Demonstrating Prolog Usage:

Building a Simple Game Program

By Wan Hussain Wan Ishak

Abstract

To demonstrate the use of Prolog and how programming in Prolog is easy and efficient, this paper describe an implementation of the game *booby-trap*, a game similar to Minesweeper found on most personal computers with windows.

Introduction

Prolog (Programming in Logic) is a well known programming language in the field of Artificial Intelligence (AI). Although Prolog has much in common with Lisp (List Processing), only Prolog has a built in automated reasoning system or inference engine. Since its invention in 1972 by Alain Colmerauer and his colleagues at the University of Aix-Marsielle, France, there have been many versions of Prolog. These include Turbo Prolog [1] – now supported by the Prolog Development Center *www.pdc.dk*), BinProlog [2] (*www.binnetcorp.com/BinProlog*), SICStus Prolog (Intelligent Systems Laboratory, 1995 - *www.sics.se/sicstus*) and LamdaProlog [3]. There are also many other popular Prolog's such as Logic Programming Associates Prolog (LPA - *www.lpa.co.uk*) and AMZI Prolog (*www.amzi.com*). Dozens of Prolog books and tutorials mention other lesser-known Prologs.

Covington [4], one of many who tout Prologs versatility as a programming language, points out how it can implement a variety of algorithms; not just those for which it was specially designed. Prolog consists of two main components: facts and rules. The facts describe relationships between objects whereas the rules define relationships between groups of facts. Prolog is primarily based on three main concepts: pattern matching, automatic backtracking and tree-based data structuring.

In the real world, logic-programming languages such as Prolog have not fulfilled their potential when compared to other conventional programming languages [5]. This was not due as much to limitations in the language as to the ebb and flow of popular media and general computer hype. According to Somogyi, logic-programming languages have theoretical advantages over conventional programming languages, however they have not had the same impact on the computer industry. Although this claim may be true in general, in AI research many intelligent and knowledge-based systems have been successfully developed and deployed using Prolog [4]. Many of these applications would be much harder to develop in languages other than Prolog. Furthermore, Prolog is an "expressive" programming language, which contrasts with conventional "imperative" languages (see sidebar on imperative languages). Prolog contains a number of advanced features, including high-level declarative programming, automatic dynamic memory allocation and deallocation, built in database functionality, incremental compilation and meta-programming [6]. In his book, Luger [7] states that Prolog has made many contributions to AI problem solving with its declarative semantics and built-in unification.

Today, Prolog's have been enhanced with development environments and many integrated tools to simplify implementation. This paper describes the implementation of a simple game program using LPA Win-Prolog. This particular Prolog offers several tools for application development, including a tool for developing Graphical User Interface (GUI), namely the LPA Dialog Editor. This tool lets you draw screens and generate and test the Prolog code in-situ. The LPA Dialog Editor is implemented entirely in LPA Win-Prolog using the buit-in predicates described in this article.

GUI Development

The GUI, an important element in system development, must support an interactive interaction between the system and the user. LPA's Win-Prolog allows interactive dialog development utilizing a window dialog with GUI elements and control features along with a message handler to interpret the control messages.

Win-Prolog offers two class types for developing dialogs: window class and control class. A window class supplies a skeleton framework for developing dialogs. The Window control class provides the means to embed control objects into windows with several control classes such as button, edit, list box, combo box, static, scrollbar and graphics. Using the Dialog Editor plug-in (Figure 1), the user controls objects by click and drag on the scratch window. The Dialog Editor plug-in is a toolkit for easy development of GUIs and it allows easy

