CSF407 AI Assignment 2

Name: Nahush Harihar Kumta
ID NO: 2017A7PS0930G

Problem Statement:

The task is to help an agent navigate through the wumpus world by using propositional logic and making inferences using a Knowledge Base. The wumpus world has 16 squares. One of the squares has a wumpus which if the agent enters, it dies. Similarly there is a pit in one of the squares and again the agent dies if it enters that square. There is exactly one such wumpus and pit in the whole environment and it is given that they won't be there in the starting square [1, 1]. Our task is to help the agent navigate this partially observable world, by determining which squares are safe on the go through an inference algorithm that uses a knowledge base (KB) and reach the goal state of the square [4, 4]. The agent can percieve its location and update its KB with its percepts.

Solution Approach:

The first step was to add all the required rules into the KB.

Some rules were given to us in the problem. These rules were then encoded in the KB at the start before the agent starts navigating the wumpus world. These rules were encoded using propositional logic as explained below:

Symbols used: Pxy, Wxy, Bxy, Sxy imply there is a pit, wumpus, breeze amd stench in square [x, y] respectively.
Totally 64 symbols where used (16 of each kind).
To represent clauses: A space between the symbols involved indicates an 'V' (or) operation.

Rules:
1) There is exactly one wumpus and one pit:
This has two parts to it:
Part1:
There is atleast one pit: 'P11 P21 P31 P41 P12 P22 P32 P42 P13 P23 P33 P43 P14 P24 P34 P44'
Similarly for atleast one wumpus: 'W11 W21 W31 W41 W12 W22 W32 W42 W13 W23 W33 W43 W14 W24 W34 W44'
Part2:
There is atmost one pit: We add rules of the form !Pxy !Px'y' for each pair of squares [x,y] and [x',y'] s.t x != x' and y != y'. Here '!P' indicates 'not of pit'.
There is atmost one wumpus: We add rules of the form !Wxy !Wx'y' for each pair of squares [x,y] and [x',y'] s.t x != x' and y != y'. Here '!W' indicates 'not of wumpus'.

2) There is breeze in squares adjacent to the pit:
This also has two parts to it:
Part1:
There is a breeze implies one of the adjacent squares have a pit:
We add rules of the form '!Bxy Px1y1 Px2y2 ... Pxnyn' where [xi, yi] for i = 1, 2, .. , n are squares adjacent to the agents current location [x,y]. We do this for all [x, y]. Observe that n cannot be greater than 4.
Part2:
There is no breeze implies all adjacent squares have no pits:
We add rules of the form 'Bxy !Px1y1', 'Bxy !Px2y2', ... , 'Bxy !Pxnyn' where [xi, yi] for i = 1, 2, .. , n are squares adjacent to the agents current location [x,y]. We do this for all [x, y]. Again observe that n cannot be greater than 4.

3) We repeat point 2 with the percept of stench instead.
4) The starting square [1,1] is safe:
We add '!P11' and '!W11' to the KB.

Observe that now the KB is basically a CNF where its clauses are the individual strings in the list. All the clauses are considered to be true and any model that results in even one of them being false is automatically deemed as incorrect.

The second step was to make the agent navigate the wumpus world.

The code for this resembles the depth first search code where if the agent is at square [x, y] then it chooses to go to one of its adjacent safe squares. A visited set was maintained that indicates whether a square was visited already. Moreover, a safe set was maintained to keep track of all the sqaures that were already inferred as safe once by the DPLL algorithm. This was done inorder to prevent uneccessary extra calls to the DPLL algorithm.
The agent navigates the world and as it navigates and percieves new things it updates its KB immediately with these percepts. If the agent is not able to conclusively decide the safeness of any of its adjacent squares, it returns to its previous square and tries other paths. If it still can't conclude, it returns back to the starting square [1,1], resets its visited array and restarts with its updated KB. As the agent gains more knowledge (in the form of an KB updated with new percepts) of the environment, it will eventually be able to determine whether a particular square is safe or not and reach the goal state.

The third step was to build an inferencing algorithm that used the KB to make inferences.

The inference algorithm used was the standard DPLL algorithm which checks all possible models of the CNF sentence determined by the KB to see whether it is satisfiable. So if we want to check whether a square [x,y] is safe for the agent to enter, we check for whether the clause !Pxy ^ !Wxy is true. Consequently, this means KB ^ (Pxy V Wxy) is not satisfiable and the DPLL algorithm should return false for the list KB + ['Pxy Wxy']. This was the idea used.

The DPLL algorithm uses three heuristics namely Early Termination, Pure Symbols and Unit Clauses.

Early Termination: This makes use of the fact that clauses are disjunct of literals. This means that all of the literals have to be assigned to false for the clause to be false and just one literal needs to be true for the clause to be true. So as soon as we make a literal true, we make the clause to be true. Adding onto this, all clauses in the KB need to be true if it is to be satisfiable. This means that even if one clause is false, we can deem that model to be false and prune the search tree considerably and make the search faster by avoiding such models of assignments to symbols.

Pure Symbol: A symbol is pure if it appears as the same type of literal (either its negative or postive but not both) among all the clauses. In this case we can assign the symbol such that the literal is true (false in case of a negation and true in case of positive).

Unit Clause: A symbol is a unit symbol if it is the only unassigned symbol in a clause. This heuristic proves to be very useful in this example because we update the KB with unit clauses like Bxy or Sxy on observing a breeze or a stench respectively. Also, with the presence of many two literal clauses in the KB, assigning one of the literals in these clauses ends up making it a unit clause. So the unit clause heuristics proves to be very useful in making the DPLL algorithm terminate quickly in this example.

The table for the effectiveness of each heuristic has been shown below. This was tabulated by removing one heuristic among the three used and seeing the number of calls made to the DPLL function.

The heuristic removed	The number of totals calls made to DPLL function throughout the whole navigation of the environment (including recursive calls)
Early Termination	953
Pure Symbol	953
Unit Clause	Did not terminate

Note: This is for the example in the problem statement.

Conclusion:

The algorithm helped implement a logical agent that can make inferences for its own.