Prolog 실습 과제 : 지뢰찾기

작성자 : 2019134006 김준섭

1. ?-open\_at(Game, X, Y, Status).
   1. 구성
      1. 문법 : open\_at(Game, X, Y, Status).,
      2. 기능 : 해당 위치의 지뢰 포함 여부 및 주변 지뢰 개수를 표시
      3. 입출력 :
         1. 입력 :
            1. Game : 게임 이름 (ex. game)
            2. X : 열어볼 지점의 X 좌표
            3. Y : 열어볼 지점의 Y 좌표
            4. Status : 상태 정보 저장 공간 🡨 변수에 값 저장
         2. 출력 :
            1. 함수 출력: 없음 (무조건 true)
            2. 변수 출력: 상태 정보 저장 공간

X, Y에 지뢰가 있는 경우: Status = bomb

X, Y 주변에 지뢰가 없는 경우: Status = empty

X, Y 주변에 지뢰가 있는 경우: Status = 폭탄의 개수

?- open\_at(game,4,1,S).

S = bomb .

?- open\_at(game,1,1,S).

S = empty .

?- open\_at(game,3,1,S).

S = 1 .

* 1. 구현 방법
     1. open\_at/4를 구현하기 위해 주변의 8칸에 존재하는 지뢰의 개수를 세는 rule count(Game, X, Y, N)를 새롭게 정의하고, 이를 open\_at(Game, X, Y, Status) 계산에 활용하였다.
        1. count(Game, X, Y, N)
           1. 인자

Game: 게임 이름 (ex : game)

X, Y: 셀의 좌표.

N: 주어진 셀 주변의 폭탄 수.

* + - * 1. 작동방식

size(Game, SizeX, SizeY): 게임 보드의 크기를 가져오기

X1 is max(X-1, 1) 및 X2 is min(X+1, SizeX): 내장 predicates인 min/2, max/2를 이용해 인접한 셀을 확인하기 위한 수평 경계를 결정하고, 보드 한계 내에 있는지 확인

findall/3, between/3을 통해  
X,Y의 값에 따라 주변의 칸의 영역이 달라짐을 상기하여, 주변 칸의 범위를 설정해주고, between/3을 통해 해당 셀 주변의 영역을 설정하고, 주변 셀 중 폭탄이 존재하는 좌표를 Bombs에 저장

length(Bombs, N): 찾아낸 폭탄의 수를 세고 이 수를 N으로 반환

* + - 1. open\_at(Game, X, Y, Status) : 해당 좌표의 셀을 열었을 때 셀 상태를 결정
         1. 셀에 폭탄이 있으면 (bomb(Game, X, Y)), Status = ‘bomb’으로 설정
         2. 그렇지 않다면, count/4를 사용하여 주변에 있는 폭탄의 수를 파악

주변에 폭탄이 없으면 (N = 0), Status = ‘empty’

주변에 폭탄이 있으면, Status = 폭탄들의 수 (N)로 설정

* 1. 코드

% 1. open\_at

count(Game, X, Y, N) :-

    size(Game, SizeX, SizeY),

    X1 is max(X-1, 1),

    X2 is min(X+1, SizeX),

    Y1 is max(Y-1, 1),

    Y2 is min(Y+1, SizeY),

    findall((X5,Y5), (between(X1, X2, X5), between(Y1, Y2, Y5), bomb(Game, X5, Y5)), Bombs),

    length(Bombs, N).

open\_at(Game, X, Y, Status) :-

    size(Game, SizeX, SizeY),

    X > 0, X =< SizeX,

    Y > 0, Y =< SizeY,

    (   bomb(Game, X, Y) -> Status = bomb

    ;   count(Game, X, Y, N),

        (   N = 0 -> Status = empty

        ;   Status = N

        )

    ).

1. ?-show\_answer(Game).
   1. 구성
      1. 문법 : show\_answer(Game).
      2. 기능 : 지뢰 및 주변 지뢰 개수를 모두 표시한 지도 출력
      3. 입출력 :
         1. 입력 : 게임 이름 (ex. game)
         2. 출력 :
            1. 화면 출력 : NxM 전체 지도
            2. 함수 출력 : 없음 (무조건 true)
         3. 출력 시 format() 함수 사용
            1. Bomb인 경우: format(“\* “)
            2. Empty인 경우: format(“ “)
            3. 주변에 N개의 폭탄이 있는 경우: format(“~d “, N)
            4. 줄 넘김은 nl 사용

?- show\_answer(game).

