

Önálló laboratórium

Keretrendszer szövegalapú kalandjátékhoz Prolog nyelven

Készítette: Szabó Gergő

Konzulens: Szeredi Péter

Tartalomjegyzék

1.	Bevezetés.....	3
2.	Szövegalapú kalandjátékok és a Prolog	4
3.	Szoftvermodulok és kapcsolataik.....	5
4.	A <i>game_params</i> modul	6
5.	A <i>game_language</i> modul.....	9
6.	A <i>game_motor</i> modul.....	9
7.	A <i>break_in</i> és <i>break_in_text</i> modulok.....	10
8.	Rövid esettanulmány a játékról	12

1. Bevezetés

Ebben a fejezetben szeretnék egy rövid, átfogó összeggést adni a félév munkájáról, illetve a beszámoló felépítéséről.

Számomra ez a feladat nem csak egy egyszerű tanulási kihívásként jelent meg, hanem egy új világba való elmerülésként is, hiszen eddig nem volt alkalmam dolgozni deklaratív nyelvekkel egy összetettebb projekt keretein belül. A Prolog ("Programmation en Logique"), mint deklaratív programozási nyelv, az előző ismereteimet felülírva új megközelítéseket és gondolkodásmódot kívánt tőlem. Eddig inkább imperatív nyelvekkel dolgoztam, így a Prologgal egészen más perspektívából kellett megközelítenem a problémamegoldást. Ennek megfelelően a félév eleje a nyelv sajátosságainak, működési elveinek a megtanulásával, gyakorlásával telt a konzulensem segítségével.

Ezt követően, hozzáláthattam a projektmunkámhoz, ami nem más, mint egy szövegalapú kalandjáték implementálása Prolog nyelven. A szövegalapú kalandjátékok lényege, hogy a játékosok a történet és a környezet felfedezésére összpontosítanak, szöveges leírások és parancsok segítségével haladnak előre. Egyik fontos jellemzőjük, ami miatt szimpatikus volt nekem ez a játékműfaj erre a feladatra, hogy gyakran egyáltalán nem tartalmaznak grafikát, a hangsúly a szövegen és a játékos kreativitásán van. A grafikai elemek minimalizálásával nem csupán az egyszerűsége törekedtem, hanem kicsit tehermentesíteni szerettem volna magam, hogy elsősorban a Prolog programozási nyelvvel való ismerkedésre tudjak fókuszálni. Mivel a Prolog számomra eddig nagyrészt ismeretlen terület volt, kitűzött célom volt, hogy a játékefejlesztés során inkább ezt a próbatételt állítsam magam elé.

A játék elkészítését követően, a projekten végeztem egy további módosítást, szétválasztottam a játékspecifikus kódrészeket az általános kódrészekről, így, a konkrét játékon túl, született egy keretrendszer, amiben megalkotható (bizonyos megkötésekkel) tetszőleges szövegalapú kalandjáték.

A beszámolómban először összegzem, miért tartom a Prolog-ot kiváló eszköznek szövegalapú kalandjátékok megalkotására, majd ismertetem a projekt felépítését, a keretrendszer használatát, végül pedig egy rövid esettanulmányon bemutatom működés közben.

Összességében elmondhatom, hogy rengeteget tanultam a feladatból, és kiemelkedő tapasztalatokra tettem szert a deklaratív szoftverfejlesztés terén. A Prolog visszalépési mechanizmusaival való foglalkozás és a logikai következtetés alkalmazása során számos nehézséget sikerült megoldanom, és egy új szögből láthattam rá a programozás kihívásaira.


2. Szövegalapú kalandjátékok és a Prolog

A szövegalapú kalandjátékok kettő fontos fázisból állnak:

1. A játék felvázolja a játékosnak szöveges leírással, annak helyzetét, állapotát.
2. A játékos kiad egy parancsot, ami alapján a játék új állapotba léphet.

