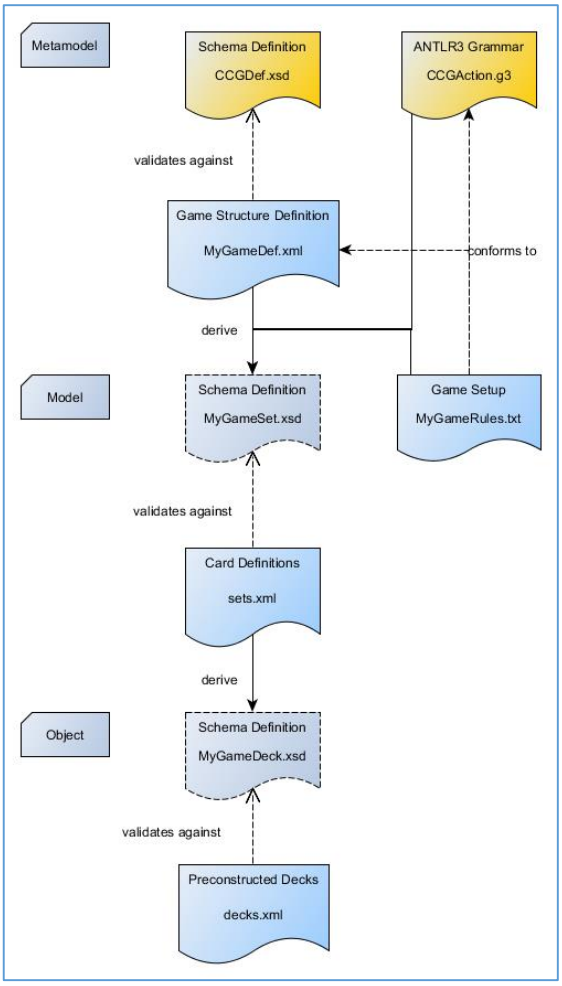
- A mintaillesztés a nyelvtan első szabályával indul és sorban próbálkozik a többivel. Ha az illesztés sikeres, a többi szabállyal már nem foglalkozik. Érdemes tehát a „megengedőbb” jellegű szabályokat későbbre halasztani, ellenkező esetben ugyanis egy ilyen általános szabály eltakarhat egy olyat, ami szigorúbb és igazából szükségünk lenne rá.
- Kerülni kell a balrekurzív szabályok használatát, ugyanis ezek a végtelen ciklus veszélyét hordozzák a feldolgozás során. Balrekurzív szabály állhat önmagában, vagy előfordulhat több, körkörösen egymásra hivatkozó szabály esetén.

2.2.2 Meta-adatok

A meta-adatok olyan adatok, amelyek információt szolgáltatnak a játék készítésével kapcsolatban: cím, készítő, leírás. A játék működését ezek az adatok nem befolyásolják. Az adatokat a struktúrát leíró *CCGDef* sémának megfelelő XML nyelven lehet megadni.

2.2.3 A CCG nyelvcsalád

A nyelvcsalád három absztrakciós szintet ölel át, a magasabb szintű nyelvek definiálják azokat az elemeket, amiket az alacsonyabb szintűek felhasználnak. A nyelvcsaládot alkotó nyelvtanokat és azok kapcsolatát a 3. ábra mutatja. Sárga háttere van a keretrendszer részét képező nyelvtanoknak, világoskék háttere van a felhasználó szerkesztése hatására generált dokumentumoknak és sötétkék a háttere az implicit nyelvtanoknak.



3. ábra A CCG nyelvcsalád tagjai

2.2.4 A struktúra meta-modellje

A meta-modell azt adja meg, hogy milyen építőegységeket használhatunk fel a szerkesztés során és azokat hogyan tudjuk egymáshoz kapcsolni. Statikus szempontból ide tartoznak a játékosok, a tulajdonságok, a különböző kártyacsoportok. A meta-adatokkal együtt ezeket az elemeket felhasználva kialakítható a játék struktúrája, amit a *CCGDef* séma definícióval lehet validálni. A séma részletes ismertetése a következő fejezetben olvasható. Általánosságban elmondható, hogy a struktúrát leíró nyelvek XML alapúak, a feldolgozásuk *LINQ to XML* segítségével történik.

2.2.5 A folyamatok meta-modellje

A játék dinamikus leírásához a kártyajáték-specifikus cselekvések mellett olyan általános célú nyelvi elemekre is szükség lesz, mint a vezérlési szerkezetek, operátorok, operandusok, primitív típusok, változók. A nyelvtan, ami ezeket az elemeket tartalmazza a *CCGAction* nevet kapta és ANTLR3 nyelven van definiálva.

2.2.6 A kártyák listája

A *CCGDef* sémára illeszkedő példány XML és a *CCGAction* nyelvtan felhasználásával olyan séma készíthető, ami a játék kártyáinak felvételére alkalmas. Ez a séma tartalmazza a közös és a játék-specifikus attribútumokat. Az attribútumok értékét itt lehet kitölteni. Itt van lehetőség a kártya viselkedésének tisztázására is CCGAction nyelven. Az így feltöltött listát végül ki lehet íratni a sémára illeszkedő XML fájlba.

2.2.7 Pakli szerkesztés

Az előző pontban leírt lista alapján készülhetnek el az előre összeállított paklik. A kártyák neve mellé csak annyi információ kerül, hogy a pakli mennyi darabot tartalmaz belőlük. A paklik összetételét is XML formátumban lehet elmenteni.

2.3 Kódgenerálás

Feltételezve, hogy a *CCGAction* parsere nem talált hibát a szkriptnyelven írt utasítások feldolgozása során, a kimeneten megjelenő absztrakt szintaxisfa alapján C# nyelvű kódot kell tudnia a rendszernek generálni.

Ezt a feladatot egy erre a célra felépített osztály végzi, az AST gyökéreleme, valamint egy segédfa (kontextusfa) aktuális eleme alapján. A kontextusfa feladata tárolni

a feldolgozás láthatósági körében elérhető változók és műveletek neveit, típusát, az esetleges paraméterek számát és típusát. A kontextusfa követi az absztrakt szintaxisfát, de nem a másolata annak. Az AST legtöbb eleme a kontextus szempontjából érdektelen, csak a vezérlési szerkezetek és névdeklarációk jelennek meg benne. Névismétlés esetén csak a legutolsó fog látszódni, amíg a láthatóság hatásköre nem lép vissza az általánosabb szintre.

A kódgenerálás során az AST aktuális eleme alapján egy kódrészlet készül, ami kitölthető a gyerekelemek végigjárásával. A kontextusfát folyamatosan karbantartva lehet figyelni a típushelyességre, értelmezhetőségre (pl. a predikátum biztosan logikai típusú eredményt adna-e). Ha semmilyen hiba nem történt a feldolgozás során, a kódrészlet elfogadásra kerül, egyébként olyan kód kerül a helyére, ami jelzi a hibát.

3 Részletes megvalósítás

Ebben a fejezetben bemutatom a CCG nyelvcsalád tagjait és részletesen ismertetem azok feldolgozási módját.

