Project Report: Modified version of Mastermind

by Lorenzo Orsini and Gianluca Di Tuccio

Introduction

For this project, we have developed a modified version of the game Mastermind by using Prolog. This report is organized as follows. In the first paragraph, the rules of the modified Mastermind are explained, while in the second is reported a description of the Prolog code developed for this project.
The program can be run only on the stable release of SWI-Prolog, which can be found here.
All the documentation (code and project report) can be found here: https://github.com/DitucSpa/Prolog_ModifiedMastermind.

1. The Modified Mastermind Game

Mastermind is a game in which the user plays against the CPU, and it has the aim of guessing a word through a predefined number of attempts.
The CPU provides a list of words, based on the chosen difficulty, and the user selects a word at each attempt. If the chosen word matches the one to guess, the user wins the game. Otherwise, the CPU provides help to the user, which is represented by a ***Likeness***: this value indicates how many letters of the chosen word are present in the same position as the target word.

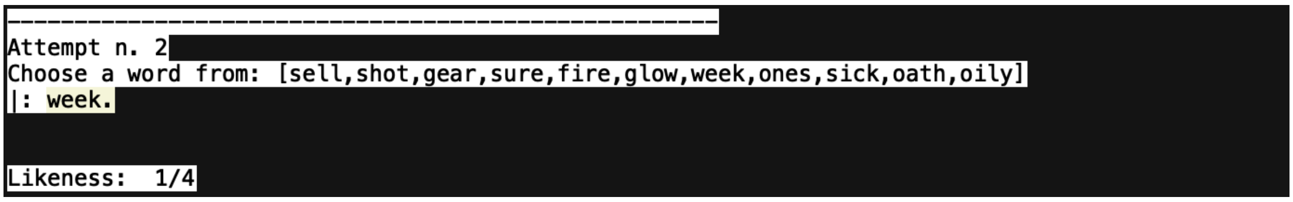
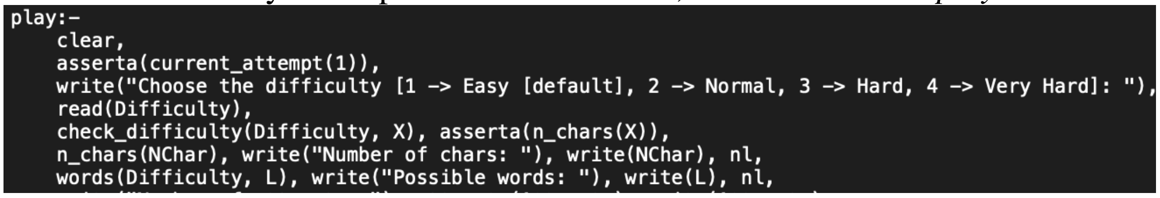

```
-------------------------------------------------------
Attempt n. 2
Choose a word from: [sell,shot,gear,sure,fire,glow,week,ones,sick,oath,oily]
|: week.


Likeness:  1/4
```

Figure 1: an example of the game. Here the user didn't guess the right word, so the Likeness 1/4 means that the word chosen (week) contains one letter that is also present in the target word.

The CPU wins when the user fails to guess the word in the predetermined number of attempts.
For this project, we have established 4 difficulties: easy, normal, hard and very hard. For each difficulty change the number of attempts, length and number of words:
- ***Easy*** modality: 4 attempts and 11 words of 4 letters each;
- ***Normal*** modality: 5 attempts and 10 words of 5 letters each;
- ***Hard*** modality: 5 attempts and 13 words of 5 letters each;
- ***Very*** hard modality: 5 attempts and 19 words of 7 letters each.

