

Documentatie Proiect

Limbaje Formale si Translatoare

Joc Text-Based Adventure

The Cursed Vault

Simina Dan-Marius

Pantea Marius-Nicusor

29 mai 2025

Cuprins

1	Descrierea Proiectului	3
1.1	Cerintele Proiectului	3
1.1.1	Scopul Proiectului	3
1.1.2	Obiective Principale	3
1.1.3	Functionalitati Implementate	3
2	Detalii Tehnice	3
2.1	Structura Proiectului	3
2.1.1	Arhitectura Sistemului	4
2.1.2	Tokeni Identificati	4
2.1.3	Gramatica Implementata	4
2.2	Instrumente si Tehnologii Utilizate	4
2.3	Configuratia Mediului de Dezvoltare	5
2.3.1	Cerinte de Sistem	5
2.3.2	Instalarea Dependentei	5
2.3.3	Compilarea Proiectului	5
3	Detalii de Implementare	5
3.1	Implementarea Componentelor Cheie	5
3.1.1	Analizorul Lexical (Lex)	5
3.1.2	Analizorul Sintactic (Yacc)	6
3.1.3	Motorul de Joc	6
3.1.4	Validarea Semantica	7
4	Testare si Validare	7
4.1	Cazuri de Test si Rezultate	7
4.1.1	Test 1: Incarcarea Configuratiei Valide	7
4.1.2	Test 2: Validarea Erorilor de Referinta	7
4.1.3	Test 3: Gameplay Complet	8
4.1.4	Test 4: Gestionarea Erorilor de Sintaxa	8
4.2	Screenshot-uri	8
5	Evaluare	10
5.1	Puncte Forte	10
5.2	Puncte Slabe	10
5.3	Dezvoltari Ulterioare	10
6	Concluzii	11
6.1	Realizari Principale	11
6.2	Atingerea Obiectivelor	11
6.3	Reflectii Personale	11

1 Descrierea Proiectului

1.1 Cerintele Proiectului

Acest proiect reprezinta implementarea unui motor de joc text-based adventure folosind instrumentele Lex si Yacc pentru parsing-ul si interpretarea unei gramatici personalizate de configurare a jocurilor.

1.1.1 Scopul Proiectului

Dezvoltarea unui sistem complet de joc text-based care poate interpreta fisiere de configuratie structurate pentru a genera experiente interactive de tip adventure game. Proiectul demonstreaza utilizarea practica a tehnologiilor de compilare pentru crearea unui DSL (Domain Specific Language) pentru definirea jocurilor.

1.1.2 Obiective Principale

- Implementarea unui parser robust folosind Lex si Yacc
- Crearea unui motor de joc functional cu sistem de inventar, lupta si puzzle-uri
- Dezvoltarea unei gramatici pentru definirea elementelor de joc
- Validarea si gestionarea erorilor in fisierele de configuratie

1.1.3 Functionalitati Implementate

- **Sistem de camere:** Navigare intre diferite locatii cu conexiuni bidirectionale
- **Sistem de obiecte:** Gestionarea obiectelor cu proprietati complexe (takeable, requires, contents)
- **Sistem de inventar:** Capacitate limitata cu operatii de take/drop
- **Sistem de capcane:** Mecanisme de damage cu posibilitate de protectie
- **Sistem de lupta:** Inamici cu weakness-uri si rewards
- **Sistem de chei si incuietori:** Restrictionarea accesului la anumite resurse
- **Validare semantica:** Verificarea referintelor si consistentei datelor

2 Detalii Tehnice

2.1 Structura Proiectului

Proiectul este organizat in urmatoarele componente principale:

- **game_engine.c:** Motorul principal al jocului
- **yacc_source.y:** Gramatica Yacc pentru parsing
- **lex_source.l:** Lexer-ul Flex pentru tokenizare

- **config.txt**: Fisier de configuratie al jocului
- **Makefile.mk**: Script de compilare

2.1.1 Arhitectura Sistemului

config.txt → Lex Scanner → Yacc Parser → Game Data Structures → Game Engine

2.1.2 Tokeni Identificati

Lexer-ul identifica urmatoorii tokeni principali:

```

1 game          return GAME;
2 start_room    return START_ROOM;
3 health        return HEALTH;
4 room          return ROOM;
5 description   return DESCRIPTION;
6 objects       return OBJECTS;
7 connections   return CONNECTIONS;
8 takeable      return TAKEABLE;
9 requires      return REQUIRES;
10 contents     return CONTENTS;
11 hidden_item   return HIDDEN_ITEM;
12 examine      return EXAMINE;
13 trap         return TRAP;
14 enemy        return ENEMY;
15 damage       return DAMAGE;
16 weakness     return WEAKNESS;

