Department of Computer Science
Technical University of Cluj-Napoca

Artificial Intelligence

Laboratory activity

Name: Risa Alexandru
Group: 30231
Email: lexriswow@gmail.com

Teaching Assistant: Adrian Groza
Adrian.Groza@cs.utcluj.ro

Contents

Table 1: Lab scheduling

Activity	**Deadline**
Searching agents, Linux, Latex, Python, Pacman	W_1
Uninformed search	W_2
Informed Search	W_3
Adversarial search	W_4
Propositional logic	W_5
First order logic	W_6
Inference in first order logic	W_7
Knowledge representation in first order logic	W_8
Classical planning	W_9
Contingent, conformant and probabilistic planning	W_{10}
Multi-agent planing	W_{11}
Modelling planning domains	W_{12}
Planning with event calculus	W_{14}

Lab organisation.

1. Laboratory work is 25% from the final grade.
2. There are three deliverables in total: 1. Search, 2. Logic, 3. Planning.
3. Before each deadline, you have to send your work (latex documentation/code) at moodle.cs.utcluj.ro
4. We use Linux and Latex
5. Plagiarism: Don't be a cheater! Cheating affects your colleagues, scholarships and a lot more.

Chapter 1

A1: Search

1. Breadth first search(BFS)
2. Depth first search(DFS)
3. Uniform-cost search(UCS)
4. Greedy best first search(GreedyBFS)
5. A* search(A*)
6. Manhattan heuristic
7. Euclidean heuristic

1.0.1 Breadth first search(BFS)

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key'), and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level.

```
procedure BFS(G, root) is
    let Q be a queue
    label root as discovered
    Q.enqueue(root)

    while Q is not empty do
        v := Q.dequeue()
        if v is the goal then
            return v
        for all edges from v to w in G.adjacentEdges(v) do
             if w is not labeled as discovered then
                 label w as discovered
                 Q.enqueue(w)
end BFS
```

1.0.2 Depth first search(DFS)

Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

```
procedure DFS(G, v) is
    let S be a stack
    S.push(v)

    while S is not empty do
        v = S.pop()
        if v is not labeled as discovered then
            label v as discovered
            for all edges from v to w in G.adjacentEdges(v) do
                S.push(w)
end DFS
```

1.0.3 Uniform-cost search(UCS)

Uniform-Cost Search is a variant of Dijikstra's algorithm. Here, instead of inserting all vertices into a priority queue, we insert only source, then one by one insert when needed. In every step, we check if the item is already in priority queue (using visited array). If yes, we perform decrease key, else we insert it.

```
procedure UCS(problem) is
    if problem's initial state is a goal then
        return empty path
    frontier ← a priority queue ordered by the cost, with a starting node
    reached ← a table of {state: the best path that reached state}
    solution ← failure

    while frontier not empty and top(frontier) cheaper than solution do
        parent ← pop(frontier)
        for child in successors(parent) do
            s ← child.state
            if s not in reached or child cheaper path than reached[s] then
                reached[s] ← child
                add child to the frontier
                if child is a goal and is cheaper than solution then
                    solution = child
    return solution
end UCS
```

1.0.4 Greedy best first search(GBFS)

Best-first search is a search algorithm which explores a graph by expanding the most promising node chosen according to a specified rule(heuristic).

```
procedure: GreedyBFS
    Insert (state=initial_state, h=initial_heuristic, counter=0) into search_queue;
    While search_queue not empty do
        current_queue_entry = pop item from front of search_queue;
        current_entry = state from current_queue_entry;
        current_heuristic = heuristic from current_queue_entry;
        applicable_actions = array of actions applicable in current_state;
        for all index ?i in applicable_actions >= starting_counter do
```

```
            current_action = applicable_actions[?i];
            successor_state = current_state.apply(current_action);
            if successor state is the goal then
                return plan and exit;
            end if
            successor_heuristic = heuristic value of successor_state;
            if successor_heuristic < current_heuristic then
                insert(current_state, current_heuristic, ?i+1) at front of search_queue;
                insert(successor_state, successor_heuristic, 0) at front of search_queue
                break for;
            else
                insert (successor_state, successor_heuristic, 0) into search_queue;
            end if
        end for
    end while
exit { no plan found
```