1 \* \* 2 1 1

1 2 2 2 \* 1

1 1 1

1 1 1 1 1 1

1 3 \* 2 1 1 2 \* 1

\* 3 \* 2 1 \* 2 1 1

1 3 2 2 1 1 2 1 1

1 \* 1 1 \* 1

true .

* 1. 구현 방법
     1. show\_answer/1 구현을 위해 show\_rows, show\_row, print\_status를 새롭게 정의하였고, 이들을 show\_answer/1 구현에 활용하였다.
        1. show\_answer(Game) : 보드 전체 출력
           1. 인자 :

Game: 게임 이름 (ex : game)

* + - * 1. 작동방식 :

게임 보드의 크기를 확인한 후, show\_rows 서술어를 호출하여 게임 보드의 각 행을 출력.

* + - 1. show\_rows(Game, CurrentRow, SizeX, SizeY) : 현재 행부터 모든 행을 출력
         1. 인자

Game: 게임 이름 (ex : game)

CurrentRow: 현재 출력 중인 행.

SizeX, SizeY: 게임 보드의 크기.

* + - * 1. 작동방식 :

현재 행 번호가 게임 보드의 높이 이하인 경우, 해당 행의 셀을 출력하는 show\_row를 호출 후 다음 행으로 넘어감.

모든 행을 출력한 후에는 더 이상 진행하지 않음

* + - 1. show\_row(Game, CurrentCol, Y, SizeX) : 지정된 행의 모든 셀을 출력
         1. 인자

Game: 게임 이름 (ex : game)

CurrentCol: 현재 출력 중인 열.

Y: 출력하고 있는 행 번호.

SizeX: 게임 보드의 가로 크기.

* + - * 1. 작동방식 :

현재 열 번호가 게임 보드의 가로 크기 이하인 경우, 해당 셀의 상태를 open\_at을 사용하여 확인

print\_status를 사용하여 상태에 맞는 출력한 후, 다음 열로 넘어감.

* + - 1. print\_status(Status) : 셀의 상태에 따라 적절한 문자 출력
         1. 인자 :

Status: 셀의 상태.

* + - * 1. 작동방식 :

bomb인 경우, "\* "를 출력 (폭탄)

empty인 경우, " "를 출력 (비어있음)

숫자(N)인 경우, 해당 숫자를 출력 (주변 폭탄의 수 표시)

* 1. 코드

% 2. show\_answer(Game).

show\_answer(Game) :-

    size(Game, SizeX, SizeY),

    show\_rows(Game, 1, SizeX, SizeY),

    true.

show\_rows(Game, CurrentRow, SizeX, SizeY) :-

    CurrentRow =< SizeY,

    show\_row(Game, 1, CurrentRow, SizeX),

    nl,

    NextRow is CurrentRow + 1,

    show\_rows(Game, NextRow, SizeX, SizeY).

show\_rows(\_, CurrentRow, \_, SizeY) :-

    CurrentRow > SizeY.

show\_row(Game, CurrentCol, Y, SizeX) :-

    CurrentCol =< SizeX,

    open\_at(Game, CurrentCol, Y, Status),

    print\_status(Status),

    NextCol is CurrentCol + 1,

    show\_row(Game, NextCol, Y, SizeX).

show\_row(\_, CurrentCol, \_, SizeX) :-

    CurrentCol > SizeX.

print\_status(bomb) :-

    format("\* ").

print\_status(empty) :-

    format("  ").

print\_status(N) :-

    integer(N),

    format("~d ", [N]).

1. ?-is\_survive(Game, Opened).
   1. 구성
      1. 문법 : is\_survive(Game, Opened).
      2. 기능 : 열린 위치들이 지뢰를 제외한 위치만 포함되었는지의 여부
      3. 입출력 :
         1. 입력 :
            1. 게임 이름 (ex. game)
            2. 열린 위치 목록 리스트 (ex. [[1,2],[2,3],[3,4]])
         2. 출력 : 생존여부
            1. 지뢰가 포함되는 경우 false
            2. 포함되지 않는 경우 true

?- is\_survive(game,[]).

true .