3.1 A szerkezetet leíró nyelvek részletezése

3.1.1 LINQ to XML

A LINQ to XML olyan XML programozási interfész, amely lehetővé teszi XML dokumentumok menedzselését .NET környezetben, kiegészítve a LINQ adatkezelő technológia funkcióival. Ez a könyvtár az alábbi műveletek elvégzését támogatja [6]:

- XML fájl/folyam beolvasása memóriába
- XML sorosítása fájlba/folyamba
- XML létrehozása programozott módon
- Lekérdezés futtatása XML-en belül
- A memóriában lévő XML módosítása
- XML validációja sémadefiníció alapján

3.1.2 CCGDef

A legfelső absztrakciós szinten helyezkedik el a *CCGDef* XML séma definíció. Ez a nyelv adja meg azt a keretet, amire a többi statikus nyelv támaszkodik. Ezen a nyelven van leírva az, hogy milyen módon vehető fel egy új kártyacsoport, attribútum, vagy fázis a játékhoz. A *CCGDef* séma definíció struktúrája a függelékben található.

3.1.2.1 A játék definíció

A dokumentum gyökéreleme (*gamedef*), amiben a következő meta-adatokat adhatjuk meg: a játék címe (*title*), készítője (*creator*) és rövid leírása (*description*). Mind string típusú elem.

```
<gameDef>
  <title>Proba jatek</title>
  <creator>Kohalmi Istvan</creator>
  <description>Pelda jatek bemutatasa</description>
```

4. ábra Példa meta-adatokra

Ezeken túl a soron következő elemek használata engedélyezett a leírásban.

3.1.2.2 Csoportok (*groups*)

Ebben az elemben kerül sor a kártyacsoportok (*group*) felsorolására. A *group* összetett típusú elem, az alábbi elemeket tartalmazza:

- *gmode*: kötött string típusú, értéke lehet: „private”, vagy „shared”
- *gtype*: kötött string típusú, értéke lehet: „deck”, „hand”, vagy „zone”
- *tag*: string típusú, a játék leírásában ezzel a rövid névvel lehet rá hivatkozni
- *name*: string típusú, a felhasználó ezt a nevét látja a csoportnak a felületen
- *minCards*: int típusú, legkevesebb ennyi kártya van benne induláskor (pakli mentésekor fontos)
- *maxCards*: int típusú, ennél több lap nem lehet egyszerre a csoportban
- *visibleBy*: kötött string típusú, értéke lehet: „player”, „none” és „both”

```
<group>
  <gmode>private</gmode>
  <gtype>zone</gtype>
  <tag>VNG</tag>
  <name>Vanguard</name>
  <maxCards>5</maxCards>
  <visibleBy>both</visibleBy>
</group>
```

5. ábra Példa egy csoport definíciójára

3.1.2.3 Tulajdonságok (attribute) típusai

Minden tulajdonság négy típus valamelyikébe sorolható be:

- Szám (*number*): gyűjtögetős kártyajátékok esetén csak egész számokra van szükség, a legtöbb esetben ezek nemnegatívak
- Szöveg (*string*): karakterlánc, két idézőjel között

- Értékkészlet (*token*): több *string* közül lehet választani, akár többet is. Tipikus felhasználás lehet a különböző kártyatípusok/altípusok megadása.
- Logikai (*bool*): egy állapot meglétét jelzi

3.1.2.4 Játékosok (*player*)

Ebben az elemben lehet felsorolni a játékosok tulajdonságait (*playerAttr*). A *playerAttr* összetett típusú elem, az alábbi elemekből áll össze:

- *aType*: kötött string típusú, a tulajdonság típusát adja meg (ld. 3.1.2.3)
- *tag*: string típusú, a játékban ezzel a rövid névvel lehet rá hivatkozni
- *name*: string típusú, a felhasználó ezt a nevét látja az attribútumnak a felületen
- *default*: string típusú, opcionális elem, az attribútum alapértelmezett értékét tárolja
- *minValue*: byte típusú, a számalapú attribútum nem vehet fel ennél alacsonyabb értéket
- *maxValue*: byte típusú, a számalapú attribútum nem vehet fel ennél magasabb értéket
- *values*: string típusú elemek (*choice*) felsorolása

3.1.2.5 Kártyák (*card*)

Ebben az elemben lehet felsorolni a kártyák attribútumait (*cardAttr*). A *cardAttr* összetett típusú elem, az alábbi elemekből áll össze:

- *aType*: kötött string típusú, a tulajdonság típusát adja meg (ld. 3.1.2.3)
- *tag*: string típusú, a játékban ezzel a rövid névvel lehet rá hivatkozni
- *name*: string típusú, a felhasználó ezt a nevét látja az attribútumnak a felületen
- *default*: string típusú, opcionális elem, az attribútum alapértelmezett értékét tárolja
- *minValue*: byte típusú, a számalapú attribútum nem vehet fel ennél alacsonyabb értéket

- *maxValue*: byte típusú, a számalapú attribútum nem vehet fel ennél magasabb értéket
- *values*: string típusú elemek (*choice*) felsorolása

```
<cardAttr>
  <atype>token</atype>
  <tag>TYPE</tag>
  <name>Type</name>
  <default></default>
  <minValue>0</minValue>
  <maxValue>0</maxValue>
  <values>
    <choice>Monster</choice>
    <choice>Item</choice>
    <choice>Spell</choice>
  </values>
</cardAttr>
```

6. ábra Példa egy attribútum definiálására

3.1.2.6 Fázisok (*phases*)

Az fázis elemek (*phase*) felvételi sorrendjétől függ a játékon belüli sorrendjük. A *phase* összetett típusú elem és az alábbi részekből áll:

- *tag*: string típusú, a játékban ezzel a rövid névvel lehet rá hivatkozni
- *name*: string típusú, a felhasználó ezt a nevét látja a fázisnak a felületen

```
<phase>
  <tag>DRAW</tag>
  <name>Draw</name>
</phase>
```

7. ábra Példa fázis deklarálására

3.1.3 MyGameSet

Ez a séma írja le a játékhoz tartozó lapok leírását, különböző szettekbe rendezve, ha ilyen csoportosítás szükségesnek bizonyul. Ez a séma implicit abban a tekintetben, hogy a szerkesztő által generált szetteket tartalmazó XML a példányának tekinthető, de maga a séma nem jön létre. Ettől függetlenül van néhány olyan elem, amit az összes ilyen XML tartalmaz, ezeket az elemeket mutatom be a továbbiakban:

3.1.3.1 *SETS*

A szett séma gyökéreleme.

3.1.3.2 *SET*

A szett nevén kívül (*SNAME*) kártyaelemek (*CARD*) felsorolását tartalmazza.