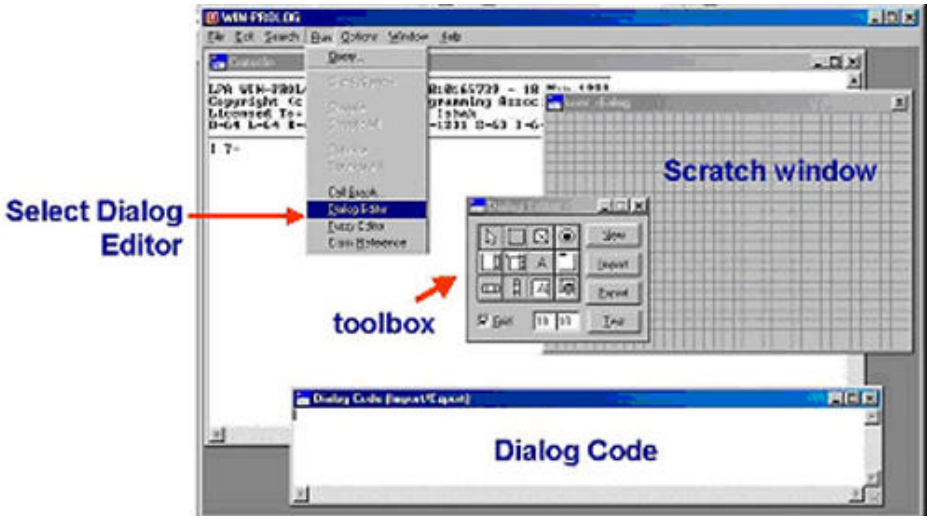

Figure 1: The Dialog Editor plug-in.

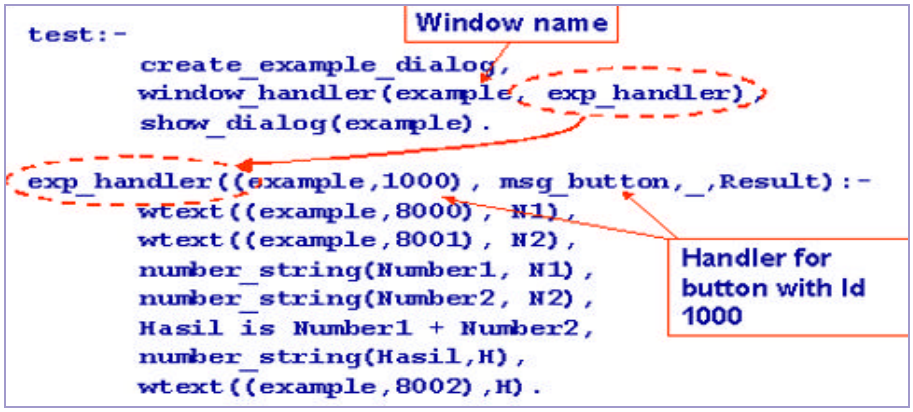

Figure 2: Creating and defining handler for the example window.

Table 1: Built-in Predicates for Window Development	
Predicate	**Description**
wdcreate/7	**Create a dialog window**
wcreate/8	**Create a window**
wccreate/8	**Create a control window**
wucreate/6	**Create a user MDI window**
wtcreate/6	**Create a text window**
wcount/4	**Get windows char, word and line counts**
wtext/2	**Get or set the window text**

creation and maintenance of the dialog code.

Clicking the export button on the toolbox lets you view the dialog code. The code appears in a dialog code window. The code must be paste into a program file.

Win-Prolog provides several built-in predicates for GUI development. The Dialog Editor automatically creates some of these predicates, such as wdcreate/7 and wccreate/8. The window predicates are shown in Table 1.

Other predicates support the control classes. These predicates are shown in Table 2.

Massage Handling

After designing the interface and placing the source code into the program file, the window handler is specified. This handler catches any message generated by the window and performs the specified action. The relation between a window and its handler can be defined using the window_handler(Window, Handler) predicate (Figure 2). For example, the window handler could be exp_handler. Hence, the relation between example window and exp_handler is defined as window_handler(example, exp_handler).

A Booby-Trap Game

The booby-trap game is similar to Minesweeper except the rules and how it plays are slightly different (Figure 3). After the player selects a button, the game returns a score. If the button is a trap then the total score is reset to zero.

The development of a program such as this game involves the following steps:

Select and drag - Create the buttons and edit control to display the score.

Defined the handler - The handler is used to link the button with the appropriate user defined action.