2. Prolog Implementation

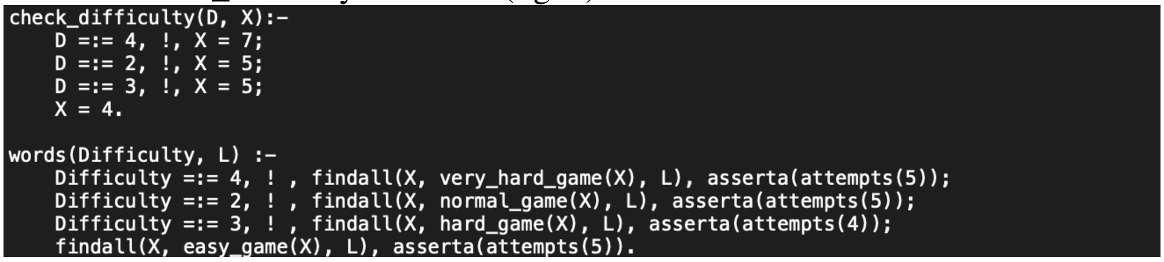
For uploading the file "*modified_mastermind.pl*" on ***SWI-PROLOG***, digit the path where you have saved the file. For example: *consult("/Users/.../modified_mastermind.pl")*. Remember that it is important to put the . char (dot) at the end of each command.
Once the program is open, the user can start the game with the command "*play.*" or clean the terminal with the command "*clear.*" Once the "play." command has been typed, the user selects the difficulty. To implement this function, we have used the *play* rule:



```
play:-
    clear,
    asserta(current_attempt(1)),
    write("Choose the difficulty [1 -> Easy [default], 2 -> Normal, 3 -> Hard, 4 -> Very Hard]: "),
    read(Difficulty),
    check_difficulty(Difficulty, X), asserta(n_chars(X)),
    n_chars(NChar), write("Number of chars: "), write(NChar), nl,
    words(Difficulty, L), write("Possible words: "), write(L), nl,
```

Figure 2: the play rule.

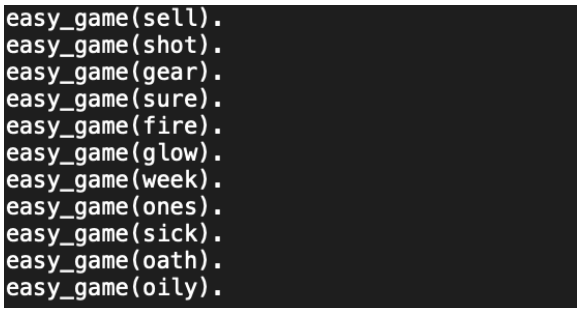
When it is called, the *play* rule first clears the window and then it asks to the user to select the difficulty number with the command *write*. Once the user types the difficulty number, the rule "check_difficulty" is called (fig. 3):



```
check_difficulty(D, X):-
    D =:= 4, !, X = 7;
    D =:= 2, !, X = 5;
    D =:= 3, !, X = 5;
    X = 4.

words(Difficulty, L) :-
    Difficulty =:= 4, ! , findall(X, very_hard_game(X), L), asserta(attempts(5));
    Difficulty =:= 2, ! , findall(X, normal_game(X), L), asserta(attempts(5));
    Difficulty =:= 3, ! , findall(X, hard_game(X), L), asserta(attempts(4));
    findall(X, easy_game(X), L), asserta(attempts(5)).
```

Figure 3: the check_difficulty rule and words rules.

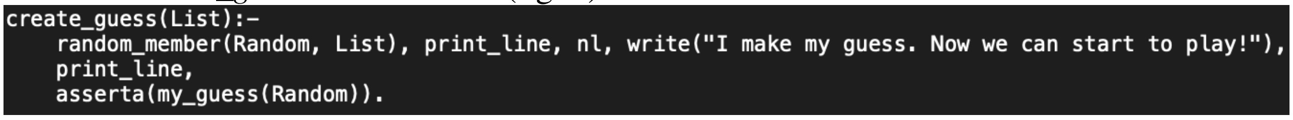
check_difficulty associates each difficulty number with the respective number of letters of each word. In particular, if the user types a different number than 2, 3, or 4, the default difficulty is set to easy (4 chars). Then the rule play ends by calling the predicate *asserta* to add to the database the number of letters and print the list of possible words in the window.
Then the rule *words* is called: here, according to the difficulty, the rule returns the list that contains the words of this difficulty (fig. 4), by using the predicate *findall*. In addition, the rule adds the number of attempts for the chosen difficulty, which will be used later to check how many attempts are left to the user.



```
easy_game(sell).
easy_game(shot).
easy_game(gear).
easy_game(sure).
easy_game(fire).
easy_game(glow).
easy_game(week).
easy_game(ones).
easy_game(sick).
easy_game(oath).
easy_game(oily).
```

Figure 4: an example of possible words
(described as facts) for an easy game.