3.1.3.3 *CARD*

Összetett típus, a keretrendszer alapján az alábbi elemek alkotják:

- *NAME*: string típusú, a lap nevét jelöli
- *PLAY:* string típusú, a lap kijátszásakor végrehajtandó műveleteket tartalmazza *CCGAction* nyelven, opcionális kitölteni
- *DISCARD:* string típusú, a lap eldobásakor végrehajtandó műveleteket tartalmazza *CCGAction* nyelven, opcionális kitölteni

Ezeken kívül itt van lehetőség megadni az egyes attribútumok értékét, az attribútumok jelölőnevét (*CCGDef*-ben tag) használva itt XML tag-ként.

```
<CARD>
  <NAME>Elder Sign</NAME>
  <COST>0</COST>
  <ATK>0</ATK>
  <PLAY>ME.INV SET true ME.ED+=2</PLAY>
  <DISCARD></DISCARD>
</CARD>
```

8. ábra Példa egy lap definíciójára, a *The Necronomicon* című CCG-ből

3.1.4 MyGameDeck

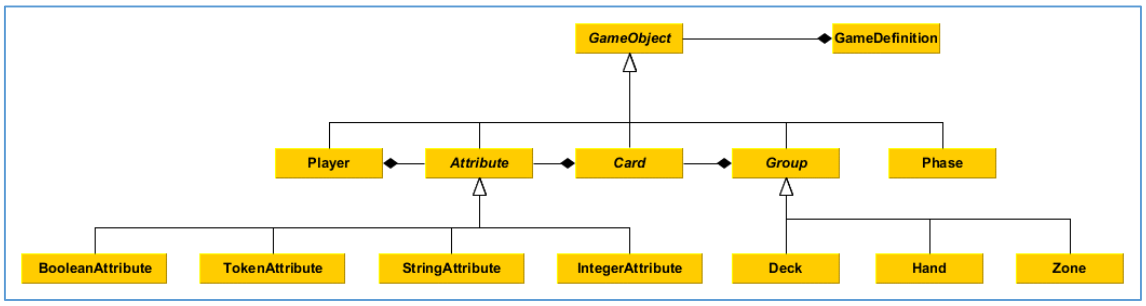
Ez a nyelv csak annyiban különbözik a szett sémától, hogy *SET* helyett *DECK* van benne, és a *CARD* elemek két gyereket tartalmaznak: *NAME* és *COUNT*.

3.1.5 A játékdefiníció modellje

A szerkesztő saját modellt épít a felhasználó hatására. Ahhoz, hogy a kimentett XML fájlok betölthetők és szerkeszthetők legyenek később, szükség van olyan C# osztályokra, amelyek ezt a modellt megvalósítják.

A játékban minden osztály közös őse a *GameObject* absztrakt osztály, ami vár egy nevet és egy jelölést. Ezen kívül lehetőséget biztosít arra, hogy a felhasználó saját akciót rendeljen minden leszármazott objektumhoz. A modell egyszerűsített osztálydiagramját mutatja a 9. ábra.

A szett XML-ből generálandóak majd a *Card* osztály leszármazottai, így mindegyik lap a saját implementációja szerint fog majd működni.



9. ábra A játékdefiníció egyszerűsített osztálydiagramja

3.2 A folyamatokat leíró nyelvek részletezése

Ebben az alfejezetben a játék viselkedését meghatározó nyelvtannal (*CCGAction*) és az általa definiált nyelv felhasználási módjaival foglalkozom.

3.2.1 CCGAction

A nyelvtant ANTLR3 környezetben építettem, a nyelvfelismerő osztályokat C# nyelven generálja a rendszer. A nyelv magas absztrakciós szintű, jellegét tekintve imperatív. Tartalmaz általános célú nyelvi elemeket, illetve szakirány-specifikus kifejezéseket is.

3.2.1.1 Parser szabályok

- **Akció (*action*)**

Leginkább a kódblokkhoz hasonlítható. Két részre lehet tagolni. Az első rész az esetleges változók deklarálásáért felel (*declaration*), a második a parancsokat tartalmazza (*command*).

- **Változó deklarálása (declaration)**

Egy változót deklarál az aktuális blokk hatáskörében. Meg kell adni a változó nevét (*tag*) és típusát (*Type*).

- **Név (*tag*)**

Egy erőforrás nevét takarja, két részből áll. Az első a tulajdonos megjelölése:

  - ME: a játékos maga
  - OPP: a játékos ellenfele

- SHR: közös erőforrás

A második rész az erőforrás elérési útvonala (*path*).

- **Elérési útvonal (path)**

Egy lánc, ami megadja egy adatelem elérési útvonalát: .nev1.nev2.nev3… formában.

- **Parancs (*command*)**

Gyűjtőfogalom, a helyén állhat:

- vezérlési szerkezet (*controlCommand*)
- módosító kifejezés (*modExpression*)
- üres sor (*EMPTY*)
- felhasználó által definiált parancs (*userDefined*)
- kártyát érintő parancs (*cardCommand*)
- csoportot érintő parancs (*groupCommand*)

- **Vezérlési szerkezet (*controlCommand*)**

A következő vezérlési szerkezeteket tartalmazza a nyelv:

- SWITCH: paramétere egy szám típusú változó, a paraméter értékétől függően viselkedik úgy, mint a C# switch-e.
- RANDOM: a kockadobás szimulálására, a szerepe hasonló a SWITCH-éhez, ezúttal azonban a kapott paraméter a felső határt adja meg egy véletlen szám generálásához, amely szám a SWITCH szerkezet változójául szolgál.
- ?(pred.):*ifAction*:*elseAction*: egyszerű döntési szerkezet. A predikátumnak logikai típusúnak kell lennie, különben a kifejezés értelmezhetetlen marad.
- FOREACH: paraméterül egy csoport nevét kapja, aminek összes lapjára elvégzi a saját blokkjában lévő műveleteket, a futó változóra a ’card’ szóval lehet hivatkozni.

- **Módosító kifejezés (*modExpression*)**

Olyan operátorok kerülnek ide, amik baloldali operandusuk (*leftOperand*) értékét befolyásolni tudják. Kétféle típusuk van: két- (*ModOperator*), illetve egyváltozós (*ModUnaryOperator*). A kétváltozós típus második operandusa (*operand*) nem változik.

- **Baloperandus (*leftOperand*)**

Ezek az operandusok csakis névvel (*tag*) rendelkező objektumok lehetnek.

- **Operandus (*operand*)**

Ezek olyan kifejezések, amelyeknek kötött típusa van és az értékük nem változhat az operátor hatására. Ilyen operandusok a különböző literálok (szám, szöveg, lista, logikai érték), változók neve, illetve olyan összetett kifejezések (*mildExpression*), amik nem tartalmaznak módosító operátort.

- **Kiértékelő kifejezés (*mildExpression*)**

Ezek a kifejezések egyik operandusuk (*operand*) értékét sem változtathatják meg. Az egymásba ágyazott kifejezéseknek minden szinten típushelyesnek kell lenniük. Ezeknek is van két- (MildOperator) és egyváltozós (MildUnaryOperator) változata.

- **Saját műveletek (*userDefined*)**

Ezeket a műveleteket a tervező definiálja (*actionDefinition*) a játékszabályok létrehozása során. A feldolgozás során figyelni kell arra, hogy a paraméterek száma és típusa egyezik-e az előírttal.