?- is\_survive(game,[[3,1]]).

true .

?- is\_survive(game,[[3,1],[4,1]]).

false.

* 1. 구현방법
     1. is\_survive /2 구현을 위해 내장 predicate인 member를 이용
        1. show\_answer(Game) : 보드 전체 출력
           1. 인자 :

Game: 게임 이름 (ex : game)

Opened: 현재까지 열린 셀들의 리스트.

* + - * 1. 작동방식 :

\+ 연산자는 "not"을 의미. 즉, "다음 조건이 거짓이 아닌 경우"를 의미.

member([X, Y], Opened)는 Opened 리스트 안에 [X, Y] 셀이 있는지 확인.

bomb(Game, X, Y)는 게임 상태 Game에서 해당 [X, Y] 위치에 폭탄이 있는지 확인.

이 두 조건을 조합하여, Opened 리스트 내에 폭탄이 있는 셀이 없어야 한다. 즉, Opened리스트 내에 폭탄이 없다면 true를 반환. 폭탄이 존재할 경우에는 false를 반환.

* 1. 코드

% 3. is\_survive(Game, Opened)

is\_survive(Game, Opened) :-

    \+ (member([X, Y], Opened), bomb(Game, X, Y)).

1. ?- show\_status(Game, Opened).
   1. 구성
      1. 문법 : show\_status(Game, Opened).
      2. 기능 : 열린 부분에 대한 게임의 상태를 모두 표시,   
         현재 열린 지점에 대한 정보만 표시
      3. 입출력 :
         1. 입력 :
            1. 게임 이름 (ex. game)
            2. 열린 위치 목록 리스트 (ex. [[1,2],[2,3],[3,4]])
         2. 출력 : 생존여부
            1. 화면 출력: NxM 전체 지도

열린 지점만 출력

이외 지점은 ? 출력

* + - * 1. 함수 출력: 없음 (무조건 true)
      1. 해당 쿼리에서 지뢰의 위치는 Opened 입력으로 들어오지 않는다 가정
      2. show\_answer와 같은 방법으로 출력

?- show\_status(game,[[1,1],[2,1]]).

1 ? ? ? ? 1

1 2 2 2 ? 1

1 1 1

1 1 1 1 1 1

1 3 ? 2 1 1 2 ? ?

? ? ? ? ? ? ? ? ?

? ? ? ? ? ? ? ? ?

? ? ? ? ? ? ? ? ?

true .

* 1. 구현방법
     1. show\_status/2 구현을 위해 open\_adjacent, expand\_opened, show\_rows\_status, show\_row\_status 만들어 사용함
        1. open\_adjacent(Game, X, Y, Opened, UpdatedOpened) :   
           폭탄이 없는 경우 지정된 셀 주변의 셀들을 open
           1. 인자 :

Game: 게임 이름 (ex : game)

X, Y: 현재 셀의 좌표.

Opened: 이미 열린 셀들의 목록.

UpdatedOpened: 새롭게 업데이트된 열린 셀들의 목록.

* + - * 1. 작동방식 :

주어진 셀의 상태를 확인하고, 만약 그 셀이 빈 셀(empty)이면, 주변 셀들을 연다.

이웃 셀들 중 아직 열리지 않은 셀들을 찾아내고(\+ member([NX, NY], Opened)), 폭탄이 아닌 셀들만 추가로 연다(Status2 \= bomb).

expand\_opened를 호출하여, 새롭게 열린 셀들을 포함한 업데이트된 목록을 생성

* + - 1. expand\_opened(Game, [[X, Y] | Rest], Opened, UpdatedOpened):   
         주어진 셀 목록을 기반으로 추가적인 셀들을 open
         1. 인자 :

Game: 게임 이름 (ex : game)

[[X, Y] | Rest]: 처리할 셀 목록.

Opened: 이미 열린 셀들의 목록.

UpdatedOpened: 업데이트된 열린 셀들의 목록.

* + - * 1. 작동방식 :

각 셀에 대해, 이미 열린 셀인지 확인하고(member([X, Y], Opened)), 아직 열리지 않은 경우 주변 셀들을 열기(open\_adjacent).