A kiadott parancs után a játék ismét felvázolja (ezúttal a már megváltozott) állapotot, és vár a következő parancsra.

```
You're in the following area: car
You can see the following items:
  lockpick
  battery
  fried_chicken
From here you can go to:
  gate
Obstacles that stand in your way: -
> take lockpick
The lockpick is now in your possession!
>
```

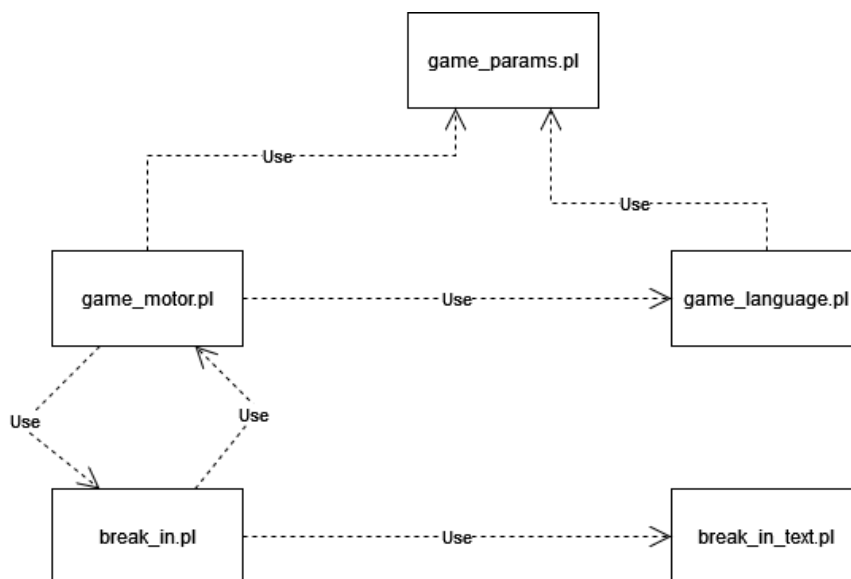


1. ábra: Példa a készített játékból

A kiadható parancsok, és lehetséges argumentumaik függenek az adott állapottól, amiben a játékos éppen van. Például a mellékelt ábrán nem tudnánk felvenni a lockpick-et, ha nem lenne velünk egy térben. Ez azt jelenti, hogy amikor a felhasználó kiad egy parancsot argumentumokkal vagy anélkül, a programnak be kell járnia egy keresési teret, amiben megállapítja, hogy az a parancs, azokkal az argumentumokkal, abban az állapotban elvégezhető-e. A Prolog nyelv egyik főprofilja éppen keresési terek bejárása, és bennük alkalmas megoldások keresése, így tökéletes eszköz erre a feladatra.

Ezentúl a Prolog a mintaillesztés és backtracking mechanizmusainak köszönhetően erős eszköz természetes nyelvek feldolgozására. Ezt az erősséget kamatoztathatjuk a szövegalapú kalandjátékoknál is. Jobb lenne például, ha a mellékelt ábrán látható „take lockpick” kötött szintaxisú parancs helyett meg lehetne fogalmazni természetesebben is az utasítást, pl. „pick up the lockpick”. A Prolog beépített nyelvtan-mechanizmusa, a Definite Clause Grammar (DCG), alkalmas arra, hogy definiáljuk a játékunk által használt nyelvtant, amivel aztán akár természetes szöveget is feldolgozhatunk és értelmezhetünk. Ennek következtében a játékosok megadhatják számukra természetes nyelven az utasításokat és nem kell parancsokat magolniuk.

3. Szoftvermodulok és kapcsolataik



2. ábra: A szoftver moduljai és függőségeik

Az elkészített projekt öt modulból áll, minden modulhoz külön .pl kiterjesztésű fájl tartozik. Az egyes modulokról részletesen is lesz szó, de itt röviden bemutatom őket.