- **Művelet definiálása (*actionDefinition*)**

Meg kell adni a művelet nevét, a paraméterek (*parameter*) nevét (*tag*) és típusát (*Type*), végül magát a cselekményt (*action*).

- **Kártya utasítás (*cardCommand*)**

A kártyák mozgatásával kapcsolatos parancsok gyűjtőneve. Ezek a parancsok lehetnek:

  - Forgatás: UP, DOWN, LEFT, RIGHT

- Fordítás: FACEUP, FACEDOWN
- Mozgatás: MOVETO, DISCARD, PLAY, ATTACHTO

Mindegyik legalább egy kártyára (*cardOperand*) hivatkozik.

- **Kártya megadása (*cardOperand*)**

Kártyát meg lehet adni a létrehozásakor generált azonosítójával, vagy egy csoportban elfoglalt helyével. Ezen kívül lehetőség van a „card” kulcsszóval hivatkozni egy FOREACH ciklus futó változójára. Végül adott csoport valamelyik kártyáját kiválaszttathatjuk bármelyik játékossal (PICK, FORCEPICK), vagy választhatunk egyet véletlenszerűen (RANDOMPICK).

- **Csoport utasítás (*groupCommand*)**

Kártyacsoportokhoz köthető utasítások tartoznak ehhez a szabályhoz:

- DRAWTO: kiválasztja azt a csoportot, ahova a pakliból húzni kell
- DISCARDTO: kiválasztja azt a csoportot, ahova a játékból kikerülő lapok kerülnek
- DRAW: húz egy lapot a pakliból
- DRAWMANY: paraméterként kapja azt, hogy mennyi lapot kell húzni
- SHUFFLE: megkeveri a lapokat a csoportban

Mindegyik használatához legalább egy csoportot (*groupOperand*) meg kell tudni adni.

- **Csoport megadása (*groupOperand*)**

Kétféle módon lehet hivatkozni egy csoportra: a nevével, vagy egy kártyalapnak ismerjük az azonosítóját és a befoglaló csoportot keressük.

3.2.1.2 Lexer szabályok

- **Változó típusok (*Type*)**

A változók típusát jelöli, értéke lehet: number, string, token, boolean.

- **Kétváltozós módosító operátor (*ModOperator*)**

Olyan operátorok, amik módosíthatják a bal operandusukat. Ide tartoznak az aritmetikai műveletek (+=,-=,*=,/=), az értékadás (SET), listakezelő parancsok (ADD, REMOVE, SELECT, UNSELECT).

- **Egyváltozós módosító operátor (*ModUnaryOperator*)**

Egyváltozós operátorok, amik módosíthatják az operandusuk értékét. Ide tartozik a végleges logikai negálás (NEGATE), valamint az inkrementálás (INC, DEC).

- **Kiértékelő kétváltozós operátor (*MildOperator*)**

Ezek az operátorok nem változtatják meg egyik operandusukat sem, csupán kiértékelés a feladatuk. Itt is találhatóak aritmetikai műveletek (+,-,/,*), komparáló operátorok (=,<>,<,<=,>,>=, MATCH), logikai operátorok (AND, OR).

- **Kiértékelő kétváltozós operátor (*UnaryOperator*)**

Olyan egyváltozós operátorok, amik csak kiértékelésre használatosak. A felismert unáris operátorok a következők:

  - NOT – a logikai operandus negáltját adja vissza
  - COUNT – a csoport operandus kártyáinak számát adja vissza
  - ROLL – kockadobás, az operandus a maximális értéket adja meg.

- **Megnevezés (*ID*)**

Nevet jelöl, az első karaktere csak betű, vagy aláhúzás lehet, a többi betű, számjegy, vagy aláhúzás.

- **Egész szám (*INT*)**

Egész szám literál megadására alkalmas.

- **Zárójelek (*LPAR, RPAR*)**
- **Szögletes zárójelek (*LBR, RBR*)**
- **Szövegliterál (*STRING*)**

Bármit elfogad két idézőjel között.

- **Feldolgozást nem igénylő parancs (*EMPTY*)**

Egyetlen pontosvessző

- **Szünet (*WS*)**

A whitespace karaktereket foglalja magába. Nyelvi szinten lehetőség van ezeket elrejteni, így a parser már nem foglalkozik velük. A *CCGAction* nyelvtan nem alkalmaz kitüntetett delimiter karaktert az utasítások között, bármilyen whitespace karakter megteszi.

3.2.2 A nyelv felhasználási módjai

A CCGAction nyelv két helyen kerül felhasználásra. Egyrészt a játékszabályok leírásánál, ami magában foglalja a felhasználó saját műveleteinek definiálását, a játékterep előkészítését, a kezdeti értékek beállítását. Ezeken túl itt vannak leírva részletesen a fázisok működése is.

A másik eset az, amikor a szetteken belül felvesszük a kártyák saját viselkedését is az attribútumaik mellett. Itt már fel kell tudni ismerni a felhasználó által definiált parancsokat is.

3.3 Kódgenerálás

Ebben az alpontban a fordítás menetének részletezésével foglalkozom. A kódgenerálás célnyelve mindvégig C#.

3.3.1 A kontextusfa, mint segédeszköz

A *CCGAction* nyelven írt szöveg olvasása közben találkozunk egy objektum nevével, amiről el kell tudnunk dönteni, hogy a vezérlés aktuális hatáskörében a név ismert-e egyáltalán, milyen típusú objektumot címez, ha igen, illetve hogyan lehet az értékét előállítani.

A fordítás során az *ANTLR* által generált parser kimenetét, az absztrakt szintaxisfát járjuk be. Az egyes csomópontokat, mint szövegsablont alkalmazva, a hézagokat a levelek bejárásával lehet kitölteni. Csak akkor szabad a kitöltött sablont elfogadni, ha a hierarchia minden szintjén a típushelyesség fennáll.

A fenti két felmerülő problémát orvosolja a kontextusfa alkalmazása, ami követi a feldolgozott AST csomópontban a vezérlés scope-ját és visszamenőlegesen frissíti a

vizsgált kifejezés típusát minden bizonytalan szinten akkor, ha az egy mélyebb szinten egyértelművé vált (pl. egy összetett kifejezés feldolgozása során egy literált találunk).

3.3.2 A kontextusfa építőelemei

A kontextusfa csomópontjait a *CCGActionContext* osztály példányai alkotják. Ennek az osztálynak három mezője van, továbbá egy referenciája a szülő csomópontra és egy listája a gyermek csomópontok referenciáihoz. Az elemek fa struktúrába való rendezése biztosítja azt, hogy a vezérlés és a láthatóság hatásköre felsőbb szintek felé érvényes legyen, de a testvérek már ne lássák egymás deklarációit.

3.3.2.1 ContextType

Az első mező a kontextus típusának (*ContextType*) kiválasztása, ami az AST-hez hasonlóan különböző tokenek hierarchikus kapcsolatát építi fel. A fa gyökerében a kontextus típus a *ROOT* értéket veszi fel, ez az egyetlen, aminek nincs szülő referenciája. Mivel a blokkok felépítése olyan, hogy a változók deklarálása megelőzi a parancsokat, az AST-ben korábban szerepelnek, így azzal soha nem kell foglalkozni, hogy esetleg olyan nevet találunk, ami még nem ismert.