이 과정을 모든 셀에 대해 반복

expand\_opened를 호출하여, 새롭게 열린 셀들을 포함한 업데이트된 목록을 생성

* + - 1. show\_status(Game, Opened) :   
         현재까지 열린 셀들을 기반으로 게임 보드의 상태를 출력
         1. 인자 :

Game: 게임 이름 (ex : game)

Opened: 열린 셀들의 목록.

* + - * 1. 작동방식 :

expand\_opened를 사용하여 추가적으로 열 수 있는 셀들을 모두 연다.

이 과정을 모든 셀에 대해 반복

show\_rows\_status를 호출하여 현재 게임 보드의 상태를 출력

* + - 1. show\_rows\_status(Game, CurrentRow, SizeX, SizeY, UpdatedOpened) :   
         게임 보드의 각 행을 출력
         1. 인자 :

Game: 게임 이름 (ex : game)

CurrentRow: 현재 행.

SizeX, SizeY: 게임 보드의 크기.

UpdatedOpened: 업데이트된 열린 셀들의 목록.

* + - * 1. 작동방식 :

각 행에 대해 show\_row\_status를 호출하여 셀의 상태를 출력

열린 셀들은 해당 상태를, 아직 열리지 않은 셀들은 ?로 표시

* + - 1. show\_row\_status(Game, CurrentCol, Y, SizeX, UpdatedOpened) :   
         주어진 행의 각 셀을 출력
         1. 인자 :

Game: 게임 이름 (ex : game)

CurrentCol: 현재 열.

Y: 현재 행.

SizeX: 게임 보드의 가로 크기.

UpdatedOpened: 업데이트된 열린 셀들의 목록

* + - * 1. 작동방식 :

각 열에 대해, 셀이 열려있는지 확인 후(member([CurrentCol, Y], UpdatedOpened)), 열린 경우 해당 상태를 open\_at과 print\_status를 사용하여 출력

아직 열리지 않은 셀은 ?로 표시

* 1. 코드

% 4. show\_status(Game, Opened).

open\_adjacent(Game, X, Y, Opened, UpdatedOpened) :-

    size(Game, SizeX, SizeY),

    X1 is max(X-1, 1),

    X2 is min(X+1, SizeX),

    Y1 is max(Y-1, 1),

    Y2 is min(Y+1, SizeY),

    open\_at(Game, X, Y, Status),

    (   Status = empty

    ->  findall([NX, NY],

                (   between(X1, X2, NX),

                    between(Y1, Y2, NY),

                    \+ member([NX, NY], Opened),

                    open\_at(Game, NX, NY, Status2),

                    Status2 \= bomb

                ),

                Adjacent),

        expand\_opened(Game, Adjacent, [[X, Y] | Opened], UpdatedOpened)

    ;   UpdatedOpened = [[X, Y] | Opened]

    ).

expand\_opened(\_, [], Opened, Opened).

expand\_opened(Game, [[X, Y] | Rest], Opened, UpdatedOpened) :-

    (   member([X, Y], Opened)

    ->  expand\_opened(Game, Rest, Opened, UpdatedOpened)

    ;   open\_adjacent(Game, X, Y, Opened, TempOpened),

        expand\_opened(Game, Rest, TempOpened, UpdatedOpened)

    ).

show\_status(Game, Opened) :-

    size(Game, SizeX, SizeY),

    expand\_opened(Game, Opened, [], UpdatedOpened),

    show\_rows\_status(Game, 1, SizeX, SizeY, UpdatedOpened).

show\_rows\_status(Game, CurrentRow, SizeX, SizeY, UpdatedOpened) :-

    CurrentRow =< SizeY,

    show\_row\_status(Game, 1, CurrentRow, SizeX, UpdatedOpened),

    nl,

    NextRow is CurrentRow + 1,

    show\_rows\_status(Game, NextRow, SizeX, SizeY, UpdatedOpened).

show\_rows\_status(\_, CurrentRow, \_, SizeY, \_) :-

    CurrentRow > SizeY.

show\_row\_status(Game, CurrentCol, Y, SizeX, UpdatedOpened) :-

    CurrentCol =< SizeX,

    (   member([CurrentCol, Y], UpdatedOpened)

    -> open\_at(Game, CurrentCol, Y, Status),

     print\_status(Status)

    ;   format("? ")

    ),

    NextCol is CurrentCol + 1,

    show\_row\_status(Game, NextCol, Y, SizeX, UpdatedOpened).

show\_row\_status(\_, CurrentCol, \_, SizeX, \_) :-

    CurrentCol > SizeX.