Keretrendszer moduljai:

- *game_params*:
A keretrendszer config fájlja. Itt kell megadnia a keretrendszer felhasználójának a játéka jellemzőit (parancsok, argumentumok, nyelvi elemek, stb.).
- *game_language*:
Ez a modul végzi a játékos által megadott utasítások nyelvi feldolgozását a *game_params*-ban definiált nyelvi elemek alapján.
- *game_motor*:
A játéklogika általános része található itt, amely nem tartozik konkrét játékimplementációhoz.

Konkrét játék moduljai:

- *break_in*:
A modul tartalmazza az általam írt konkrét játék logikáját. A játékban egy betörőt alakítunk, akinek a célja, hogy ellopon egy értékes ékszert, innen a név.
- *break_in_text*:
A konkrét játék szöveges üzeneteit tartalmazó modul.

4. A *game_params* modul

Ha valaki a keretrendszeremet szeretné használni szövegalapú kalandjáték készítéséhez, ebben a modulban kell megadnia a játéka jellemzőit. Ezeket két részre oszthatjuk, az egyik a játéklogikához, a másik pedig a játék nyelvéhez tartozik.

A játéklogikánál a következő dolgokat kell megadnunk:

- **Parancsok:** Milyen parancsok vannak a játékban, amiket a játékos kiadhat. Ezeket a *commands* predikátumban kell definiálni.

```
commands([look,
            inventory,
            inspect,
            goto,
            take,
            combine,
            use,
            quit
        ]).
```

3. ábra: Példa a *break_in* játék esetében a *commands* predikátumra

- **Argumentumok:** Milyen argumentumtípusok vannak a játékban, és melyik parancshoz milyen argumentumtípusok tartozhatnak. Ezeket az *argument_types* és a *command_argument_pairs* predikátumokban kell definiálni.

```
argument_types([item, location, obstacle]).
```

```
command_argument_pairs([look/none, inventory/none, quit/none,
                        inspect/item, inspect/obstacle, goto/location, take/item,
                        combine/item-item, use/item-obstacle]).
```

4. ábra: Példa a *break_in* játék esetében az *argument_types* és a *command_argument_pairs* predikátumokra

Megjegyzés: A *none* argumentumtípus azt jelzi, hogy a parancshoz nem tartozik argumentum. Az *item-item* argumentumtípus azt jelenti, hogy az adott parancshoz kettő argumentum is tartozik, mindkettő *item* típusú. A keretrendszer csak 0, 1 és 2 argumentumú parancsokat támogat.

- **Állapot:** A játék során folyamatosan nyilván kell tartani a játékállapotot, hiszen az határozza meg éppen mit tehet a játékos és mit nem. A játék közben ez az állapot változik, de a kezdő állapotot itt kell megadni. Az állapot struktúrája kötött: egy prolog lista, amelynek elemei prolog struktúrák, amelyek vagy egyetlen elemet, vagy pedig egy prolog listát tartalmaznak. Ezt a strukturális megkötést megtartva bármilyen tartalommal megtölthetjük az állapotot a készítendő játékunktól függően. A kezdő állapotot az *init_state* predikátumban kell definiálni.

```

init_state([
  here(car),
  have([]),
  items([
    car/lockpick,
    car/battery,
    car/fried_chicken,
    flowergarden/nightshade_flower,
    garage/wirecutter
  ]),
  obstacles([
    gate/gate-lockpick/courtyard,
    courtyard/dog-poisonous_chicken/garage,
    courtyard/main_door-golden_key/entrance_hall,
    flowergarden/fence-wirecutter/backyard
  ]),
  open_doors([
    car/gate,
    courtyard/flowergarden
  ]),
  craft_recipes([
    fried_chicken+nightshade_flower>>poisonous_chicken
  ])
]).

```

5. ábra: Példa a *break_in* játék esetében az *init_state* predikátumra

Néhány konkrétum a példából: A *have* struktúra tartalmazza milyen tárgyak vannak éppen a játékosnál (kezdetben üres), *items* tartalmazza melyik helyen, melyik tárgy található, *obstacles* tartalmazza melyik helyszínek között áll átjárási akadály, és melyik tárggyal lehet áthidalni az akadályt, stb.