A rendszer az alábbi kontextus típusokat képes kezelni:

NONE, ACTION, CASE, DECLARATION, DEFINITION, USERFUNCTION, EXPRESSION, FOREACH, IF, LEFTOP, MODIFICATION, OPERAND, PARAMETER, PREDICATE, ROOT, SWITCH, VARIABLE

3.3.2.2 VariableType

Egy másik mezőben van lehetőség a változó típusának megadására deklarálás során, vagy az összetett kifejezés típusának kikövetkeztetésére a felismert változónevek, literálok alapján.

Ha nem deklarációról, vagy literálról van szó, az újonnan létrehozott kontextus csomópontja a *NONE* változótípust kapja. Ez az érték azt jelenti, hogy a kifejezés típusát egyelőre nem lehetett kiértékelni. Ha az alsóbb szinten valahol egyértelművé válik egy operandus típusa, az a saját típusát beállítja, majd amikor a szülőhöz visszakerül a vezérlés, leellenőrizheti, hogy a gyerekének a típusa a környezetéhez illő-e. Ha minden gyerekelem típusa megfelelő, az általa épített kódrészlet elfogadásra kerül és

beépíthetővé válik az ő szülőjének a kódrészletébe. Ellenkező esetben valamilyen hibajelzésre kerül sor.

A rendszer az alábbi változó típusokat fogadja el:

NONE, BOOLEAN, CARD, GROUP, NUMBER, PHASE, STRING, TOKEN

3.3.2.3 A változó neve

Az utolsó mező a változó neve, amire majd keresni lehet az összes leszármazottból. A névre a rekurzív *LookupVariable(name:string)* metódus segítségével lehet rákeresni, ami a változó típusával tér vissza, ha az deklarálva van és *NONE*-nal, ha nem. A metódus a kiindulási csomópontot vizsgálja, és a közvetlen gyerekeit. Ha ilyen névvel nem talál deklarált változót, a metódus a szülő csomóponton meghívja önmagát, így előbb-utóbb eljut a gyökérelemhez. Ha a gyökér csomópontban sem található a változó a visszatérési érték *NONE*.

3.3.3 A játékszabályok fordítása

A játékszabályok fordításához először létre kell hozni egy új kontextusfát. Ezt követi a kontextusfa inicializálása, ami a játékdefiníció alapján feltölti az összes elérhető attribútummal és csoporttal, így később a parancsok felismerhetik azokat.

Magát a fordítást a *CCGActionCompiler* osztály végzi statikus metódusok segítségével. Ezek a metódusok bemenetként várják az absztrakt szintaxisfának és a kontextusfának aktuális csomópontjait, valamint azt, hogy milyen mélyen tart a feldolgozás (erre csupán a formázás során lesz szükség).

3.3.4 A kártyalista fordítása

Minden kártya, ami a szett XML-ben felsorolásra kerül, külön osztályba fordul. Ezek az osztályok a *Card* absztrakt osztályból származnak. Az osztály neve a kártya nevének biztonságos formája. Az attribútumok asszociatív tömböt alkotnak, a kártya-specifikus műveletek pedig egyetlen közös *Resolve* metódusba fordulnak, amit az osztály még a *GameObject*-től örökölt.

3.3.5 CCGActionCompiler

A fordítás úgy működik, hogy az AST adott csomópontját kezelni képes *processX* metódus felépíti a saját hatáskörébe tartozó kódrészlet szerkezetét és a hézagokat kitölti

a továbbhívott *processY* metódus eredményével, ahol Y az AST fában X gyermeke. A továbbhívás kaphat saját kontextusfa csomópontot, amit fel kell venni a hívó fél csomópontjának gyerekei közé, így lehetővé tesszük a keresést és a típushelyesség betarttatását. Mindegyik *process* metódus az általa felépített kódrészlettel tér vissza.

4 Felhasználói felület

Ebben a fejezetben bemutatok néhány képernyőképet a szerkesztő alkalmazásból.

4.1 Főmenü

A program indításakor megjelenő ablak. A kezelőfelület négy lehetőséget kínál: új kártyajáték definiálása, egy már elmentett játékleírás betöltése, egy megépített játék futtatása és kilépés.

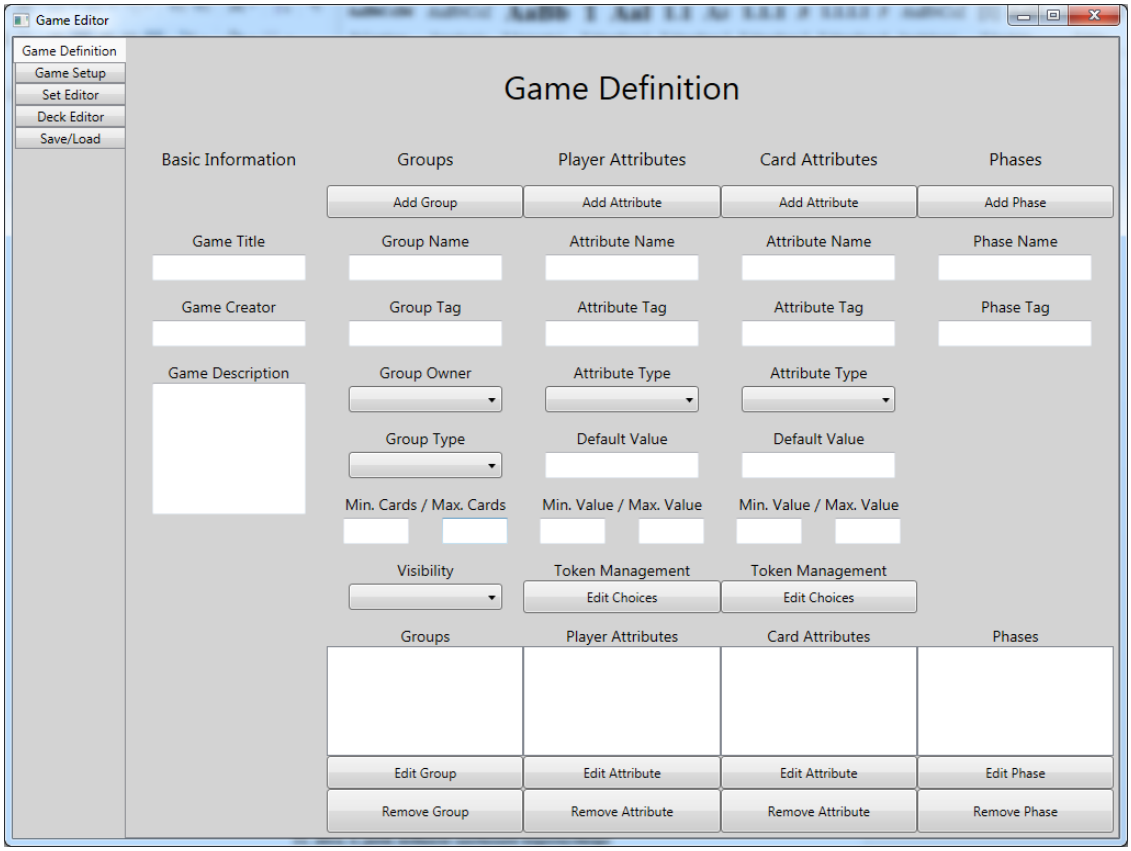
10. ábra A főmenü képe

4.2 A szerkesztő felület

Itt a szerkesztés során használható képernyőket mutatom be.