1. ?-is\_win(Game, Opened).
   1. 구성
      1. 문법 : is\_win(Game, Opened).
      2. 기능 : 게임의 승리 여부 확인  
         지뢰가 아닌 모든 지점이 열려 있는지 확인
      3. 입출력 :
         1. 입력 :
            1. 게임 이름 (ex. game)
            2. 열린 위치 목록 리스트 (ex. [[1,2],[2,3],[3,4]])
         2. 출력 : 승리여부
            1. 승리 시 true
            2. 승리가 아닌 경우 false  
               (패배 포함)

?- is\_win(game,[]).

false.

?- is\_win(game,[[3,1]]).

false.

?- is\_win(game,[[3,1],[4,1]]).

false.

?- is\_win(game,[[3,1],[4,1],…]).

false. #[4,1]이 폭탄

?- is\_win(game,[[3,1],[3,3],…]).

true .

* 1. 구현방법
     1. Is\_win/2 구현을 위해 fopened, find\_adjacent\_to\_open, can\_be\_opened, neighbor 만들어 사용함
        1. fopened(Game, Opened, OpenedCells) :   
           expand\_opened를 사용하여, 게임에서 현재까지 안전하게 열린 모든 셀들을 계산
           1. 인자 :

Game: 게임 이름 (ex : game)

Opened: 이미 열린 셀들의 목록.

OpenedCells: 결과적으로 열린 모든 셀들의 목록.

* + - * 1. 작동방식 :

이전에 만들어 둔 expand\_opened를 호출하여 OpenedCells에 최종적으로 열린 셀들의 좌표 리스트를 저장한다.

* + - 1. find\_adjacent\_to\_open(Game, X, Y, AdjacentCells) :   
         주어진 셀 주변에 안전하게 열 수 있는 셀들을 찾기
         1. 인자 :

Game: 게임 이름 (ex : game)

X, Y: 현재 셀의 좌표.

AdjacentCells: 열 수 있는 인접 셀들의 목록.

* + - * 1. 작동방식 :

neighbor 서술어를 사용하여 인접한 셀들을 찾고, can\_be\_opened를 통해 폭탄이 없는 셀만을 필터링

* + - 1. can\_be\_opened(Game, X, Y) :   
         주어진 셀이 안전하게 열 수 있는지 확인
         1. 인자 :

Game: 게임 이름 (ex : game)

X, Y: 셀의 좌표.

* + - * 1. 작동방식 :

셀이 게임 보드 내에 있고(X > 0, X =< SizeX, Y > 0, Y =< SizeY), 폭탄이 없는 경우(\+ bomb(Game, X, Y))에만 true를 반환

* + - 1. neighbor(X, Y, NX, NY) :   
         주어진 셀의 모든 인접 셀들의 좌표를 계산
         1. 인자 :

X, Y: 현재 셀의 좌표.

NX, NY: 인접 셀의 좌표.

* + - * 1. 작동방식 :

현재 셀 주변의 8개 인접 셀(수평, 수직, 대각선)의 좌표를 계산

* + - 1. is\_win(Game, Opened) :   
         플레이어가 게임에서 승리했는지 확인
         1. 인자 :

Game: 게임 이름 (ex : game)

Opened: 열린 셀들의 목록.

* + - * 1. 작동방식 :

먼저 fopene를 사용하여 현재까지 안전하게 열린 모든 셀들을 계산

게임 보드의 크기와 폭탄의 수를 기반으로 전체 셀 수에서 폭탄 수를 빼서 안전하게 열어야 할 셀의 총 수를 계산

열린 셀들의 수가 이 총 수와 동일하고, 열린 셀 중에 폭탄이 없는 경우(\+ (member([X, Y], OpenedCells), bomb(Game, X, Y)))에 승리로 판단

* 1. 코드

% 5. is\_win

fopened(Game, Opened, OpenedCells) :-

    expand\_opened(Game, Opened, [], OpenedCells).

find\_adjacent\_to\_open(Game, X, Y, AdjacentCells) :-

    findall([NX, NY],

            (neighbor(X, Y, NX, NY), can\_be\_opened(Game, NX, NY)),

            AdjacentCells).