A játék belső logikáján túl azt is definiálnunk kell a keretrendszernek, hogy milyen szavak tartoznak a játék nyelvébe. A parancsokhoz igék, az argumentumokhoz főnevek tartoznak.

- **Igék:** Milyen igék milyen parancsra forduljanak le. Ezt a *verbs_for_commands* predikátumban kell megadnunk.

```

verbs_for_commands([look/[look, l, [look, around]],
  inventory/[inventory, i],
  quit/[quit, exit, end, bye, q],
  inspect/[inspect, [check, out], investigate, review],
  goto/[go, g],
  take/[take, [pick, up]],
  combine/[combine, [put, together], merge],
  use/[use, utilize, u]
]).

```

6. ábra: Példa a *break_in* játék esetében az *verbs_for_commands* predikátumra

Megjegyzés: Ha egy parancshoz több szóból álló ige is tartozik (pl. check out), azt egy belső listába kell helyezni.

- **Főnevek:** Milyen argumentumtípusokhoz milyen főnevek tartozhatnak a játék nyelvében. Ezt a *nouns_for_argument_types* predikátumban kell definiálni.

```
nouns_for_argument_types([item/[lockpick, battery, [fried, chicken],
                             [nightshade, flower], wirecutter],
                           location/[car, gate, courtyard, flowergarden,
                                      [entrance, hall], garage, backyard],
                           obstacle/[gate, dog, [main, door], fence]
                           ]).
```

7. ábra: Példa a *break_in* játék esetében az *nouns_for_argument_types* predikátumra

Megjegyzés: A több szóból álló főneveket (pl. main door) egy belső listába kell elhelyezni.

- **Prefixek:** Természetes megfogalmazásban gyakran kerülhetnek olyan „töltelékszavak” főnevek elé, amelyek a játéklogika számára nem bírnak információval, azonban, ha azt szeretnénk, hogy lehessen természetes szöveggel utasítani a játékot, akkor gondolnunk kell rájuk. Vegyük például a „combine” szót. Ez a szó a *break_in* játékomban egy két argumentumos parancs, amely két *item* argumentumot vár. Természetes nyelvben mondhatom azt is, hogy „combine item1 and item2” vagy azt is, hogy „combine item1 with item2”. Ebben a példában most az „and” és a „with” voltak ezek a töltelékszavak, ezeket neveztem prefixeknek. A *prefixes_for_argument_types* predikátumban kell megadnunk milyen argumentumtípushoz milyen prefixek tartozhatnak. (A névelőkkel nem kell itt foglalkozni, azokat a nyelvtan intézi (5).)

```
prefixes_for_argument_types([item/[],
                             location/[to],
                             obstacle/[],
                             item-item/[]-[with, and],
                             item-obstacle/[]-[with, on]
                             ]).
```

8. ábra: Példa a *break_in* játék esetében a *prefixes_for_argument_types* predikátumra

5. A *game_language* modul

A *game_language* modul végzi a nyelvfeldolgozást egy DCG-vel megvalósított nyelvtan alapján. A nyelvtan felépítése a következő:

command → verb

command → verb compound_nounphrase

compound_nounphrase → nounphrase

compound_nounphrase → prefix nounphrase

compound_nounphrase → nounphrase nounphrase

compound_nounphrase → prefix nounphrase nounphrase

compound_nounphrase → nounphrase prefix nounphrase

compound_nounphrase → prefix nounphrase prefix nounphrase

nounphrase → noun

nounphrase → det noun

det → „a”

det → „the”

A verb, prefix, noun szabályok nincsenek feltüntetve, mivel azok jobb oldalán a *game_params* modulban definiált szavak fognak állni az adott játéktól függően.

A modul *command* predikátuma kap egy szólistát, amit megpróbál feldolgozni a nyelvtannak megfelelően. Ha sikerül, *Command* és *ArgL* változókba bekerül a parancs és annak argumentumai, ha pedig nem, akkor *fail* történik.