1.0.5 A* search(A*)

A* (pronounced "A-star") is a graph traversal and path search algorithm, which is often used in many fields of computer science due to its completeness, optimality, and optimal efficiency. One major practical drawback is its $O(b^d)$ space complexity, as it stores all generated nodes in memory. Thus, in practical travel-routing systems, it is generally outperformed by algorithms which can pre-process the graph to attain better performance, as well as memory-bounded approaches; however, A* is still the best solution in many cases.

```
function reconstruct_path(cameFrom, current)
    total_path := {current}
    while current in cameFrom.Keys:
        current := cameFrom[current]
        total_path.prepend(current)
    return total_path

procedure A_Star(start, goal, h)
    openSet := {start}
    cameFrom := an empty map
    gScore := map with default value of Infinity
    gScore[start] := 0
    fScore := map with default value of Infinity
    fScore[start] := h(start)

    while openSet is not empty
        current := the node in openSet having the lowest fScore[] value
        if current = goal
            return reconstruct_path(cameFrom, current)
        openSet.Remove(current)
        for each neighbor of current
            tentative_gScore := gScore[current] + d(current, neighbor)
            if tentative_gScore < gScore[neighbor]
                cameFrom[neighbor] := current
                gScore[neighbor] := tentative_gScore
                fScore[neighbor] := gScore[neighbor] + h(neighbor)
                if neighbor not in openSet
                    openSet.add(neighbor)
    return failure
```

1.0.6 Manhattan heuristic

The standard heuristic for a square grid is the Manhattan distance. Look at your cost function and find the minimum cost D for moving from one space to an adjacent space. In the simple case, you can set D to be 1. The heuristic on a square grid where you can move in 4 directions should be D times the Manhattan distance:

```
procedure heuristic(node)
    dx = abs(node.x - goal.x)
    dy = abs(node.y - goal.y)
    return D * (dx + dy)
```

1.0.7 Euclidean heuristic

If your units can move at any angle (instead of grid directions), then you should probably use a straight line distance:

```
procedure heuristic(node)
    dx = abs(node.x - goal.x)
    dy = abs(node.y - goal.y)
    return D * sqrt(dx * dx + dy * dy)
```

1.0.8 Metrics

The aim of this project is to observe the difference in performance of the studied algorithms, given the three types of problems that have been approached within the Pacman framework so far: PositionSearchProblem(PSP), CornersProblem(CP) and FoodSearchProblem(FSP).

All algorithms solve the classical search PositionSearchProblem very quickly. Though it is observed that DFS does as good of a solution as the other algorithms in the tests I've run, it is known that the implementations of the BFS and DFS algorithms do not guarantee an optimal solution. Also one should remark the advantages of using the A* search since despite the lower number of nodes expanded it still reaches the best solution found by the algorithms in the given tests.

It is fairly easy to observe that whenever a quick, not necessarily optimal solution is required for the CornersProblem, the DFS algorithm performs best, since it requires the least amount of nodes to be expanded. On the other hand, the BFS and UCS algorithms reach terribly high computation times, which makes them unsuitable for applications where a quick reliable result is needed. Once again the A* algorithm achieves the best performance by far, showing just how much of a difference a decent heuristic can make if we were to look at the UCS algorithm.

When it comes to the FoodSearchProblem, even the A* with the default Manhattan heuristic struggles, not being able to solve the bigSearch case given about 3-4h of running time.

A note about the performance of the extra algorithm I chose to implement would be that it appears to perform better then some of its counterparts. At the same time, its greedy nature makes it that the optimality of the solution is very problem-dependent, thus making it unsuitable for certain applications that involve volatile problems.