```

Listing 1: Definitia tokenilor in Lex

2.1.3 Gramatica Implementata

Gramatica defineste urmatoarea structura ierarhica:

```

1 program: GAME STRING OPEN_BRACE configuration CLOSED_BRACE
2
3 configuration: start_room health objects rooms
4
5 object_config: IDENTIFIER COLON OPEN_BRACE object_props CLOSED_BRACE
6
7 room_definition: ROOM STRING OPEN_BRACE room_content_list CLOSED_BRACE

```

Listing 2: Fragmentul principal din gramatica Yacc

2.2 Instrumente si Tehnologii Utilizate

- **Flex (Fast Lexical Analyzer)**: Pentru analiza lexicala si tokenizare
- **Yacc/Bison**: Pentru analiza sintactica si construirea AST-ului
- **Limbajul C**: Pentru implementarea logicii de joc
- **GCC**: Compilatorul folosit
- **Make**: Pentru automatizarea procesului de build

2.3 Configuratia Mediului de Dezvoltare

2.3.1 Cerinte de Sistem

- Sistem de operare: Linux
- Compilator GCC
- Flex
- Yacc/Bison
- Make utility

2.3.2 Instalarea Dependentelor

Pe sistemele Ubuntu/Debian:

```
1 sudo apt-get install flex bison gcc make
```

2.3.3 Compilarea Proiectului

```
1 # Compilarea completa
2 make -f Makefile.mk
3
4 # Sau pas cu pas:
5 yacc -d yacc_source.y
6 flex lex_source.l
7 gcc -Wall -Wextra -g -c *.c
8 gcc *.o -o minigame -lfl
```

3 Detalii de Implementare

3.1 Implementarea Componentelor Cheie

3.1.1 Analizorul Lexical (Lex)

Lexer-ul gestioneaza tokenizarea fisierului de configuratie:

```
1 [a-zA-Z_][a-zA-Z0-9_]* {
2     yylval.str = strdup(yytext);
3     return IDENTIFIER;
4 }
5
6 \"[^\"]*\" {
7     yylval.str = strdup(yytext + 1);
8     yylval.str[strlen(yylval.str) - 1] = '\\0';
9     return STRING;
10 }
```

Listing 3: Exemplu de recunoastere token in Lex

3.1.2 Analizorul Sintactic (Yacc)

Parser-ul construiește structurile de date ale jocului:

```
1 object_config: IDENTIFIER COLON OPEN_BRACE {
2     if ($1 == NULL || strlen($1) == 0) {
3         report_error("Object name cannot be empty");
4         YYERROR;
5     }
6     if (object_name_exists($1)) {
7         char error_msg[256];
8         snprintf(error_msg, sizeof(error_msg),
9                 "Object '%s' already exists", $1);
10        report_error(error_msg);
11        YYERROR;
12    }
13    reset_temp_object();
14    temp_object.name = strdup($1);
15 } object_props CLOSED_BRACE {
16     if (!add_temp_object(temp_object)) {
17         YYERROR;
18     }
19 }
```

Listing 4: Exemplu de regula Yacc pentru obiecte

3.1.3 Motorul de Joc

Funcția principală de procesare a comenzilor:

```
1 void process_command(Player* player, char* input) {
2     char command[100];
3     char argument[100] = "";
4
5     sscanf(input, "%s %[^\\n]", command, argument);
6     to_lowercase(command);
7
8     if (strcmp(command, "go") == 0) {
9         Room* next_room = find_connected_room(
10             player->current_room, argument);
11
12         if (!next_room) {
13             printf("You can't go that way.\\n");
14             return;
15         }
16
17         if (!can_enter_room(player, next_room)) {
18             check_room_requirements(player, next_room);
19             return;
20         }
21
22         player->current_room = next_room;
23         print_room(player->current_room);
24     }
25 }
```

Listing 5: Fragmentul principal din game loop

3.1.4 Validarea Semantica

Sistemul include validari comprehensive pentru consistenta datelor:

```
1 bool validate_game_data() {
2     bool valid = true;
3
4     // Verifica daca camera de start exista
5     if (temp_start_room_name &&
6         !room_name_exists(temp_start_room_name)) {
7         char error_msg[256];
8         snprintf(error_msg, sizeof(error_msg),
9                  "Start room '%s' does not exist",
10                 temp_start_room_name);
11         report_error(error_msg);
12         valid = false;
13     }
14
15     // Valideaza referintele obiectelor
16     for (int i = 0; i < temp_object_count; i++) {
17         TempObject* obj = &temp_objects[i];
18
19         if (obj->requires_name &&
20             !object_name_exists(obj->requires_name)) {
21             // Raporteaza eroarea
22             valid = false;
23         }
24     }
25
26     return valid;
27 }
```

Listing 6: Exemplu de validare semantica

4 Testare si Validare

4.1 Cazuri de Test si Rezultate

4.1.1 Test 1: Incarcarea Configuratiei Valide

Input: Fisierul config.txt complet cu toate elementele definite corect.