6. A *game_motor* modul

A *game_motor* modul felelős magának a játéknak a folyamatos futásáért, ő kezeli a játékciklust:

1. Bekéri a felhasználótól az utasítást.
2. A kapott szöveget szólistára bontja, majd átadja *game_language*-nak feldolgozásra.
3. Ha sikerült feldolgozni, akkor a kapott parancs nevének megfelelő predikátumot, a kapott argumentumokkal, meghívja a konkrét játéklógika moduljában (ez az én esetemben a *break_in* modul). Ha nem sikerül, akkor jelzi a felhasználónak, hogy nem érti az utasítást, és visszalép 1.-re.
4. Visszalép 1.-re.

Ezt a ciklust addig folytatja, ameddig a játék vége feltétel nem teljesül. Ez a teljesülhet úgy, hogy a játékos kiadja a quit parancsot, vagy a konkrét játéklógika is kérheti a játék végét.

A játékciklus kezelésén túl a *game_motor* modul még biztosít predikátumokat a játékállapot kényelmes kezeléséhez, ezeket a konkrét játéklógika moduljában fel lehet használni az állapot változtatásához. A predikátumok a következők:

- *get_state*
Az állapot alállapotainak lekérdezéséhez használatos.
- *modify_substate_swap*
Az állapot egy alállapotának a lecserélésére használatos.
- *modify_substate_add*
Az állapot egy alállapotához új érték hozzáadására használatos.
- *modify_substate_del*
Az állapot egy alállapotából érték törlésére használatos.

7. A *break_in* és *break_in_text* modulok

A *break_in* modulban írtam meg a konkrét játékom logikáját. Ha valaki játékot készít a keretrendszerben, akkor ehelyett meg kell írnia a saját modulját, annyi megkötéssel, hogy a *game_params* modulban definiált parancsok nevével azonos nevű predikátumoknak szerepelnie kell a kódban, mert a *game_motor* azokat fogja meghívni, tehát azok lesznek a modul belépési pontjai.

A *break_in_text* modulban levő predikátumok csak szöveg kiírást végeznek. Ezt fontosnak tartottam külön választani a logikától, mivel pl., ha valaki le akarja fordítani a játék nyelvét mondjuk angolról magyarra, akkor nem kell hozzányúlnia a logikához, elég ezt a modult lecserélni egy másikra.

Az általam írt játékról röviden:

A játékos egy betörőt alakít, akinek el kell lopnia egy értékes ékszert. Ehhez szobáról szobára kell mozognia, amíg el nem éri a szobát az ékszerrel. A szobákban különböző tárgyak is lehetnek, amiket a játékos felvehet és magával vihet. Egyes szobák bejáratai blokkolva vannak valamilyen akadállyal. Ahhoz, hogy az akadályokon át lehessen jutni, a talált tárgyakat kell felhasználni. Bizonyos tárgyakat összekombinálva előállítható új tárgy is, ami szintén hasznos lehet az akadályok kiküszöböléséhez. Az akadályok és tárgyak tüzetesebben is megvizsgálhatóbbak, ekkor a játék ad néhány információt a vizsgált dologról, hogy mihez érdemes kezdeni vele.

A játék argumentumtípusai: *item*, *location*, *obstacle*

A játékban használható parancsok az argumentumaikkal:

- **look**
A játékos karaktere körbenéz. A játék kiírja milyen tárgyak vannak a karakter aktuális helyszínén és milyen más helyszínekre vezet onnan út.
- **inventory**
Kilistázza milyen tárgyak találhatóak a játékosnál.
- **inspect** *item* | *obstacle*
item és *obstacle* argumentummal is hívható. Kiírja az adott tárgy/akadály jellemzőit, abban az esetben, ha a karakterrel azonos helyszínen van a tárgy/akadály.
- **goto** *location*
A karakter átmegy az argumentumban megnevezett helyre, abban az esetben, amennyiben az aktuális helyről van oda út és nem állja az útját egy akadály.