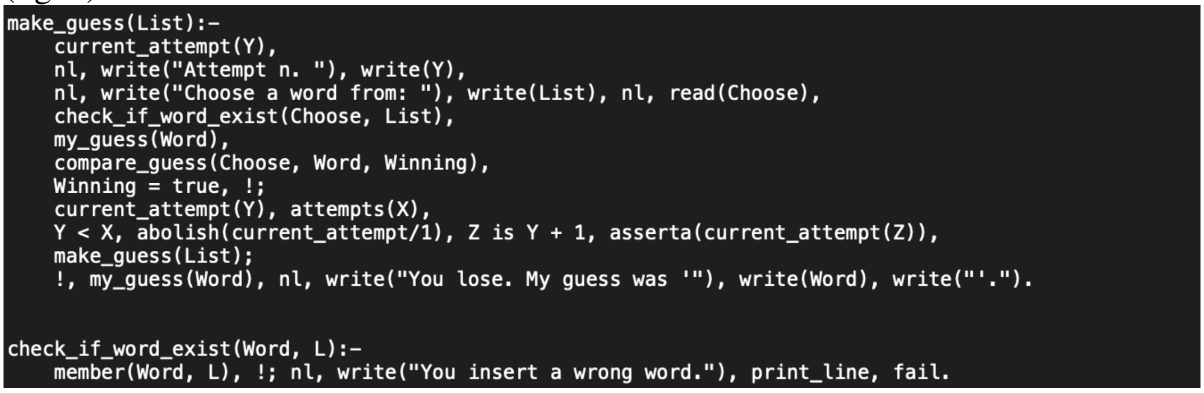
Now the *create_guess* rule is called (fig. 5):



```
create_guess(List):-
    random_member(Random, List), print_line, nl, write("I make my guess. Now we can start to play!"),
    print_line,
    asserta(my_guess(Random)).
```

Figure 5: the create_guess rule.

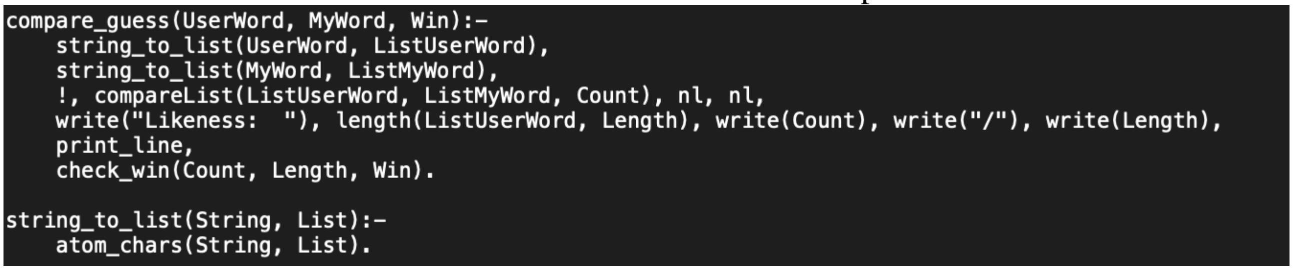
This rule randomly selects a word from the list of possible chosen difficulty, which becomes the target word for the user. Then the user makes the guess thanks to the *make_guess* rule (fig. 6):



```
make_guess(List):-
    current_attempt(Y),
    nl, write("Attempt n. "), write(Y),
    nl, write("Choose a word from: "), write(List), nl, read(Choose),
    check_if_word_exist(Choose, List),
    my_guess(Word),
    compare_guess(Choose, Word, Winning),
    Winning = true, !;
    current_attempt(Y), attempts(X),
    Y < X, abolish(current_attempt/1), Z is Y + 1, asserta(current_attempt(Z)),
    make_guess(List);
    !, my_guess(Word), nl, write("You lose. My guess was '"), write(Word), write("'.").


check_if_word_exist(Word, L):-
    member(Word, L), !; nl, write("You insert a wrong word."), print_line, fail.
```