Table 2: Built-in Predicates for Control Classes		
Control Class	**Predicate**	**Description**
Button Class	**wbtnsel/2**	**Get or set the selection status of the given radio or checkbox "button".**
List box class	**wlbxadd/3**	**Add an item to a list box**
	wlbxdel/2	**Delete an item from a list box**
	wlbxfnd/4	**Find a string in a list box**
	wlbxget/3	**Get an item from a list box**
	wlbxsel/3	**Get or set selection in a list box**
Edit box class	**wedtsel/3**	**Get or set selection in an "edit" control window**
	wedtfnd/6	**Find a text string in an "edit" control window**
	wedtlin/4	**Get offsets a line in an "edit" control window**
	wedtpxy/4	**Convert between linear offset and x, y coordinates in "edit" windows**
	wedttxt/2	**Get or set the text of the given "edit" window**

Preparing the trap button - Some buttons are randomly selected and set as traps.

Click & draw

The developer selects the button control from the toolbox and places it on the scratch window (Figure 4). After the buttons are arranged side-by-side in rows and columns, the edit control used for displaying the score is added. Two additional buttons are added to perform the *New Game* and *Close* functions.

Handler

Each button has its own unique ID number as a reference. When the user clicks a button, the handler for the appropriate button is called, with the unique id, and the appropriate action is performed. In this game, only the *New* and *Close* buttons have a specific action and all of the others buttons shared the same action – depending on whether or not the button is a trap. Buttons listed in id/1 are the trap buttons while the others are not (List 1).

Figure 3: The Booby Trap Game Screen.

Preparing the trap button

The program randomly selects 50 buttons to be defined as a trap – assert in working memory (List 2).

Conclusion

Due to its vast capabilities, Prolog can be a difficult language to master or even to teach. However, programming with Prolog can be easy. As in many languages, programming by using existing example simplifies application development. While we illustrated how to develop a simple game program called booby-trap, we also demonstrated some of Prologs capabilities and how users can easily use this language to develop many types of applications. By providing high-level access to the Windows GUI, Prolog development environments, such as LPA, offers a Prolog system where areas such as games and intelligent interfaces are directly now addressable by the Prolog (only) developer. Next generation applications such as InFlow, IdeaProcessor and IntellX hint at some of the potential achievable using this powerful combination (see *www.orgnet.com*, *www.a-i-a.com/englishHomePage/IdeaProcessor.html* and *www.business-integrity.com/IntellX.html* for more information.)

```
% Run the dialog by typing "start"
% Create the dialog and defined the
%    handler

start:-
  % create the window
  create_booby_dialog, !,
  % defining the handler
  window_handler(booby,booby_handler),
  % show the dialog
  show_dialog(booby),
  % randomly choose the trap button
  create_trap(50).

% The handlers
% Called when user clicked the [X] on
%    the upper-right of the window
booby_handler(booby,msg_close,_,close).


% Called when user clicked close button
booby_handler((booby,1128),
    msg_button,_,close).


% Called when user clicked new button
booby_handler((booby,1129),
    msg_button,_,R):-
            !,
            start.


% Called when user clicked any button
%   and the button is a trap
booby_handler((booby,ID),
    msg_button,_,R):-
            id(ID),
            wtext((booby,ID),`F`),
            wtext((booby,8000),`0`),
            !,fail.


% Called when user clicked any button
booby_handler((booby,ID),
    msg_button,_,R):-
            wtext((booby,ID),`$`),
            wtext((booby,8000),X),
            number_string(Score,X),
            NewScore is Score + 10,
            number_string(NewScore,X2),
            wtext((booby,8000),X2),
            !,fail.
```

Listing 1: Booby-Trap Handler

```
% Create the trap button
create_trap(0).       % stopping criteria

% Randomly select the trap button
create_trap(Num_trap):-
    % 128 are the total number of buttons
            X is rand(128),
            Btt_ID is ip(X),
            not(id(Btt_ID)),!,
            assert(id(Btt_ID)),
            Red_Num_trap is Num_trap - 1,
            create_trap(Red_Num_trap).
```

Listing 2: Randomly creating the trap button.