- **take** *item*
A karakter felveszi az argumentumban megnevezett tárgyat, és az inventory-ába teszi, abban az esetben, amennyiben a tárgy a karakterrel azonos helyszínen tartózkodik.
- **combine** *item item*
A karakter kombinál két tárgyat, ezzel azokat elvesztve és kapva egy új tárgyat, abban az esetben, amennyiben mindkét felhasználandó tárgy nála van és az egy valid kombináció.
- **use** *item obstacle*
A karakter használ egy tárgyat egy akadályon semlegesítve ezzel az akadályt és elvesztve a tárgyat, abban az esetben, amennyiben a tárgy nála van, az akadállyal egy helyszínen tartózkodik, és az adott tárgy feloldja az adott akadályt.
- **quit**
A játék kilép.

Az, hogy mely akadályok, mely átjárókat blokkolják, mi a feloldásuk, milyen tárgyak vannak a játékosnál, milyen nyílt átjárók vannak, mik a valid tárgykombinációk, hol tartózkodik aktuálisan a játékos, mind a játékállapotban kerülnek eltárolásra, amelynek kezdeti stádiuma a *game_params* modulban található az *init_state*-ben. A játék során a játékos különböző parancsai hatására változik az állapot az implementált logikának megfelelően.

Az állapot változtatására egyik megoldás az úgynevezett dinamikus predikátumok használata, ez azonban nem számít elegáns deklaratív megoldásnak, mivel az egyes predikátumok más eredményt adhatnak a dinamikus adatoktól függően. Előállhat így az az eset, hogy egy predikátumot kétszer lefuttatunk ugyanazokkal az argumentumokkal és mégis eltérő eredményt kapunk. Emiatt egy másik megközelítést választottam. A predikátumok, amelyek használják a játékállapotot, megkapják argumentumként az egészet, elvégzik rajta a transzformációikat, majd az előállt új állapotot kiteszik kimenő argumentumként. A következő predikátum, amely használja a játékállapotot, már ezt az új állapotot fogja használni aktuális állapotként és így tovább. Ezzel a módszerrel már nem áll fent a probléma, ami a dinamikus predikátumoknál kiritkaként felmerült.

8. Rövid esettanulmány a játékról

Ebben a fejezetben egy képen keresztül röviden demonstrálnám a játék működését.

A játékos a kezdő helyiségben felvesz egy lockpick-et, átmegy a szomszédos helyiségbe, megpróbál átmenni ott egy zárt kapun, nyilván nem tud, ezután a lockpick-et felhasználva kinyitja a kaput és átmegy rajta.

```
Welcome to Break In!
```

```
In this game you're playing as a burglar. You've heard about  
a family, who has inherited an extremely valuable necklace recently.  
This week the family's on vacation, so the house is empty. The time has come  
to make your move. You drive to the place and get out of your car.
```

```
> look around
```

```
You're in the following area: car
```

```
You can see the following items:
```

```
lockpick  
battery  
fried_chicken
```

```
From here you can go to:
```

```
gate
```

```
Obstacles that stand in your way: -
```

```
> pick up the lockpick
```

```
The lockpick is now in your possession!
```

```
> go to the gate
```

```
You're in the following area: gate
```

```
You can see the following items: -
```

```
From here you can go to:
```

```
courtyard  
car
```

```
Obstacles that stand in your way:
```

```
gate
```

```
> go to the courtyard
```

```
You can't go there, because the gate is in the way.
```

```
Inspect it, if you want to know more.
```

```
> check out the gate
```

```
The gate is locked. You'll need a key or something like that to open it.
```

```
> use the lockpick on the gate
```

```
You've successfully unlocked the passage to the courtyard!
```

```
> go to the courtyard
```

```
You're in the following area: courtyard
```

```
You can see the following items: -
```

```
From here you can go to:
```

```
flowergarden  
entrance_hall  
garage  
gate
```

```
Obstacles that stand in your way:
```

```
dog  
main_door
```

```
>
```