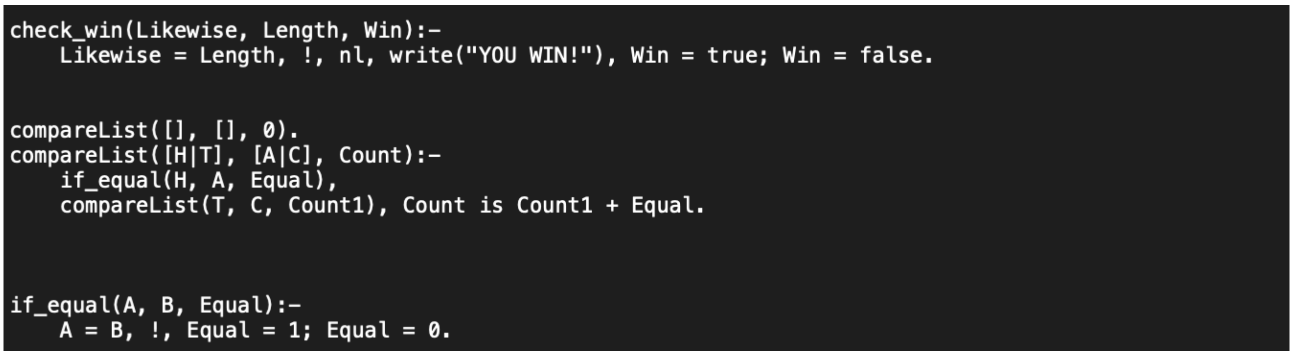
Figure 6: make_guess and check_if_word_exist rules.

This rule allows the user to write his attempt and check if it is in the list (*check_if_word_exist)* rule. Then the guess of the user is compared with the target one by calling the *compare_guess* (fig.7) rule, which converts the word from a string to a list of its letters and then it compares the two lists (guess and target word) with the rule *compareList* (fig.7). *compare_guess* print also the **Likeness** according to the counter of *compareList*, which is incremented each time the same letter is in the same position in the two words.



```
compare_guess(UserWord, MyWord, Win):-
    string_to_list(UserWord, ListUserWord),
    string_to_list(MyWord, ListMyWord),
    !, compareList(ListUserWord, ListMyWord, Count), nl, nl,
    write("Likeness:  "), length(ListUserWord, Length), write(Count), write("/"), write(Length),
    print_line,
    check_win(Count, Length, Win).

string_to_list(String, List):-
    atom_chars(String, List).
```

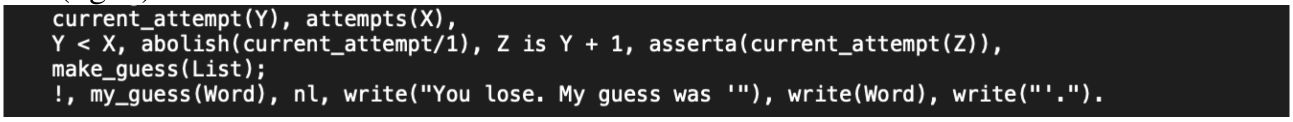
Figure 7: compare_guess and string_to_list rules.



```
check_win(Likewise, Length, Win):-
    Likewise = Length, !, nl, write("YOU WIN!"), Win = true; Win = false.


compareList([], [], 0).
compareList([H|T], [A|C], Count):-
    if_equal(H, A, Equal),
    compareList(T, C, Count1), Count is Count1 + Equal.



if_equal(A, B, Equal):-
    A = B, !, Equal = 1; Equal = 0.
```

Figure 8: check_win, compareList and if_equal rules.