Rezultat Asteptat:

Game configuration loaded successfully!

Loaded 6 objects and 5 rooms.

Rezultat Obtinut: Succes - toate obiectele si camerele au fost incarcate corect.

4.1.2 Test 2: Validarea Erorilor de Referinta

Input: Fisier de configuratie cu referinta catre un obiect inexistent:

```
1 chest: {
2     requires = nonexistent_key
3 }
```

Rezultat Asteptat:

ERROR: Object 'chest' requires non-existent object 'nonexistent_key'

Rezultat Obținut: Succes - eroarea a fost detectată și raportată corect.

4.1.3 Test 3: Gameplay Complet

Secvența de Comenzi Testată:

```
look
take shield
search loose_brick
take silver_key
go north
go west
open chest
take rusty_key
take sword
go east
go east
go north
```

Rezultat Așteptat: Jucătorul să ajungă la camera de ieșire și să câștige jocul.

Rezultat Obținut: Succes - jocul s-a desfășurat conform așteptărilor, toate mecanismele (capcane, lupta, chei) au funcționat corect.

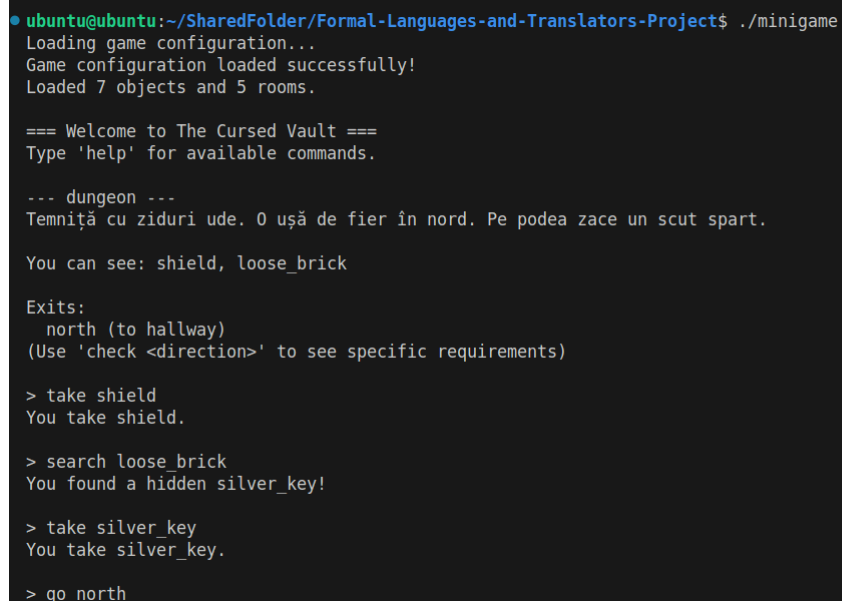
4.1.4 Test 4: Gestionarea Erorilor de Sintaxă

Input: Fișier de configurație cu sintaxă invalidă:

```
1 game "Test" {
2   invalid_keyword = "value"
3 }
```

Rezultat Obținut: Succes - parser-ul a detectat și raportat eroarea de sintaxă.

4.2 Screenshot-uri



```
• ubuntu@ubuntu:~/SharedFolder/Formal-Languages-and-Translators-Project$ ./minigame
Loading game configuration...
Game configuration loaded successfully!
Loaded 7 objects and 5 rooms.

=== Welcome to The Cursed Vault ===
Type 'help' for available commands.

--- dungeon ---
Temniță cu ziduri ude. O ușă de fier în nord. Pe podea zace un scut spart.

You can see: shield, loose_brick

Exits:
  north (to hallway)
(Use 'check <direction>' to see specific requirements)

> take shield
You take shield.

> search loose_brick
You found a hidden silver_key!

> take silver_key
You take silver_key.

> go north
```


Figura 1:

```
--- hallway ---
Coridor lung cu două statui. Aerul miroase a sulf. Vezi fante în perete pentru săgeți.

Exits:
  east (to well_room) - requires items
  west (to armory)
  south (to dungeon)
(Use 'check <direction>' to see specific requirements)
Scutul absoarbe lovitura săgeții!

> go west

--- armory ---
Sala cu arme. Pe perete: un cufăr încuiat. Ușa era blocată, dar ai reușit să intri.

You can see: chest (locked)

Exits:
  east (to hallway)
(Use 'check <direction>' to see specific requirements)

> open chest
You open the chest.
Inside the chest you find: rusty_key, sword

> take rusty_key
You take rusty_key.

> take sword
You take sword.

> go east
```

