CUHK CSCI3230

Prolog Programming Assignment – Pentago

Due date: 23:59:59 (GMT +08:00), 15th December, 2020

Introduction

Pentago is a two-player board game played on a 6x6 board, which is divided into four 3×3 sub-boards (or quadrants). This game is similar to tic-tac-toe. One player owns the red playing pieces (i.e. the red marbles) and the other one owns the black playing pieces (i.e. the black marbles). In each turn, one player places one of his/her marbles on the board and rotates a quadrant by 90 degrees. The goal of this game is to create a vertical, horizontal or diagonal row of five marbles.

In this assignment, you are required to implement a simple *Pentago player program*, which can help you win the game.

Requirement

In this assignment, you are required to implement two important Prolog predicates:

```
threatening(Board,CurrentPlayer,ThreatsCount).

pentago_ai(Board,CurrentPlayer,BestMove,NextBoard).
```

The first one "threatening" calculates the number of threats (introduced later) that the CurrentPlayer makes for his/her opponent. The second one "pentago_ai" finds the best move for the CurrentPlayer given the current board. Some related predicates can be defined and used in your program to support the inference of "threatening" and "pentagon_ai".

Board Representation

Pentago is a two-player (say Black player vs. Red player) board game. As shown in figure 1, the 6x6 board has 36 positions for the players to put their marbles on. There are 4 quadrants on the board which are labelled as *top-left*, *top-right*, *bottom-left* and *bottom-right*, respectively. These quadrants can be rotated separately (figure 2).

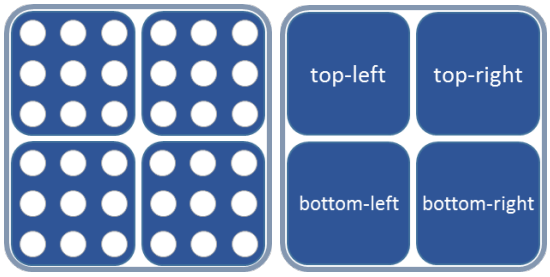


Figure 1. An empty 6x6 board with 3x3 sub-boards

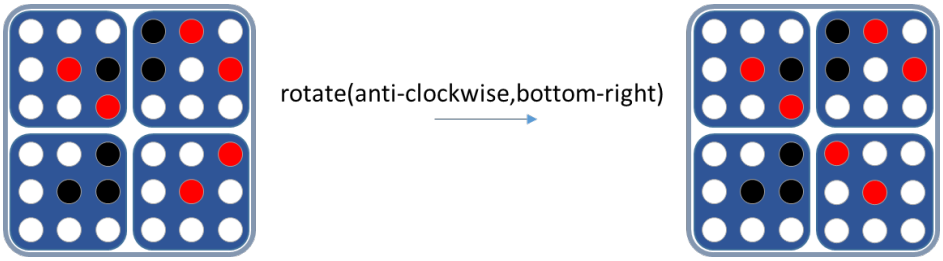


Figure 2. An example for a rotation of 90 degrees on the bottom-right quadrant in anti-clockwise.

The holes on the 6x6 board are represented using a rectangular grid. Each position is labelled as shown in figure 3. A position can only be in either one of the three states:

1. Occupied by a black marble
2. Occupied by a red marble
3. Unoccupied (i.e. no marble put on it)

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

Figure 3. Coordinates of each position in the board

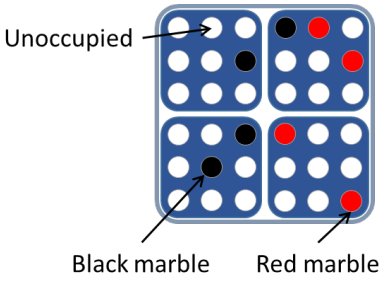


Figure 4. A position can only be in one of the three states.

We will use ***board(BlackL,RedL)*** to represent a board state. `BlackL` is a list in Prolog and stores the positions occupied by black marbles; `RedL` is a list in Prolog and stores the positions occupied by red marbles. The unoccupied positions can be inferred from them. For example, `board([],[])` represents the empty board in figure 1. `board([4,9,21,26], [5,12,22,36])` represents the board in figure 4.