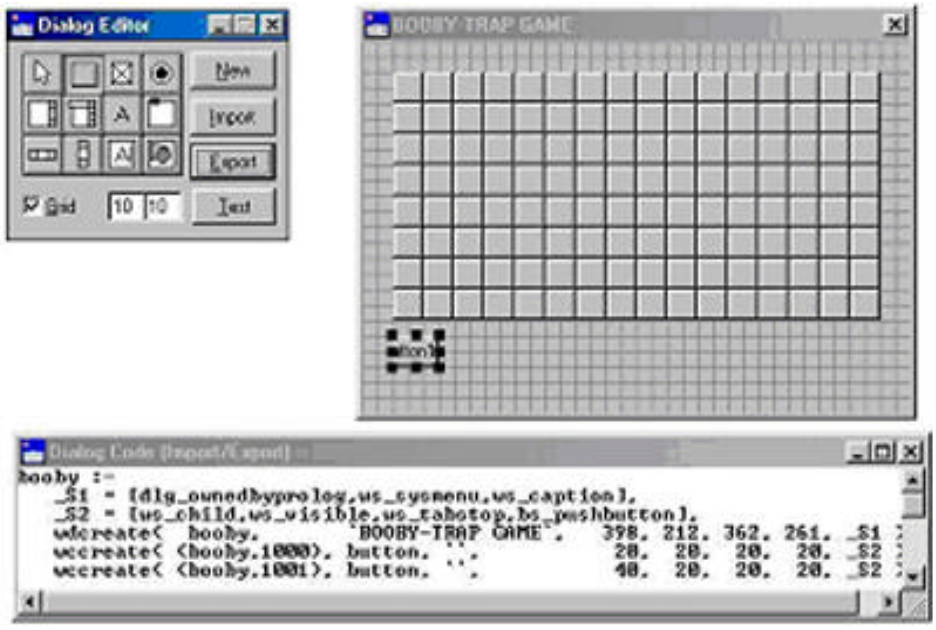

Figure 4: selects the button control from the toolbox and places it on the scratch window

References

1. Schildt, H. (1987). Advanced Turbo Prolog Version 1.1. Osborne Mc-Graw-Hill: California.

2. Tarau, P. (1995). BinProlog 4.00 User Guide. Download from *citeseer.nj.nec.com /2676.html* on 18 Januari 2003.

3. Brisset, P., and Ridoux, O. (1994) The architecture of an implementation of LambdaProlog: Prolog/Mali. In Proceedings of the Workshop on Implementation of Logic Programming, ILPS'94, Ithaca, NY. The MIT Press, November 1994. Download from *http://citeseer.nj.nec.com/article/brisset95architecture.html* on 18 Januari 2003.

4. Covington, M. A., Nute, D., and Vellino, A. (1997). Prolog Programming in Depth. Prentice-Hall: New Jersey.

5. Somogyi, Z., Henderson, F., Conway, T, and O'Keefe, R. (1995) Logic Programming for the Real World. In Donald A. Smith (Ed.), Proceedings of the ILPS'95 Postconference Workshop on Vision for the Future of Logic Programming. pp: 83-94, Portland, Oregon.

6. Roth, A., and Spenser, C. (1993) The Benefits of Prolog. Software Development. Miller Freeman Inc.

Imperative Programming

The imperative programming style describes computation in terms of program state changes and statements that change the program state. An example of a program state change would be selecting the "reply" button in an email program. Before that, the email program was displaying an email for you to read. After the state change, you now have a new email that is ready for you to edit (not just read).

Imperative programming languages have primitives very similar to the CPU's machine instructions. For example, Branch statements, memory assignment statements and add instructions. Imperative programs consist of a series of commands for the computer to perform. Almost all computer hardware is imperative — designed to execute the native code of the computer — machine code. At this level, the memory contents defined the state of the computer. Higher-level imperative style programming languages use variables to contain this state information.

Even though logic programming languages are theoretically superior to imperative programming languages such as C, C++ and Java, there are two reasons why current logic programming languages such as Prolog are not as widely used for application development.