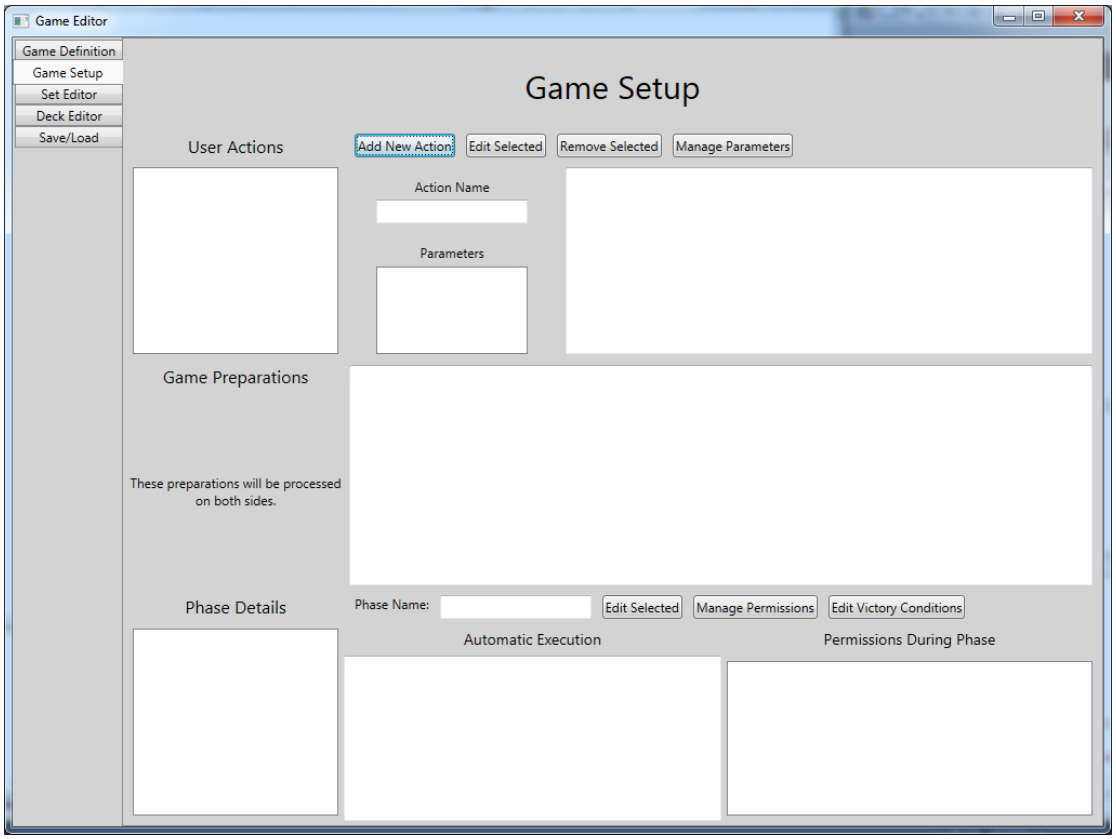
A CCGDef séma minden mezője kitölthető az alábbi képernyőn. Az első oszlopban lehet megadni a meta-adatokat. A második szolgál a játék kártyacsoportjainak felvételére és módosítására. A harmadik oszlopban a játékosok tulajdonságait lehet beállítani, a negyedik oszlopban a kártyák tulajdonságait. Az utolsó oszlop szolgál a játék

fázisainak rögzítésére. Az ezek alapján épülő modellből generálható a játékdefiníciót leíró XML állomány és vissza.



11. ábra A játék definíció szerkesztő képernyőképe

A következő oldalon definiálhatóak a játék alapszabályai. Az oldal három részre tagolódik. A felső harmadban van módja a felhasználónak definiálnia a saját utasításkészletét. A középső harmadban lévő óriási szövegdoboz szolgál a játék indítását követő előkészületek megadására (pakli összeállítás, keverés, laphúzás). Az itt leírt utasítások szimmetrikusan mennek végbe a játékosnál és ellenfelénél. Az alsó harmadban van lehetőség a fázisok szabályrendszerének kifejtésére. Nem csak azt lehet itt leírni, milyen automatikus események menjenek végbe, de azt is, hogy milyen cselekedetek engedélyezettek a játékos számára.



12. ábra A játékszabály szerkesztő képernyőképe

5 Esettanulmány

Ebben a fejezetben bemutatom egy létező gyűjtögetős kártyajáték, a Necronomicon [7] szabályrendszerének közelítését a CCG nyelvcsalád segítségével.

5.1 A játék definíció

A játék címe Necronomicon, készítette a Games of Cthulhu. Egy játszma a Cthulhu Mythos két kultistájának párharcát hivatott szemléltetni. A játszma tétje (esetleges) előmenetel a renden belül, vagy halál.

A játék során mindkét játékos 5 lappal indul. A lapok közül a soron következő játékos kiválaszt egyet és eldönti, hogy kijátssza, vagy eldobja azt. Előbbi csökkenti az épelméjűséget, utóbbi ugyanannyival növeli. Az elhasznált lap helyett a következő kör elején a játékos új lapot húz. A cél az ellenfél életpontjának 0-ra csökkentése.

A továbbiakban részletezem a játék strukturális leírását.

5.1.1 Csoportok

A játékban 3 csoport különíthető el: a pakli, a kézben tartott lapok és az idézett teremtmény (legfeljebb egy lapot tartalmaz).

```
<groups>
  <group>
    <gmode>private</gmode>
    <gtype>deck</gtype>
    <tag>DECK</tag>
    <name>Deck</name>
    <maxCards>60</maxCards>
    <visibleBy>none</visibleBy>
  </group>
  <group>
    <gmode>private</gmode>
    <gtype>hand</gtype>
    <tag>HAND</tag>
    <name>Hand</name>
    <maxCards>5</maxCards>
    <visibleBy>player</visibleBy>
  </group>
  <group>
    <gmode>private</gmode>
    <gtype>zone</gtype>
    <tag>CRT</tag>
    <name>Creature</name>
    <maxCards>1</maxCards>
    <visibleBy>both</visibleBy>
  </group>
</groups>
```

