<u>Introduction to Artificial Intelligence Coursework</u>[1]

(Submission by groups of two students allowed/welcome)
<u>Due date: Thursday 3 March 2016</u>
Electronic submission (code + this worksheet) on CATE

<u>1. Introduction</u>

We will be looking at a two player board game called "war of life" which will be played on an 8x8 board. Player 1 will start with a random configuration of 12 blue pieces and player 2 will start with a similar random configuration of 12 red pieces. An example initial configuration might be (where `b` stands for "blue piece" and `r` stands for "red piece"):

	1	2	3	4	5	6	7	8
1								r
2		r						
3				b	b		r	b
4	b	b				r		
5		b	r	b		b	b	
6		b				r		
7			b		b	r	r	r
8			r				r	r

We call the board places where pieces can be placed *cells* (there are 64 cells on an 8x8 board). In the game, player 1 goes first and moves one of his/her pieces. A piece can be moved to one of its neighbour cells (vertically, horizontally or diagonally) as long as no other piece is occupying the cell to be moved to. So, for example, the blue piece at (3, 8) can move to (2,7), (2,8) or (4,7) or (4,8), but not (3,7) because there is a red piece there already. We say that a piece is *surrounded by* the pieces in neighbouring cells.

There is a twist: after <u>each</u> player moves, "life" on the board "evolves" according to the following rules (referred to as <u>Conway's Crank</u>):

For each of the (64) cells C on the board:

- If C contains a piece and the piece is surrounded by 0 or 1 other pieces, then the piece dies of loneliness and is taken away (i.e., the cell becomes empty).
- If C contains a piece and the piece is surrounded by 4, 5, 6, 7 or 8 pieces, then the piece dies of overcrowding and is taken away.
- (If C contains a piece and the piece is surrounded by 2 or 3 pieces, then it is happy and survives.)
- If C is empty and C is surrounded by
  - 2 blue pieces and 1 red piece, or
  - 3 blue pieces,

  then a blue life is born and C is filled with a blue piece.
- If C is empty and C is surrounded by
  - 2 red pieces and 1 blue piece, or
  - 3 red pieces,

  then a red life is born and C is filled with a red piece.

The game terminates as follows:

- If at some stage no (red or blue) pieces at all are left on the board, then the game is *drawn*.
- If, when it is his/her turn, a player cannot move anywhere, then the game is declared a *stalemate* and is *drawn*.
- If one player has no pieces left on the board, then that player *loses* and the other player wins.
- If the game lasts for 250 moves without a winner, then it is declared an *exhausted draw*.

Part 1 – Getting to know the Game

Question 1

	1	2	3	4	5	6	7	8
1	b							r
2		r				b		r
3				b	b		r	b
4	b	b				r		
5			r				b	
6		b				r		
7	b		b		b	r	r	r
8			r				r	

Draw the board state after a turn of Conway's Crank, given the left board.

	1	2	3	4	5	6	7	8
1							r	
2					b	b		r
3	b	b	b		b			b
4		b	b	b	b	r		b
5	b		r			r	b	
6		b	b	b	r			r
7			b	b	b			r
8		b		b			r	r

Download the Prolog program `war_of_life.pl` from CATE.

This provides a set of predicates for playing the game:

Top Level Predicates in File

`start_config(+Configuration, -InitialBoardState)`
This returns an initial randomised board state with 12 pieces for each player on an 8x8 board.

`draw_board(+BoardState)`
Given a board state in the format described below, this predicate will present it on screen.

`next_generation(+BoardState, -NextGenerationBoardState)`
This performs a Conway Crank and produces the next generation board state.

`play(+Showboard, +FirstPlayerStrategy, +SecondPlayerStrategy, -NumberOfMoves, -WinningPlayer)`
This will play a game given the strategy of player 1 and the strategy of player 2. The `+Showboard` variable is either set to `verbose`, in which case it will print out the board states as the game progresses, or `quiet`, in which case it just returns an answer, namely the `NumberOfMoves` in the completed game and the colour of the `WinningPlayer`.

Board states are represented in the program as pairs of lists, where the first list contains the co-ordinates of all the alive blue pieces and the second list contains the co-ordinates of all the alive red pieces. For example, this is a simple board state with two alive blues and one alive red:

[[[3,4],[5,7]],[[8,8]]]

Question 2

In the box below, write down the Prolog representation for the initial board state given in question 1.