Informed search algorithm metrics						
Algorithm	Problem	Maze	Cost	Time(s)	Nodes expanded	Score
DFS	PSP	tinyMaze	10	0.0	16	500
DFS	PSP	mediumMaze	130	0.0	147	380
DFS	PSP	bigMaze	210	0.0	391	300
BFS	PSP	tinyMaze	8	0.0	17	502
BFS	PSP	mediumMaze	68	0.0	276	442
BFS	PSP	bigMaze	210	0.0	621	300
GBFSmhtn	PSP	tinyMaze	10	0.0	11	500
GBFSmhtn	PSP	mediumMaze	152	0.0	158	358
GBFSmhtn	PSP	bigMaze	210	0.0	454	300
UCS	PSP	tinyMaze	8	0.0	17	502
UCS	PSP	mediumMaze	68	0.0	275	442
UCS	PSP	bigMaze	210	0.0	620	300
A*mhtn	PSP	tinyMaze	8	0.0	15	502
A*mhtn	PSP	mediumMaze	68	0.0	222	442
A*mhtn	PSP	bigMaze	210	0.0	539	300
A*eucl	PSP	tinyMaze	8	0.0	15	502
A*eucl	PSP	mediumMaze	68	0.0	230	442
A*eucl	PSP	bigMaze	210	0.0	554	300
DFS	CP	tinyCorners	47	0.0	52	493
DFS	CP	mediumCorners	221	0.0	379	319
BFS	CP	tinyCorners	28	0.3	5350	512
BFS	CP	mediumCorners	106	550.8	2775746	434
UCS	CP	tinyCorners	28	0.2	5299	512
UCS	CP	mediumCorners	106	405.3	2835519	434
GBFScrns	CP	tinyCorners	28	0.2	227	512
GBFScrns	CP	mediumCorners	106	134.1	5902	434
A*	CP	tinyCorners	28	0.2	227	512
A*	CP	mediumCorners	106	133.4	5902	434
A*food	FSP	testSearch	7	0.0	11	513
A*food	FSP	trickySearch	60	25.5	4738	570
A*food	FSP	bigSearch	????	?????	???????	???

Chapter 2

A2: Logics

2.0.1 The Hardest Logic Puzzle Ever(Hard)

The Hardest Logic Puzzle Ever is a logic puzzle so called by American philosopher and logician George Boolos and published in The Harvard Review of Philosophy in 1996. Boolos' article includes multiple ways of solving the problem. A translation in Italian was published earlier in the newspaper La Repubblica, under the title L'indovinello più difficile del mondo.

It is stated as follows:

Three gods A, B, and C are called, in no particular order, True, False, and Random. True always speaks truly, False always speaks falsely, but whether Random speaks truly or falsely is a completely random matter. Your task is to determine the identities of A, B, and C by asking three yes-no questions; each question must be put to exactly one god. The gods understand English, but will answer all questions in their own language, in which the words for yes and no are da and ja, in some order. You do not know which word means which.

It was observed by Roberts (2001) and independently by Rabern and Rabern (2008) that the puzzle's solution can be simplified by using certain counterfactuals. The key to this solution is that, for any yes/no question Q, asking either True or False the question "If I asked you Q, would you say ja?" results in the answer ja if the truthful answer to Q is yes, and the answer da if the truthful answer to Q is no (Rabern and Rabern (2008) call this result the embedded question lemma). The reason this works can be seen by studying the logical form of the expected answer to the question.

The solution below constructs its three questions using the lemma described above:

Q1: Ask god B, "If I asked you 'Is A Random?', would you say ja?". If B answers ja, either B is Random (and is answering randomly), or B is not Random and the answer indicates that A is indeed Random. Either way, C is not Random. If B answers da, either B is Random (and is answering randomly), or B is not Random and the answer indicates that A is not Random. Either way, you know the identity of a god who is not Random.

Q2: Go to the god who was identified as not being Random by the previous question (either A or C), and ask him: "If I asked you 'Are you False?', would you say ja?". Since he is not Random, an answer of da indicates that he is True and an answer of ja indicates that he is False.

Q3: Ask the same god the question: "If I asked you 'Is B Random?', would you say ja?". If the answer is ja, B is Random; if the answer is da, the god you have not yet spoken to is Random. The remaining god can be identified by elimination.

Our objective for this problem is to model the solution of the problem such that given all possible situations, we can determine which god is which for each one using Mace4.

Due to its length, the full solution has only been placed in the Appendix and a mere sample will be analysed here(given the fact they are all conceptually the same):

```
 interpretation( 3, [number = 1,seconds = 0], [
    function(A, [0]),
    function(B, [1]),
    function(C, [2]),
    relation(da, [0]),
    relation(ja, [1]),
    relation(q1_da, [0]),
    relation(q1_ja, [1]),
    relation(q2_da, [0]),
    relation(q2_ja, [1]),
    relation(q3_da, [0]),
    relation(q3_ja, [1]),
    relation(false(_), [0,0,1]),
    relation(isFalse(_), [0,0,1]),
    relation(isRand(_), [0,1,0]),
    relation(isTrue(_), [1,0,0]),
    relation(rand(_), [0,1,0]),
    relation(true(_), [1,0,0])]).
```

As can be seen, our domain size is 3 since we have 3 gods A, B and C: one who tells the truth, one who lies and one who answers randomly. Also the words they use "da" and "ja" share the meanings of "yes" and "no", but we don't know which is which. Mace4 is thus used in order to determine all possible models of the three gods and the two meanings of the words and to check whether, given each model, it can determine which god is which.