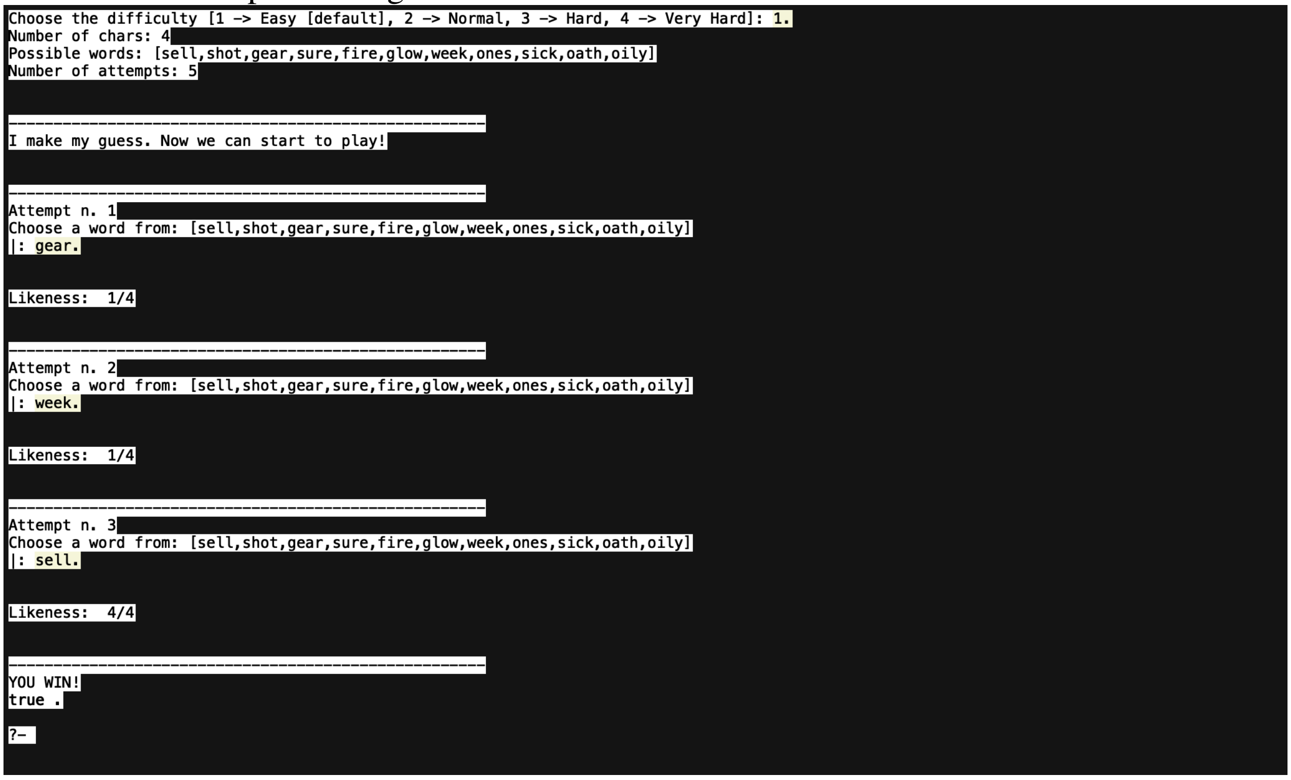
In the end, *check_win* checks if the user guess matches the target guess: this operation is done by checking if the likeness is equal to the length of the word. If this is the case, the user wins and the program terminates, otherwise make_guess checks the number of attempts left (fig. 9).



```
    current_attempt(Y), attempts(X),
    Y < X, abolish(current_attempt/1), Z is Y + 1, asserta(current_attempt(Z)),
    make_guess(List);
    !, my_guess(Word), nl, write("You lose. My guess was '"), write(Word), write("'.").
```

Figure 9: the current_attempt increment in the make_guess rule.

Let's see an example of the game:



```
Choose the difficulty [1 -> Easy [default], 2 -> Normal, 3 -> Hard, 4 -> Very Hard]: 1.
Number of chars: 4
Possible words: [sell,shot,gear,sure,fire,glow,week,ones,sick,oath,oily]
Number of attempts: 5


------------------------------------------------------
I make my guess. Now we can start to play!


------------------------------------------------------
Attempt n. 1
Choose a word from: [sell,shot,gear,sure,fire,glow,week,ones,sick,oath,oily]
|: gear.


Likeness:  1/4


------------------------------------------------------
Attempt n. 2
Choose a word from: [sell,shot,gear,sure,fire,glow,week,ones,sick,oath,oily]
|: week.


Likeness:  1/4


------------------------------------------------------
Attempt n. 3
Choose a word from: [sell,shot,gear,sure,fire,glow,week,ones,sick,oath,oily]
|: sell.


Likeness:  4/4


------------------------------------------------------
YOU WIN!
true .

?-
```