13. ábra A játékban felismerhető kártyacsoportok

5.1.2 A játékos tulajdonságai

A játékos legfontosabb tulajdonsága az életereje (Life), ugyanis ha ez 1 alá csökken, a párbajt elvesztette. További szám típusú tulajdonságok még a misztikus erő (Arcane Power), fertőzöttség (Taint), ősi védelem (Elder Defense), erőforrás az épelméjűség (Sanity). Logikai típusú tulajdonság a sebezhetetlenség (Invulnerability), ami a következő támadás kivédése után eltűnik, valamint annak jelzésére, hogy valaki kimarad a körből, a Skip tulajdonságot vettem fel. „Választható” tulajdonság az őrület (Insanity), a játék 4 őrület közül választ egyet véletlenszerűen (megalománia, tériszony, idegengyűlölet, skizofrénia), ha az épelméjűség 0 alá esik.

```
<player>
  <playerAttr>
    <atype>number</atype>
    <tag>HP</tag>
    <name>Life</name>
    <default>100</default>
    <minValue>0</minValue>
    <maxValue>100</maxValue>
  </playerAttr>
  <playerAttr>
    <atype>number</atype>
    <tag>SAN</tag>
    <name>Sanity</name>
    <default>30</default>
    <minValue>-10</minValue>
  </playerAttr>
  <playerAttr>
    <atype>number</atype>
    <tag>ED</tag>
    <name>Elder Defense</name>
    <default>0</default>
    <minValue>0</minValue>
  </playerAttr>
  <playerAttr>
    <atype>number</atype>
    <tag>TA</tag>
    <name>Taint</name>
    <default>0</default>
    <minValue>0</minValue>
  </playerAttr>
  <playerAttr>
    <atype>number</atype>
    <tag>AP</tag>
    <name>Arcane Power</name>
    <default>0</default>
    <minValue>0</minValue>
  </playerAttr>
  <playerAttr>
    <atype>token</atype>
    <tag>INS</tag>
    <name>Insanity</name>
    <values>
      <choice>Sane</choice>
      <choice>Xenophobia</choice>
      <choice>Schizophrenia</choice>
      <choice>Megalomania</choice>
      <choice>Agoraphobia</choice>
    </values>
  </playerAttr>
  <playerAttr>
    <atype>bool</atype>
    <tag>SKIP</tag>
    <name>Skip turn</name>
    <default>false</default>
  </playerAttr>
  <playerAttr>
    <atype>bool</atype>
    <tag>INV</tag>
    <name>Invulnerability</name>
    <default>false</default>
  </playerAttr>
</player>
```

14. ábra A játékos tulajdonságai

5.1.3 A kártyák tulajdonságai

A lapoknak van egy kijátszási költsége (Cost), amit az épelméjűségből kell levonni, ha a lapot kijátsszák. A játékos dönthet úgyis, hogy a körében eldobja a lapot, ekkor az épelméjűsége ezzel az értékkel nő. A játék során van lehetőség különböző

lényeket idézni, ezeknek van támadóértékük (Attack). A lények csak akkor támadnak, ha a gazdájukat megtámadták egy képességgel. Ezt a megtorlást blokkolni lehet a saját lénnyel, ekkor is a támadóértéket kell figyelembe venni.

```
<card>
  <cardAttr>
    <atype>number</atype>
    <tag>COST</tag>
    <name>Cost</name>
    <default>0</default>
    <minValue>0</minValue>
  </cardAttr>
  <cardAttr>
    <atype>number</atype>
    <tag>ATK</tag>
    <name>Attack</name>
    <default>0</default>
    <minValue>0</minValue>
  </cardAttr>
 </card>
```

15. ábra A kártyák tulajdonságai

5.1.4 Fázisok

Három fázisból áll a játék: laphúzás fázis (Draw), főfázis (Main), lezáró fázis (Wrap Up). A laphúzás fázis automatikusan végbemegy, a fő fázisban egyetlen kártyát használhat fel a játékos (kijátssza/eldobja), az utolsó fázisban pedig a kör végén automatikusan futó folyamatok zajlanak le (pl. életerő csökkentése fertőzöttséggel).

```
 <phases>
   <phase>
     <tag>DRAW</tag>
     <name>Draw</name>
   </phase>
   <phase>
     <tag>MAIN</tag>
     <name>Main</name>
   </phase>
   <phase>
     <tag>WRAP</tag>
     <name>Wrap Up</name>
   </phase>
 </phases>
</gameDef>
```

16. ábra A játék fázisai

5.2 Játékszabályok

Az alábbiakban néhány játékszabály leírását ismertetem a CCGAction nyelvtan szintaktikáját követve.

5.2.1 Győzelmi kondíciók

A játék véget ér, ha valamelyik játékosnak 0, vagy az alá csökken az élete. Ezt automatikusan lehet ellenőrizni az utolsó fázisban.

```
(ME.HP > 0 AND OPP.HP < 1)
```

5.2.2 Fertőzöttség

A lezáró fázis alatt az életpontnak csökkennie kell a fertőzöttség értékével. Fertőzésbe nem lehet belehalni, így az 1 alá nem viszi az életet (a második kettőspont választja le az else ágat).

```
?((ME.HP <= ME.TA)):ME.HP SET 1:ME.HP -= ME.TA END
```

5.2.3 Tériszony

Ha a játékosnak tériszonya van, minden másodlagos tulajdonsága a kör végén 0 lesz.

```
?((ME.INS = „Agoraphobia”)):ME.AP SET 0 ME.ED SET 0 ME.TA SET 0 END
```

5.2.4 Megalománia

Ha a játékos megalomániás, a köre végén növekedik a misztikus ereje, de a fertőzöttsége is.

```
?((ME.INS = „Megalomania”)):ME.AP += 5 ME.TA +=2 END
```

5.3 Kódgenerálás szett leírásból

Ebben a pontban bemutatok néhány kártyát, amikhez a függelékben egymás alatt feltüntetem az eredeti kártya képét, a kártya leírását a szett XML fájlban, végül az az alapján generált C# nyelvű osztályt.

Essence of the Soul: 3 józanságért cserébe 9 életpontot gyógyít. Ezt a kártyát a képessége egyszerűsége miatt választottam elsőnek.

Dark Young Charge: 4 költségű lap, sebzi az ellenfelet 14-gyel, plusz a használó játékos aktuális misztikus erejével. Az ellenfél ősi védelme csökkenti az elszenvedett

sebzést. Azért választottam ezt második példának, mert ebben már vannak egymásba ágyazott kifejezések is.

Mad Experiment: 3 költségű lap, ami 4 lehetséges kimenetelt okozhat, a választás véletlenszerű. Egyrészt idézhet egy „Shoggoth”-ot, ami idézéskor megeszi az ellenfél lényét. Másrészt sebezhet az előző kártyáéhoz hasonló módon. Harmadik lehetőség a másodlagos tulajdonságok növelése. Végül az utolsó kimenetel az, hogy a használója megőrül. Azért választottam ezt a lapot, mert itt már megfigyelhető egy vezérlési szerkezet is.

6 Összefoglalás

Ebben a fejezetben összefoglalom az eddigi munkát, az elért eredményeket. Ezt követően felvázolom a jövőbeli terveimet a témával kapcsolatosan és felvetek néhány lehetséges továbbfejlesztési irányt.