can\_be\_opened(Game, X, Y) :-

    size(Game, SizeX, SizeY),

    X > 0, X =< SizeX, Y > 0, Y =< SizeY,

    \+ bomb(Game, X, Y).

neighbor(X, Y, NX, NY) :-

    NX is X + 1, NY is Y; NX is X - 1, NY is Y;

    NX is X, NY is Y + 1; NX is X, NY is Y - 1;

    NX is X + 1, NY is Y + 1; NX is X - 1, NY is Y - 1;

    NX is X + 1, NY is Y - 1; NX is X - 1, NY is Y + 1.

is\_win(Game, Opened) :-

    fopened(Game, Opened, OpenedCells),

    size(Game, SizeX, SizeY),

    findall((X5,Y5), (between(1, SizeX, X5), between(1, SizeY, Y5), bomb(Game, X5, Y5)), Bombs),

    length(Bombs, Nbombs),

    Size is SizeX \* SizeY - Nbombs,

    length(OpenedCells, Size),

    \+ (member([X, Y], OpenedCells), bomb(Game, X, Y)).

3. 전체 코드 (“2019134006.pl”)

% facts

% size(game,9,9).

% bomb(game,4,1).

% bomb(game,5,1).

% bomb(game,7,2).

% bomb(game,3,6).

% bomb(game,8,6).

% bomb(game,1,7).

% bomb(game,3,7).

% bomb(game,6,7).

% bomb(game,3,9).

% bomb(game,8,9).

% size(game,4,4).

% bomb(game,1,1).

% bomb(game,1,2).

% bomb(game,1,3).

% bomb(game,1,4).

% bomb(game,2,4).

% bomb(game,3,4).

% bomb(game,4,4).

% bomb(game,4,3).

% bomb(game,4,2).

% bomb(game,4,1).

% size(game,7,9).

% bomb(game,1,3).

% bomb(game,2,6).

% bomb(game,7,2).

% bomb(game,4,7).

% bomb(game,7,1).

% bomb(game,2,9).

% bomb(game,1,1).

% bomb(game,7,9).

% bomb(game,3,2).

% bomb(game,1,9).

% size(game,13,11).

% bomb(game,1,1).

% bomb(game,4,1).

% bomb(game,7,2).

% bomb(game,13,2).

% bomb(game,9,3).

% bomb(game,1,4).

% bomb(game,5,4).

% bomb(game,10,4).

% bomb(game,1,5).

% bomb(game,7,6).

% bomb(game,1,7).

% bomb(game,3,9).

% bomb(game,8,9).

% bomb(game,10,9).

% bomb(game,13,10).

% rules

% 1. open\_at

count(Game, X, Y, N) :-

    size(Game, SizeX, SizeY),

    X1 is max(X-1, 1),

    X2 is min(X+1, SizeX),

    Y1 is max(Y-1, 1),

    Y2 is min(Y+1, SizeY),

    findall((X5,Y5), (between(X1, X2, X5), between(Y1, Y2, Y5), bomb(Game, X5, Y5)), Bombs),

    length(Bombs, N).

open\_at(Game, X, Y, Status) :-

    size(Game, SizeX, SizeY),

    X > 0, X =< SizeX,

    Y > 0, Y =< SizeY,

    (   bomb(Game, X, Y) -> Status = bomb

    ;   count(Game, X, Y, N),

        (   N = 0 -> Status = empty

        ;   Status = N

        )

    ).

% 2. show\_answer(Game).

show\_answer(Game) :-

    size(Game, SizeX, SizeY),

    show\_rows(Game, 1, SizeX, SizeY),

    true.

show\_rows(Game, CurrentRow, SizeX, SizeY) :-

    CurrentRow =< SizeY,

    show\_row(Game, 1, CurrentRow, SizeX),

    nl,

    NextRow is CurrentRow + 1,

    show\_rows(Game, NextRow, SizeX, SizeY).

show\_rows(\_, CurrentRow, \_, SizeY) :-

    CurrentRow > SizeY.

show\_row(Game, CurrentCol, Y, SizeX) :-

    CurrentCol =< SizeX,

    open\_at(Game, CurrentCol, Y, Status),

    print\_status(Status),

    NextCol is CurrentCol + 1,

    show\_row(Game, NextCol, Y, SizeX).

show\_row(\_, CurrentCol, \_, SizeX) :-

    CurrentCol > SizeX.

print\_status(bomb) :-

    format("\* ").

print\_status(empty) :-

    format("  ").

print\_status(N) :-

    integer(N),

    format("~d ", [N]).