Figura 2:

```
--- hallway ---
Coridor lung cu două statui. Aerul miroase a sulf. Vezi fante în perete pentru săgeți.

Exits:
  east (to well_room) - requires items
  west (to armory)
  south (to dungeon)
(Use 'check <direction>' to see specific requirements)
Scutul absoarbe lovitura săgeții!

> go east

--- well_room ---
O fântână cu apă neagră. Corzi învechite atârnă de tavan. Un zombi păzește ușa de nord!

Ușa este încuiată! Ai nevoie de o cheie.

Exits:
  north (to exit) - requires items
  west (to hallway)
(Use 'check <direction>' to see specific requirements)
Tai rapid corzile cu spada înainte să cadă!
You encounter zombie!
You use sword against zombie!
You defeat zombie!
You found gold_key!

> go north
```

Figura 3:

```
--- exit ---
AI DESCHIS UȘA BLOCATĂ CU CELE DOUĂ CHEI! LIBER!

Ușa de ieșire are două încuietori. Ai nevoie de ambele chei!

Congratulations! You have won the game!

You completed the game successfully!
```

Figura 4:

5 Evaluare

5.1 Puncte Forte

- **Parser Robust:** Implementarea completa cu Lex si Yacc ofera o analiza precisa a fisierelor de configuratie
- **Validare Comprehensiva:** Sistemul detecteaza si raporteaza diverse tipuri de erori semantice
- **Arhitectura Modulara:** Separarea clara intre parser, validare si motorul de joc
- **Gameplay Complet:** Toate functionalitatile planificate sunt implementate si functioneaza
- **Gestionarea Memoriei:** Utilizarea corecta a functiilor de alocare/eliberare de memorie
- **Extensibilitate:** Gramatica permite adaugarea usoara de noi elemente

5.2 Puncte Slabe

- **Limitari de Capacitate:** Limitari hard-coded pentru numarul maxim de obiecte si camere
- **Lipsa de Persistenta:** Progresul jocului nu se salveaza intre sesiuni
- **Interface Minimalista:** Lipsesc elemente de formatare avansata pentru text
- **Documentatia Codului:** Unele functii ar beneficia de mai multe comentarii
- **Memory Leaks Potentiale:** In anumite scenarii de eroare, memoria alocata dinamic ar putea sa nu fie eliberata

5.3 Dezvoltari Ulterioare

- **Sistem de Save/Load:** Implementarea salvarii progresului in fisiere
- **Parser Imbunatatit:** Suport pentru comentarii pe mai multe linii si escape sequences
- **Sistem de Scoruri:** Adaugarea unui sistem de punctaj si realizari
- **Interface Grafica:** Portarea catre o interfata grafica simpla
- **Multiplayer:** Suport pentru mai multi jucatori in acelasi joc
- **Scripting Avansat:** Adaugarea de conditii si evenimente complexe
- **Audio:** Integrarea de efecte sonore si muzica de fundal
- **Localizare:** Suport pentru mai multe limbi

6 Concluzii

6.1 Realizari Principale

Proiectul a reusit sa implementeze cu succes toate obiectivele propuse:

- Crearea unui parser complet si functional folosind Lex si Yacc
- Dezvoltarea unui motor de joc text-based cu toate functionalitatile planificate
- Implementarea unui sistem robust de validare si gestionare a erorilor
- Demonstrarea utilizarii practice a tehnologiilor de compilare intr-un context real

6.2 Atingerea Obiectivelor

Obiectivele initiale au fost atinse in totalitate:

- Parser functional pentru DSL-ul de configurare jocuri
- Motor de joc complet cu toate sistemele implementate
- Validare semantica comprehensiva
- Gestionarea erorilor si raportarea precisa

6.3 Reflectii Personale

Procesul de dezvoltare a oferit o intelegere aprofundata a:

- **Tehnologiilor de Compilare:** Utilizarea practica a Lex si Yacc in proiecte reale
- **Design de Limbaje:** Principiile de creare a unei gramatici consistente si expresive
- **Arhitectura Software:** Importanta separarii responsabilitatilor in sisteme complexe
- **Validarea Datelor:** Necesitatea verificarilor comprehensive pentru robustete

Lectii Invatate:

- Planificarea atenta a gramaticii previne multe probleme ulterioare
- Validarea semantica este la fel de importanta ca cea sintactica
- Testarea incrementala accelereaza procesul de dezvoltare
- Documentatia clara faciliteaza mentenanta si extensibilitatea

Aplicarea Viitoare: Cunostintele dobandite vor fi valoroase pentru:

- Dezvoltarea de compilatoare si interpretatoare
- Crearea de DSL-uri pentru domenii specifice
- Proiectarea de sisteme de parsing pentru diverse formate de date
- Implementarea de solutii robuste cu validare comprehensiva

Proiectul demonstreaza ca tehnologiile clasice de compilare raman relevante pentru solutionarea problemelor moderne de processing al limbajelor si datelor structurate.