Game Flow

The goal of this game is to create a vertical, horizontal or diagonal row of five marbles in your chosen color. The game starts with an empty board. The players take turns during the game. In each turn, the player is required to make the following two actions:

1. Marble placement: Place a marble of the chosen color on an unoccupied position
2. Quadrant rotation: Rotate a quadrant by 90 degrees either in anti-clockwise or clockwise direction

The game ends when any player creates a vertical, horizontal or diagonal row of five marbles of the same color (either before or after the quadrant rotation in their move). Looking at the board in the ending state, the player wins if he/she owns five marbles of the same color in a row. As listed in the Appendix, there are in total 32 ways to place the five marbles in a row on an empty board. Some winning cases are shown in figure 5. Note that it is possible for the two players to win this game at the same time (figure 5c). It is a draw when neither of the two players could make five marbles of the same color in a row at last. Under this ending condition, since marble placement never prevents the current player from winning but quadrant rotation can (figure 5d, if Black player rotates top-right quadrant before putting a black marble, Red player wins.), we assume that the player will always place a marble before quadrant rotation. A move is a structure in Prolog:

```
move(Position,Direction,Quadrant)
```

where

Position refers to the place where the new marble will be put on the existing board, {1,2,…,36}

Direction refers to the direction of rotation of a quadrant, {anti-clockwise, clockwise}

Quadrant refers to the 4 possible quadrants, {top-left, top-right, bottom-left, bottom-right}

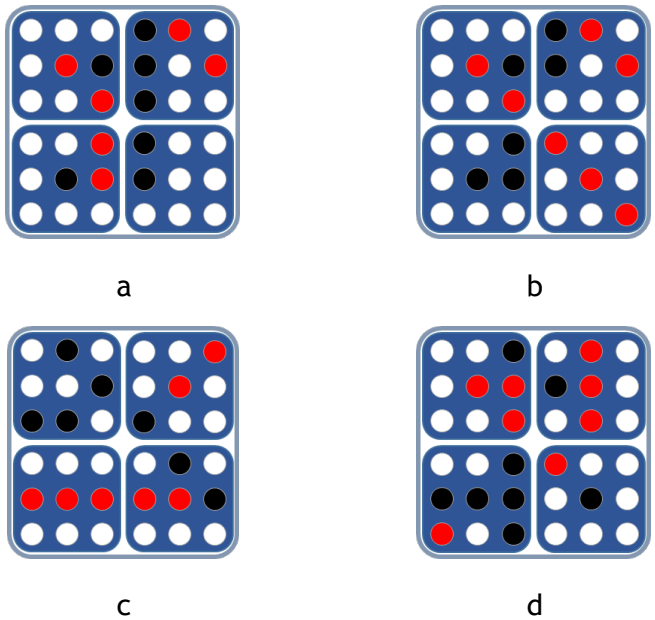


Figure 5. (a) Five black marbles in a vertical row. (b) Five red marbles in a diagonal row. (c) Five black marbles in a diagonal row and at the same time five red marbles in a horizontal row. (d) Nobody wins at the moment.

Threatening

This section describes the Prolog predicate

`threatening(Board,CurrentPlayer,ThreatsCount).`

As listed in the Appendix, there are only 32 ways to place the five marbles in a row on an empty board. We call each of these ways as a *winning row*. The current state of the board is stored in variable `Board`, which is represented as structure `board(BlackL,RedL)`. `CurrentPlayer` is threatened when any of the winning rows is filled with exactly 4 opponent's marbles with 1 empty slot, which we call each of them as a *threatening row*. From the perspective of the `CurrentPlayer`, the degree of the threat is defined as the number of threatening rows produced on a board by the opponent and is stored in `ThreatsCount`. Referred to figure 6, the number of threatening rows with respect to Black player is 3, i.e. [1,8,15,22,29], [8,15,22,29,36] and [5,11,17,23,29]. The number of threatening rows with respect to Red player is 0 (i.e. not threatened). The more threatening rows you create for your opponent, the higher the chance of winning. In your turn, the less threatening rows you encounter, the higher the chance of winning.

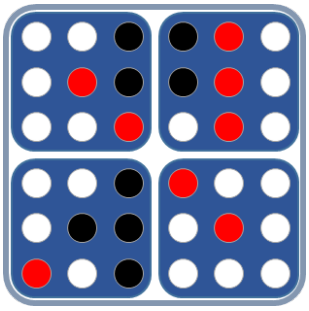


Figure 6. Coordinates of each position in the board.

On Choosing the Best Move

In this assignment, we will implement a greedy strategy in the Pentagon AI program on choosing the best move(s) by using the predicate:

`pentago_ai(Board,CurrentPlayer,BestMove,NextBoard).`

Your program will search the states of the board one step ahead (e.g. Black player's turn → Red player's turn). The choice of the move by `CurrentPlayer` is stored in `BestMove` as a `move` structure. We denote the two turns as T_n and T_{n+1} respectively, hence three layers of board states searching. Assume it is `CurrentPlayer`'s turn.

The goal of `CurrentPlayer` is to win this game as soon as possible given the current board state `Board`. `CurrentPlayer` always executes a winning move if it exists in turn T_n. Otherwise, `CurrentPlayer` player will think one turn ahead and prevent the rational opponent from winning in turn T_{n+1} unless both players win at the same time. The opponent will try his/her best to win the game and threaten `CurrentPlayer`. `CurrentPlayer` prefers a draw to a loss and will choose any move as `BestMove` if he/she must lose the game.