% 3. is\_survive(Game, Opened)

is\_survive(Game, Opened) :-

    \+ (member([X, Y], Opened), bomb(Game, X, Y)).

% 4. show\_status(Game, Opened).

open\_adjacent(Game, X, Y, Opened, UpdatedOpened) :-

    size(Game, SizeX, SizeY),

    X1 is max(X-1, 1),

    X2 is min(X+1, SizeX),

    Y1 is max(Y-1, 1),

    Y2 is min(Y+1, SizeY),

    open\_at(Game, X, Y, Status),

    (   Status = empty

    ->  findall([NX, NY],

                (   between(X1, X2, NX),

                    between(Y1, Y2, NY),

                    \+ member([NX, NY], Opened),

                    open\_at(Game, NX, NY, Status2),

                    Status2 \= bomb

                ),

                Adjacent),

        expand\_opened(Game, Adjacent, [[X, Y] | Opened], UpdatedOpened)

    ;   UpdatedOpened = [[X, Y] | Opened]

    ).

expand\_opened(\_, [], Opened, Opened).

expand\_opened(Game, [[X, Y] | Rest], Opened, UpdatedOpened) :-

    (   member([X, Y], Opened)

    ->  expand\_opened(Game, Rest, Opened, UpdatedOpened)

    ;   open\_adjacent(Game, X, Y, Opened, TempOpened),

        expand\_opened(Game, Rest, TempOpened, UpdatedOpened)

    ).

show\_status(Game, Opened) :-

    size(Game, SizeX, SizeY),

    expand\_opened(Game, Opened, [], UpdatedOpened),

    show\_rows\_status(Game, 1, SizeX, SizeY, UpdatedOpened).

show\_rows\_status(Game, CurrentRow, SizeX, SizeY, UpdatedOpened) :-

    CurrentRow =< SizeY,

    show\_row\_status(Game, 1, CurrentRow, SizeX, UpdatedOpened),

    nl,

    NextRow is CurrentRow + 1,

    show\_rows\_status(Game, NextRow, SizeX, SizeY, UpdatedOpened).

show\_rows\_status(\_, CurrentRow, \_, SizeY, \_) :-

    CurrentRow > SizeY.

show\_row\_status(Game, CurrentCol, Y, SizeX, UpdatedOpened) :-

    CurrentCol =< SizeX,

    (   member([CurrentCol, Y], UpdatedOpened)

    -> open\_at(Game, CurrentCol, Y, Status),

     print\_status(Status)

    ;   format("? ")

    ),

    NextCol is CurrentCol + 1,

    show\_row\_status(Game, NextCol, Y, SizeX, UpdatedOpened).

show\_row\_status(\_, CurrentCol, \_, SizeX, \_) :-

    CurrentCol > SizeX.

% 5. is\_win

fopened(Game, Opened, OpenedCells) :-

    expand\_opened(Game, Opened, [], OpenedCells).

find\_adjacent\_to\_open(Game, X, Y, AdjacentCells) :-

    findall([NX, NY],

            (neighbor(X, Y, NX, NY), can\_be\_opened(Game, NX, NY)),

            AdjacentCells).

can\_be\_opened(Game, X, Y) :-

    size(Game, SizeX, SizeY),

    X > 0, X =< SizeX, Y > 0, Y =< SizeY,

    \+ bomb(Game, X, Y).

neighbor(X, Y, NX, NY) :-

    NX is X + 1, NY is Y; NX is X - 1, NY is Y;

    NX is X, NY is Y + 1; NX is X, NY is Y - 1;

    NX is X + 1, NY is Y + 1; NX is X - 1, NY is Y - 1;

    NX is X + 1, NY is Y - 1; NX is X - 1, NY is Y + 1.

is\_win(Game, Opened) :-

    fopened(Game, Opened, OpenedCells),

    size(Game, SizeX, SizeY),

    findall((X5,Y5), (between(1, SizeX, X5), between(1, SizeY, Y5), bomb(Game, X5, Y5)), Bombs),

    length(Bombs, Nbombs),

    Size is SizeX \* SizeY - Nbombs,

    length(OpenedCells, Size),

    \+ (member([X, Y], OpenedCells), bomb(Game, X, Y)).

