Universität des Saarlandes
Fachrichtung 6.2 – Informatik
J. Siekmann, S. Autexier, C. Benzmüller
C. Brown, C. Hahn

1st Practical Assignment in Artificial Intelligence (SS 2005)

Issued: May 9, 2005 Due: May 23, 2003

- **Please submit your solution by sending an email to your tutor. The e-mail address of your tutor can be found on the webpage.**
- **Indicate the name and matriculation number of each student working in your project.**

Exercise P1.1: **100+20 P**

Develop and implement in Lisp a strategy to play blind Tic-Tac-Toe (see Exercise 3.4, 3rd theoretical assignment).

1. Your program must run and fulfill the interface requirements given below. (20 P)
2. Your program must be well documented und comprehensible. (40 P)
3. Explain your strategy and justify why you choose that strategy. (40 P)

We will set up a tournament and have play the programs against each other. The programmers of the seven best programs will be awarded with additional points as follows:

Position	1	2	3	4	5	6	7
Points	20	12	8	5	3	2	1

In addition, the programmers of the winning program will get a prize donated by Prof. Siekmann.

Game: The game master first initializes an empty board, which cannot be seen by the players. Then he alternately asks the players for a move. If the field is already taken, the player is asked for a different move instead. The player who first places three tokens in a row, column, or diagonal wins.

Interface: The game master is a function that takes two functions or closures as parameters. Your program will be one of the parameters, the second parameter is your opponent's program. The game master calls your function whenever it is your turn.

Your function needs to take two arguments *action* and *count*. The argument *count* is the total number of tokens already put on the board. If the argument *action* is `nil`, then it is your first try of a move. If you tried to put a token on a field that was already taken by the opponent, *action* is a list `(`*row col*`)` specifying the field you tried to take.

Your function needs to return a list of two numbers `(`*row col*`)` specifiying the field you want to take. Both *row* and *col* range from 0 to 2:

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2

You have to use the following template for your function:

```
0:  ;;; <First name> <Last name>
1:  (labels ((compute-move (x) ...)
2:           ...                    ; possibly more local functions you need
3:           (empty-board () ...)   ; initialize empty board
4:           (set-pos (row col board mark) ...) ; set token in board
5:           )
6:    (let ((board (empty-board))
7:          ...                     ; possibly more local variables you need
8:          )
9:      #'(lambda (action count)
10:         (unless (null action)
11:            ;;; store the opponent's moves
12:            (setf board
13:                  (set-pos (first action) (second action) board 'o)))
14:         (let ((pos (compute-move action)))
15:           ;;; store my own moves
16:           (setf board
17:                 (set-pos (first pos) (second pos) board 'x))
18:           pos))))
```

It is important that you define all your functions and variables locally using `labels` and `let`, respectively. Otherwise, name conflicts might occur during the tournament, since both your and your opponents programs need to be loaded. Note that `labels` allows you to write recursive functions (as opposed to `flet`).

In Lines 1-4 you define your local functions and in Lines 5 and 6 your local variables. Line 5 initializes your local board. Since you cannot see the game master's board, you have to do the bookkeeping yourself.

Lines 9-18 provide the closure that calculates your next move using *`action`* and *`count`*. Lines 10-12 marks a field with `o` when your opponent has taken it before. Line 14 calculates your move and Lines 16 and 17 mark the field you take with `x`.

Note that you have to write the functions `empty-board` and `set-pos` yourself. The function `empty-board` returns an empty board. The call (`set-pos` *`row col board mark`*) sets the token *`mark`* on the field (*`row col`*) in the board *`board`* and returns the board *`board`*.

Ensure the following properties of your program:

- You must not choose a move of which you can know that your opponent has done it before.
- Your move must address an existing field in the board.
- Your function must not run longer than one minute for one move (CPU time on a Pentium 4 PC with about 1 GHz).