When neither of players wins after turn T_{n+1} ends, `CurrentPlayer` player will reduce threats made by the opponent as much as possible. How to count the threats has been discussed in the previous section. After deciding the `BestMove`, it will be executed and

will give a new board *NextBoard*. The partial search tree formed by your program is shown in figure 7.

Note that if the player can only win by placing a marble without a quadrant rotation, the program should produce a *NextBoard* containing five marbles in a row. Only the *Position* of *BestMove* will be checked in this case while *Direction* and *Quadrant* will be ignored.

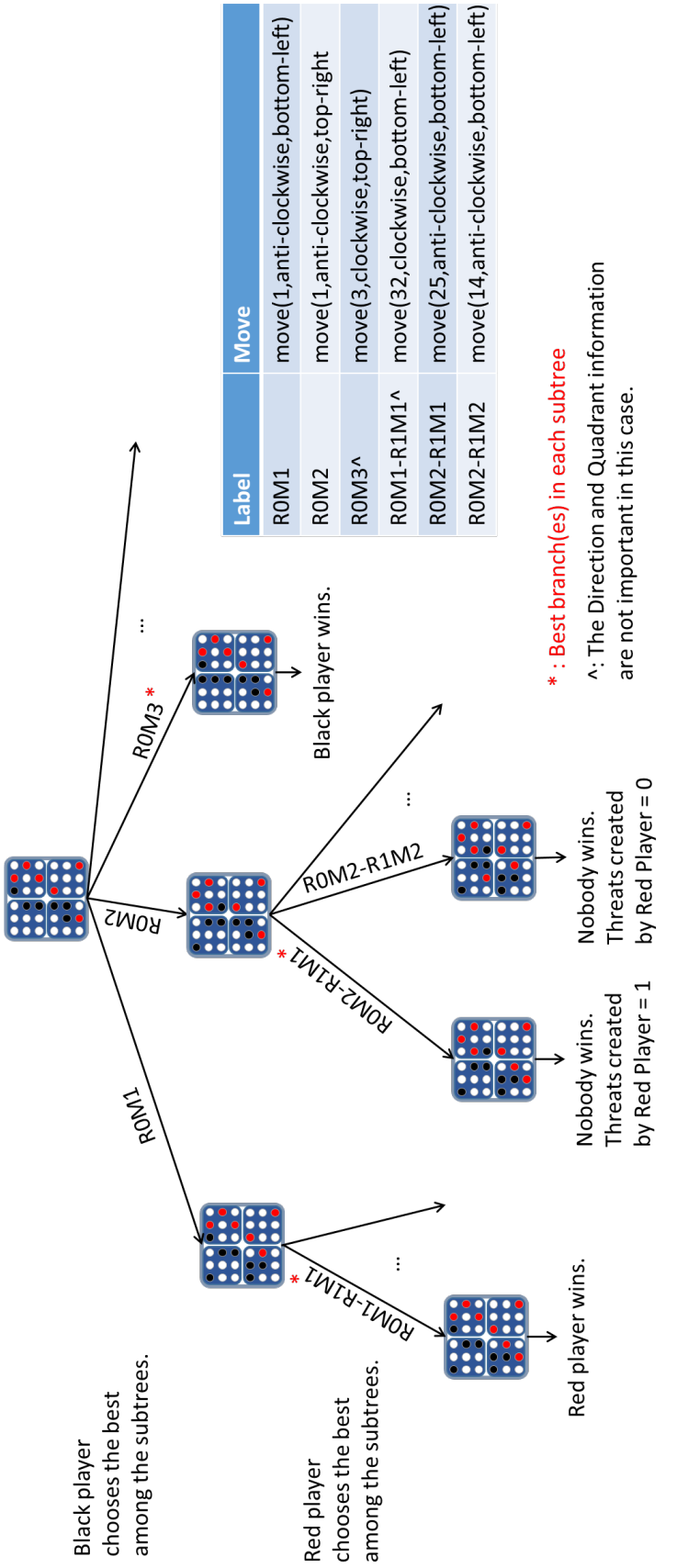
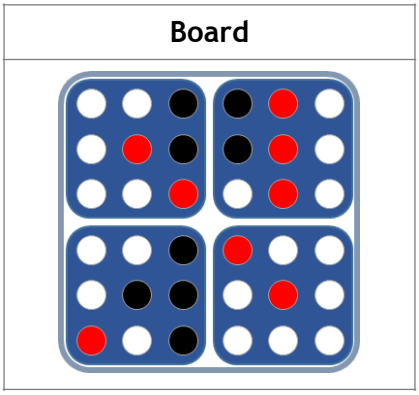


Figure 7. Search tree by the AI program.