1. 실험  
   작성한 코드가 잘 동작하는지 확인하는 작업을 수행하였다.
   1. 예시 1
      1. Facts 입력

size(game,9,9).

bomb(game,4,1).

bomb(game,5,1).

bomb(game,7,2).

bomb(game,3,6).

bomb(game,8,6).

bomb(game,1,7).

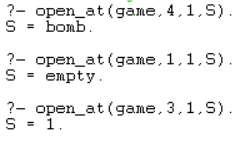
bomb(game,3,7).

bomb(game,6,7).

bomb(game,3,9).

bomb(game,8,9).

* + 1. 실행 결과



텍스트, 폰트, 화이트, 타이포그래피이(가) 표시된 사진

자동 생성된 설명

텍스트, 폰트, 영수증, 화이트이(가) 표시된 사진

자동 생성된 설명

텍스트, 폰트, 화이트, 타이포그래피이(가) 표시된 사진

자동 생성된 설명  
텍스트, 스크린샷, 폰트, 문서이(가) 표시된 사진

자동 생성된 설명  
텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

* 1. 예시 2
     1. Facts 입력

size(game,4,4).

bomb(game,1,1).

bomb(game,1,2).

bomb(game,1,3).

bomb(game,1,4).

bomb(game,2,4).

bomb(game,3,4).

bomb(game,4,4).

bomb(game,4,3).

bomb(game,4,2).

bomb(game,4,1).

* + 1. 실행 결과

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명 텍스트, 폰트, 화이트, 영수증이(가) 표시된 사진

자동 생성된 설명 텍스트, 영수증, 폰트, 화이트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 대수학이(가) 표시된 사진

자동 생성된 설명

* 1. 예시 3
     1. Facts 입력

size(game,7,9).

bomb(game,1,3).

bomb(game,2,6).

bomb(game,7,2).

bomb(game,4,7).

bomb(game,7,1).

bomb(game,2,9).

bomb(game,1,1).

bomb(game,7,9).

bomb(game,3,2).

bomb(game,1,9).

* + 1. 실행 결과

텍스트, 폰트, 스크린샷, 영수증이(가) 표시된 사진

자동 생성된 설명 텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 대수학이(가) 표시된 사진

자동 생성된 설명 텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

* 1. 예시 4 : 앞의 예제들은 강의자료에서 제공된 것. 이것은 직접 임의로 작성한 fact
     1. Facts 입력

size(game,13,11).

bomb(game,1,1).

bomb(game,4,1).

bomb(game,7,2).

bomb(game,13,2).

bomb(game,9,3).

bomb(game,1,4).

bomb(game,5,4).

bomb(game,10,4).

bomb(game,1,5).

bomb(game,7,6).

bomb(game,1,7).

bomb(game,3,9).

bomb(game,8,9). bomb(game,10,9). bomb(game,13,10).

* + 1. 실행 결과

텍스트, 스크린샷, 폰트, 문서이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 흑백, 폰트이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트, 대수학이(가) 표시된 사진

자동 생성된 설명

1. 소감 및 고찰
   1. Prolog는 필자가 평소에 사용하던 언어들이 속한 명령형 언어(Imperative Language)가 아닌 선언형 언어(Declarative language), 특히 논리 프로그래밍 언어에 속하는 프로그래밍 언어이다. Facts, rule, query로 구성되는, 명령형 언어와는 전혀 다른 독특한 시스템에 적응하는 데에 적지 않은 시간이 걸렸으나, 이를 이용해 4색정리 및 지뢰찾기의 기능을 구현해보며 기본 논리만 제시하면 query에 대한 해답을 스스로 출력해낸다는 점에서 prolog만의 강점이 있음을 확인할 수 있었다.
   2. Facts와 rule만 명확히 제시해주면 query에 대한 대답을 스스로 제시하는 Prolog의 특성상, 지뢰찾기와 같이 간단한 논리 규칙 아래에서 이루어지는 복잡하고 재귀적인 논리전개가 요구되는 문제의 해결에 매우 효과적일 것으로 기대된다.