[[[1,1],[2,6],[3,4],[3,5],[3,8],[4,1],[4,2],[5,7],[6,2],[7,1],[7,3],[7,5]], [[1,8],[2,2],[2,8],[3,7],[4,6],[5,3],[6,6],[7,6],[7,7],[7,8],[8,3],[8,7]]]

Use this representation, and the predicate

`next_generation(+BoardState, -NextGenerationBoardState)`

to check whether your answer for question 1 was correct. If it is, then run a further two generations, and put the resulting board states in the tables below:

3rd Generation:

	1	2	3	4	5	6	7	8
1						b	r	
2		b		b	b	b		r
3	b							b
4								b
5	b							b
6								r
7						r		r
8				b	b		r	r

4th Generation:

	1	2	3	4	5	6	7	8
1						b	r	
2					b	b		r
3					b			b
4							b	b
5							b	b
6								r
7					b	r		r
8					b	r	r	r

Question 3

In a Prolog shell, load the file `war_of_life.pl` and run this query:

`play(verbose, random, random, NumMoves, WinningPlayer).`

This will play a game of war of life. Each player will randomly move a piece until the game is won or drawn. The predicate records how many moves there were in the game and who won. Run this a few times to get a feel for what it does and how the games progress when players choose randomly.

Now open a new file called `my_wol.pl`. In the file, write a (Sicstus) Prolog program to act as a wrapper for the `play/5` predicate. In particular, you should write a predicate called `test_strategy/3` which takes three inputs: the number of games, N, to play, the strategy for player 1 and the strategy for player 2. When run, the predicate will play the war of life game N times and tell you (print to screen) how many draws and how many wins for each player there have been, the longest, shortest, and average moves in a game, and the average time taken to play a game. Use the `test_strategy/3` predicate to run the game 1000 times, with both players moving pieces randomly. Record the results in this box:

Number of draws	52
Number of wins for player 1 (blue)	493
Number of wins for player 2 (red)	455
Longest (non-exhaustive) game	49
Shortest game	2
Average game length (including exhaustives)	11.4
Average game time	0.10s

Question 4

Does it look like there is any advantage to playing first if both players choose moves randomly? Answer in the space below.

From the results below, it does look like playing first is an advantage if pieces are moved randomly. However I ran the random strategy 10000 times twice and for both of the runs, the ratio blue_wins/red_wins was very close to 1: the first one had a ratio of 1.01 which means that the first player won more times than the second player; the second test I got a ratio of 0.99 which mean that the second player won more games. The blue_wins/red_wins ratio being very close to 1 and alternating between >1 and <1 means that there is no real advantage to playing first if both players move their pieces randomly.

Part 2 – Implementing Strategies

In this part, we will be implementing search strategies in order to improve a war of life playing agent's performance. They already have one strategy: `random`, which chooses any piece randomly and moves it randomly. The question is: can we implement any strategy which out-performs this one?

We will look at four strategies:

Bloodlust:

This strategy chooses the next move for a player to be the one which (after Conway's crank) produces the board state with the fewest number of opponent's pieces on the board (ignoring the player's own pieces).

Self Preservation:

This strategy chooses the next move for a player to be the one which (after Conway's crank) produces the board state with the largest number of that player's pieces on the board (ignoring the opponent's pieces).

Land Grab:

This strategy chooses the next move for a player to be the one which (after Conway's crank) produces the board state which maximises this function: Number of Player's pieces – Number of Opponent's pieces.

Minimax:

This strategy looks two-ply ahead using the heuristic measure described in the Land Grab strategy. It should follow the minimax principle and take into account the opponent's move after the one being chosen for the current player.

In your file `my_wol.pl`, write five predicates (use Sicstus Prolog):

```
bloodlust(+PlayerColour, +CurrentBoardState, -NewBoardState, -Move).
self_preservation(+PlayerColour, +CurrentBoardState, -NewBoardState, -Move).
land_grab(+PlayerColour, +CurrentBoardState, -NewBoardState, -Move).
minimax(+PlayerColour, +CurrentBoardState, -NewBoardState, -Move).
```

These predicates will implement the four strategies described above by choosing the next move for a player. They will all do the same thing: choose the next move for the player. `PlayerColour` will either be the constant `b` for blue or `r` for red. The `CurrentBoardState` will be the state of the board upon which the move choice is going to be made. The predicate will produce a `NewBoardState`, in the usual representation (pair of lists) which will represent the board state after the move, but <u>before</u> Conway's crank is turned. The predicate will also return the `Move` that changed the current to the new board state. This will be represented as a list [r1,c1,r2,c2] where the move chosen is to move the piece at co-ordinate (r1, c1) to the <u>empty</u> cell at co-ordinate (r2, c2).