Illustration

Example 1



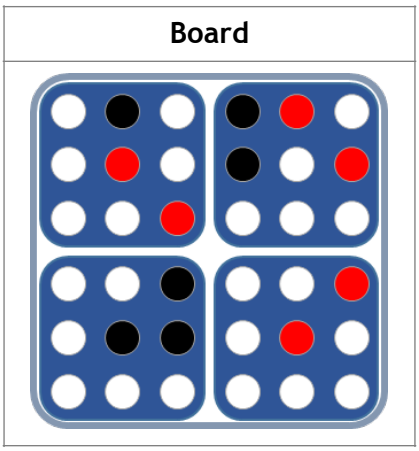

```
?- threatening(board([3,4,9,10,21,26,27,33],
[5,8,11,15,17,22,29,31]),black,S).

S = 3.

?- threatening(board([3,4,9,10,21,26,27,33],
[5,8,11,15,17,22,29,31]),red,S).

S = 0.
```

Example 2



Note that the solution for this problem may not be unique.

```
?- pentago_ai(board([2,4,10,16,21,26,27],
[5,8,12,15,24,29]),red,
BestMove,NextBoard).

BestMove = move(34,anti-clockwise,bottom-right).
```

```
NextBoard = board([2,4,10,16,21,26,27],[5,8,12,15,22,29,36]).
```

More Examples

For simplicity, one of the best moves is shown here.

```
?-pentago_ai(board([1,4,10,13,16,21,26,27,30],
[5,6,8,12,15,22,24,29,31]),black,BestMove,NextBoard).

BestMove = move(36, clockwise, top-left)
```

```
?-pentago_ai(board([1,4,7,10,18,21,26,27],
[3,5,8,12,22,24,25,29,31]),black,BestMove,NextBoard).

BestMove = move(11, anti-clockwise, top-right)
```

```
?-pentago_ai(board([1,5,6,13,16,26,32,33,36],
[3,8,11,12,17,19,20,22,24,29]),black,BestMove,NextBoard).

BestMove = move(21, anti-clockwise, top-right)
```

Marking Scheme

In the marking, we will use SWI-Prolog **version 5.10.4.** There will be ten test cases (five for "threatening" and five for "pentago_ai") and each case contributes 10% of the total marks of the assignment. You can find old versions of SWI-Prolog here http://www.swi-prolog.org/download/stable?show=all.

Submission

You should submit **a single file *student-id*.pl** through our online submission system, e.g. if your student ID is 1301234567, the filename should be 1301234567.pl. Please limit the file size to be less than 1MB. You can submit multiple times (before the deadline), and the final grade will be the highest mark of all the submissions for this assignment. Make sure that you have **removed all the *unrelated* predicates or statements** in your .pl file before submission. Also, make ensure that your program can return the results for all the test cases in 1 minute in our machine. If your program fails to do so, **zero** mark will be given.

Late submission will lead to marks deduction which is subject to the following penalty scheme.

Number of Days Late	Marks Deduction
1	10%
2	30%
3	60%

4 or more	100%

Plagiarism will be seriously punished.

Suggestions

1. Study predicate for sets.
2. Be careful on the minimax procedure.
3. Do not perform exhaustive search and early stop is necessary.
4. Use

   ```
   profile(pentago_ai(Board,CurrentPlayer,BestMove,NextBoard)).
   ```

   to optimize your program.

Extension of the assignment

This part is for students who are interested in extending the current program **after** submitting a workable version of your assignment. You may modify the program so as to make it into real Pentago player program. You may also use alpha-beta pruning technique in choosing the best moves.

References

Rules of Pentago: http://www.creativedeployment.com/clients/mindtwister/strategy-guides/MTUSA_Pentago%20Rules-Strategy%20Guide%202015.pdf

Reference manual of SWI-Prolog: http://www.swi-prolog.org/pldoc/refman/

Appendix

32 ways to make five marbles in a vertical, horizontal, or diagonal row

[1,7,13,19,25]	[5,11,17,23,29]	[8,9,10,11,12]	[13,14,15,16,17]
[1,2,3,4,5]	[5,10,15,20,25]	[8,15,22,29,36]	[14,15,16,17,18]
[1,8,15,22,29]	[6,12,18,24,30]	[9,15,21,27,33]	[19,20,21,22,23]
[2,8,14,20,26]	[6,11,16,21,26]	[10,16,22,28,34]	[20,21,22,23,24]
[2,3,4,5,6]	[7,13,19,25,31]	[11,17,23,29,35]	[25,26,27,28,29]
[2,9,16,23,30]	[7,8,9,10,11]	[11,16,21,26,31]	[26,27,28,29,30]
[3,9,15,21,27]	[7,14,21,28,35]	[12,18,24,30,36]	[31,32,33,34,35]
[4,10,16,22,28]	[8,14,20,26,32]	[12,17,22,27,32]	[32,33,34,35,36]

Have fun!