1. Logic programming languages can be significantly slower than the equivalent program logic in an imperative language such as C. Therefore, application designers concerned with performance might not consider logic programming languages. As computer speeds continue to increase, and logic-programming languages become even more efficient, this reason could someday be eliminated.

2. Current logic programming languages do not detect as many errors in programs as compilers for imperative programming languages, which can reduce productivity. Programmers must find more errors themselves, usually during debug. Languages, such as Prolog, do not require type casting, which simplifies programming and increases flexibility. However, it also means the programmer is now responsible for finding many errors that a compiler with type checks would find.

7. Luger. G. F. (2002). Artificial Intelligence: Structures and Strategies for Complex Problem Solving. Addison-Wesley: US.

8. Bratko, I. (1998) PROLOG Programming for Artificial Intelligence. Addison-Wesley: US.

9. Bringsjord, S. (2002) AI Research to AI Business, and Back: Automatic Story Generation and Intelligent Document Production. PC AI January/February 2002, pp: 36 – 43.

10. Dixon, M. L. E., Grant, P. W., Moseley, L. G., and Spenser, C. (1998) A Flex-based Expert System for Sewage Treatment Works Support. PC AI, 12(4), 35 – 38.

11. Intelligent Systems Laboratory (1995) SICStus Prolog User's Manual. Download from *citeseer.nj.nec.com/482697.html* on 18 January 2003.

12. Murphy, T. (1993) As you know, I love Prolog: LPA 386 Prolog. AI Expert. Miller Freeman Inc.

13. Spenser, C. (1997) LPA Prolog in Action. PC AI, 11(6), 40-42.

14. Taha, Z. (1988) Pemanduan Arah Robot Menggunakan Prolog. Siri Seminar Sains Komputer II, 18.1 – 18.7. Universiti Kebangsaan Malaysia: Bangi.

15. Westwood, D. (1999). LPA Win-Prolog 3.6 Technical Reference. Logic Programming Association.

Wan Hussain Wan Ishak is on the Faculty of Information Technology Universiti Utara Malaysia and can be reached at hussain@uum.edu.my

Appendix

```
% abolishing and declaring id/1 as a dynamic predicate
:- abolish(id/1), dynamic(id/1).

% creating dialog
create_booby_dialog:-
   _S1 = [dlg_ownedbyprolog,ws_sysmenu,ws_caption],
   _S2 = [ws_child,ws_visible,ws_tabstop,bs_pushbutton],
   _S3 = [ws_child,ws_visible,ss_right],
   _S4 =
[ws_child,ws_visible,ws_tabstop,ws_border,es_left,es_multiline,es_autohscroll,es_autovscroll],
   wdcreate(  booby,        `BOOBY-TRAP GAME`,  398, 212, 362, 271, _S1 ),

   % create buttons
   buttons(1,1,20,40,_S2),

   wccreate( (booby,1128),  button, `CLOSE`,    270, 210,  70,  30, _S2 ),
   wccreate( (booby,10000), static, `Score : `, 170,  10,  70,  20, _S3 ),
   wccreate( (booby,8000),  edit,   `0`,        250,  10,  90,  20, _S4 ),
   wccreate( (booby,1129),  button, `NEW`,       20, 210,  70,  30, _S2 ).

% create 128 buttons - then stop
buttons(Num,128,Left,Top,Prop):-
        !,
        createbutton(128,Left,Top,Prop).

% reset counter to 1
% only 16 buttons in a row
buttons(17,Id,Left,Top,Prop):-
        NewTop is Top + 20,
        buttons(1,Id,20,NewTop,Prop).

% create the buttons
buttons(Num,Id,Left,Top,Prop):-
        createbutton(Id,Left,Top,Prop),
        NewNum is Num + 1,
        NewId is Id + 1,
        NewLeft is Left + 20,
        buttons(NewNum,NewId,NewLeft,Top,Prop).

% create the button
createbutton(Id,Left,Top,Prop):-
        wccreate( (booby,Id),  button, ``, Left,  Top,  20,  20, Prop ).
```