It should be possible to run these strategies in the same way as the random strategy, using the constants `bloodlust`, `self_preservation`, `land_grab`, `minimax`. For example, if you load `war_of_life.pl` and `my_wol.pl` into Prolog, then type the query:

```
play(verbose, bloodlust, land_grab, NumberOfMoves, WinningPlayer).
```

this will play a game in which player 1 uses the bloodlust strategy and player 2 uses the land_grab strategy. Check that this is working OK by running a few games with different strategy pairings.

Part 3 – A Tournament

Question 5

We want to know which, if any, strategy is the best for playing this game, and we'll do this by using a tournament. Play as many games as time will allow for each pairing of strategies, and fill in the table over the page.

Question 6

If both players have the same strategy, for which strategies does it appear that playing first is an advantage/disadvantage? Answer in the space below:

If both players have the same strategy, it would appear that playing first:
- is an advantage for strategies bloodlust, land grab and minimax since the ratio (P1 wins)/(P2 wins) is far greater than 1: 1.479, 1.083 and 1.344 respectively
- does not matter for strategies random, self preservation since ratio (P1 wins)/(P2 wins) is very close to 1: 1.006 and 1.005 respectively

Question 7

Imagine playing football in a gale force wind and where the pitch is extremely muddy. Here, it is hardly worth the players kicking the ball, because the environment plays too big a factor in the game. What evidence do you have against the claim that the environment plays a bigger part than the movement of pieces in the war of life game? The environment here is Conway's Crank, which is beyond the control of the players. Answer in the space below:

All strategies use Conway's Crank as their environment however the outcome of each of them can be considerably different. For example, if P1 moves his pieces adopting the minimax strategy and P2 the random strategy, P2 only wins twice out of 800 games in my simulation: P1 almost always wins. If P2 were to use the minimax , then out of 500 games he would win 186 times which are far better odds then in the previous scenario. This means that the way the players move their pieces has plays a bigger part than the environment.

Submitting Your Work

Electronic submission: submit
1) Your code `my_wol.pl` (written in Sicstus Prolog and executable on the lab machines). Please do not include or load (or ensure_loaded) `war_of_life.pl` in `my_wol.pl`! (Yes, this means `my_wol.pl` will not work on its own, but this is crucial for the autotesting) Furthermore, use `play(quiet,…)` rather than `play(verbose,…)` and don't include any write-statements other than the necessary ones in `test_strategy` (like no. of draws, average time, …).

2) A file worksheet.pdf (your completed version of this worksheet).

P1 strategy	P2 strategy	Games Played	P1 wins	P2 wins	Draws	Av. Game Length (moves)	Av. Game Time (seconds)
Random	Random	2000	953	947	100	11.5	0.01
Random	Bloodlust	2000	348	1551	101	7.3	0.109
Random	Self Preservation	1000	103	871	26	14.1	0.232
Random	Land Grab	2000	117	1857	26	10.5	0.198
Random	Minimax	500	6	490	4	10.7	3.759
Bloodlust	Random	2000	1606	295	99	7	0.132
Bloodlust	Bloodlust	2000	1077	728	195	8	0.184
Bloodlust	Self Preservation	2000	1043	851	106	12.8	0.306
Bloodlust	Land Grab	2000	788	1108	104	10.3	0.259
Bloodlust	Minimax	1000	149	824	27	9.9	3.113
Self Preservation	Random	2000	1761	208	31	14.2	0.34
Self Preservation	Bloodlust	2000	992	904	104	12.7	0.381
Self Preservation	Self Preservation	2000	996	991	13	49.1	1.12
Self Preservation	Land Grab	1000	188	803	9	26.5	1.171
Self Preservation	Minimax	800	20	778	2	22.4	14.23
Land Grab	Random	2000	1870	93	37	10.5	0.254
Land Grab	Bloodlust	1500	974	456	70	9.6	0.31
Land Grab	Self Preservation	1000	800	192	8	26.7	1.1
Land Grab	Land Grab	1000	507	468	25	21.3	0.877
Land Grab	Minimax	1000	82	903	15	21.2	15.689
Minimax	Random	800	798	2	0	9.8	6.952
Minimax	Bloodlust	500	483	11	6	10.4	4.392
Minimax	Self Preservation	600	594	6	0	21.1	8.713
Minimax	Land Grab	800	781	14	5	19.7	9.24
Minimax	Minimax	500	250	186	164	90.2	114.343