6.1 Eddigi eredmények

A téma kidolgozása első lépéseként feltérképeztem egy általam választott szakterület (gyűjtögetős kártyajátékok) fogalomrendszerét abból a célból, hogy szöveges szakterületi nyelvet tudjak hozzá kialakítani.

A szakterületi nyelv segítségével olyan keretrendszert hoztam létre, amivel egy új kártyajáték definiálása (még ha csak egy ötlet működőképességének megállapítása is a célja) legalább egy nagyságrenddel gyorsabb, mint tisztán általános célú eszközök segítségével (néhány óra a néhány nappal szemben).

A szakterületi nyelvet egy olyan nyelvcsaládként alakítottam ki, amelyben a különböző nyelvek önmagukban csak egy absztrakciós szint meghatározásáért felelnek, de az együttműködésükkel a teljes szakterületet képesek leírni.

Természetesnek tűnt a struktúrát és a viselkedést különválasztani, az előbbit végül séma definíciókkal tudtam formába önteni, utóbbihoz pedig saját szkriptnyelvtant alakítottam ki, amit több helyen is felhasználok.

A nyelvek felismerésén túl foglalkoztam a szkriptnyelvű mondatok fordításával is, a kontextusfával támogatott fordítóm a magas szintű CCGAction nyelvről fordít C# nyelvre.

6.2 Továbbfejlesztési lehetőségek

Természetesen a fejlesztés még nem ért véget, még el kell érnem, hogy az utasítások futásidőben tudjanak fordulni és így befolyásolni tudják a működő játék vezérlését. Ezen kívül a szöveges szakterületi nyelv felé vizuális nyelvet kívánok húzni, mert azzal olyan felhasználók is tervezhetnének játékot, akiknek nincs/alig van programozási előismerete.

Egy lehetséges továbbfejlesztési irány a mesterséges intelligencia irányú bővítés, ahol az egyes játékokhoz különböző profilok rendelhetők, ahol mindegyik profil más-más hasznosságúnak tekintheti egy adott kártya használatát, ezáltal játékstílusokat lehetne kipróbálni, tesztelni.

Ambíciózus továbbfejlesztési lehetőség egy terjesztési platform fejlesztése, amin keresztül a felhasználók megoszthatják és játszhatják egymás között az újonnan definiált játékokat.

Irodalomjegyzék

[1] Collectible card game – Wikipedia, the free encyclopaedia - http://en.wikipedia.org/wiki/Collectible_card_game

[2] Scripting reference >> Legend of Grimrock - http://www.grimrock.net/modding_log1/scripting-reference/

[3] Terence Parr, Kathleen S. Fischer - LL(*): The Foundation of the ANTLR Parser Generator DRAFT - http://www.antlr.org/papers/LL-star-PLDI11.pdf

[4] Trading card – Wikipedia, the free encyclopaedia - http://en.wikipedia.org/wiki/Trading_card

[5] Five minute introduction to ANTLR3 - https://theantlrguy.atlassian.net/wiki/display/ANTLR3/Five+minute+introduction+to+ANTLR+3

[6] LINQ to XML Overwiew - http://msdn.microsoft.com/en-us/library/bb387061.aspx

[7] Play The Necronomicon, a free online gane on Kongregate - http://www.kongregate.com/games/GamesofCthulhu/the-necronomicon

Függelék

17. ábra A CCGDef séma definíció struktúrája

Az esettanulmányban említett lapok háromféle megadása


```
<CARD>
  <NAME>Essence of the Soul</NAME>
  <COST>3</COST>
  <ATK>0</ATK>
  <PLAY>ME.SAN-=3 ME.HP+=9</PLAY>
  <DISCARD>ME.SAN+=3</DISCARD>
</CARD>
```

```
public class EssenceOfTheSoulCard : Card
{
public EssenceOfTheSoulCard(){ }
public override void Resolve(string actionName){
switch (actionName)
{
case "PLAY":
{
            CCGEngine.Lookup("ME.SAN") -= 3;
            CCGEngine.Lookup("ME.HP") += 9;
}

break;
}
case "DISCARD":
{
            CCGEngine.Lookup("ME.SAN") += 3;
}

break;
}
}
}
```


```
<CARD>
  <NAME>Dark Young Charge</NAME>
  <COST>4</COST>
  <ATK>0</ATK>
  <PLAY>ME.SAN -= 4 OPP.HP -= ((ME.AP + 14)-OPP.ED)</PLAY>
  <DISCARD>ME.SAN += 4</DISCARD>
</CARD>
```

```
public class DarkYoungChargeCard : Card
{
public DarkYoungChargeCard(){ }
public override void Resolve(string actionName){
switch (actionName)
{
case "PLAY":
{
            CCGEngine.Lookup("ME.SAN") -= 4;
            CCGEngine.Lookup("OPP.HP") -= ((CCGEngine.Lookup("ME.AP") + 14)
- CCGEngine.Lookup("OPP.ED"));
}

break;
}
case "DISCARD":
{
            CCGEngine.Lookup("ME.SAN") += 4;
}

break;
}
}
}
```


```
<CARD>
  <NAME>Mad Experiment</NAME>
  <COST>3</COST>
  <ATK>0</ATK>
  <PLAY>ME.SAN -= 3 RANDOM 4 : (1) ME.CRT.NAME SET "Shoggoth" ME.CRT.ATK SET 6 OPP.CRT.NAME SET "" OPP.CRT.ATK SET 0 :
                               (2) OPP.HP -= ((ME.AP + 15)-OPP.ED) :
                               (3) ME.AP += 5 ME.ED += 5 :
                               (4) ME.SAN SET -10 :
                    ENDRANDOM</PLAY>
  <DISCARD>ME.SAN += 3 </DISCARD>
</CARD>
```

```
public class MadExperimentCard : Card
{
public MadExperimentCard(){ }
public override void Resolve(string actionName){
switch (actionName)
{
case "PLAY":
{
      CCGEngine.Lookup("ME.SAN") -= 3;

      var random_d5d0350e_3576_451a_b418_6d88a6fee670 = Random.Next(1,5);
      switch (random_d5d0350e_3576_451a_b418_6d88a6fee670)
      {
          case (1):
          {
                CCGEngine.Lookup("ME.CRT.NAME") = "Shoggoth";
                CCGEngine.Lookup("ME.CRT.ATK") = 6;
                CCGEngine.Lookup("OPP.CRT.NAME") = "";
                CCGEngine.Lookup("OPP.CRT.ATK") = 0;
               break;
          }
          case (2):
          {
                CCGEngine.Lookup("OPP.HP") -= ((CCGEngine.Lookup("ME.AP") +
15) - CCGEngine.Lookup("OPP.ED"));
               break;
          }
          case (3):
          {
                CCGEngine.Lookup("ME.AP") += 5;
                CCGEngine.Lookup("ME.ED") += 5;
               break;
          }
```

```
          case (4):
          {
                 CCGEngine.Lookup("ME.SAN") = -10;
                break;
          }
      }
      }
      break;
}
case "DISCARD":
{
     CCGEngine.Lookup("ME.SAN") += 3;
}
break;
}
}
}
```