The functions false(), true() and rand() represent the true identity of each god, used only to generate the answers to the three questions, but not anything else(since we want to determine the identity of the gods using only the answers received from the three given questions). So given the example above, god A is the one who always tells the truth, god B is the one who answers randomly and god C is the one who always lies.

The functions isFalse(), isTrue() and isRand() represent the identity of each god, as it was determined based solely on the answers received to the three questions. They have been especially added for the ease of debug and evaluation, and thus it can be immediately seen that the identities of the gods as they were generated by Mace4 correspond to the identities determined by the solution.

Last but not least, each of the three questions is modelled using 2 variables representing the answer received, which can be either "da" or "ja". Thus it can be observed that the first question is answered with a "ja", the second also with a "ja" and the third with a "ja" as well.

2.0.2 Delightful Dresses Zebra Puzzle(Medium)

Five women are side by side talking about the dresses they are wearing, which they bought recently. Each woman got discounts ranging from 5% to 25% and chose a different type of dress. Which dress is the electrician wearing?

1. The Programmer is exactly to the right of the Surgeon.

2. The woman wearing the Black dress is somewhere to the right of the woman who got the 10% discount.
3. The Actress is next to the woman wearing the Black dress.
4. The Psychologist is next to the woman wearing the Bodycon dress.
5. Megan is the oldest woman.
6. The lady wearing the White dress is somewhere to the left of the 36-year-old woman.
7. The woman wearing the Sheath dress is next to the woman that got 15% off on her new dress.
8. The woman wearing the Purple dress is somewhere between the youngest woman and the woman wearing the White dress, in that order.
9. The lady who got the smallest discount is exactly to the left of the lady who got the 15% discount.
10. The woman who got the 20% discount is somewhere to the right of the woman wearing the White dress.
11. The Surgeon is exactly to the left of the woman wearing the Sundress.
12. Anna is wearing the White dress.
13. Megan is immediately before the woman who got the 5% discount.
14. The Sheath dress is Red.
15. At one of the ends is the 33-year-old woman.
16. Sara is exactly to the right of the lady wearing the Bodycon dress.
17. Lauren is 33 years old.
18. At one of the ends is the woman wearing the Sheath dress.
19. The woman who got the 1% discount is exactly to the left of the woman who got 5% off.
20. The A-line dress is Purple.

The output solution as given by Mace4:

```
    === Mace4 starting on domain size 5. ===

------ process 4012 exit (max_models) ------
interpretation( 5, [number = 1,seconds = 0], [
    function(Actress, [4]),
    function(Aline, [1]),
    function(Anna, [2]),
    function(Black, [3]),
    function(Bodycon, [2]),
    function(Fifteen, [3]),
    function(Five, [2]),
    function(Fourty, [1]),
    function(Lauren, [4]),
    function(Meghan, [1]),
    function(Programmer, [3]),
    function(Psychologist, [1]),
    function(Purple, [1]),
    function(Red, [4]),
    function(Sara, [3]),
    function(Sheath, [4]),
    function(Sundress, [3]),
    function(Surgeon, [2]),
    function(Ten, [1]),
    function(Thirty, [0]),
    function(Thirtysix, [3]),
    function(Thirtythree, [4]),
    function(Twenty, [4]),
    function(White, [2]),
    relation(left_neighbor(,), [
        0,1,0,0,0,
        0,0,1,0,0,
        0,0,0,1,0,
        0,0,0,0,1,
        0,0,0,0,0]),
    relation(left_neighbor_ish(,), [
        0,0,0,0,0,
        1,0,0,0,0,
        1,1,0,0,0,
        1,1,1,0,0,
        1,1,1,1,0]),
    relation(neighbors(,), [
        0,1,0,0,0,
        1,0,1,0,0,
        0,1,0,1,0,
        0,0,1,0,1,
        0,0,0,1,0]),
    relation(right_neighbor(,), [
        0,0,0,0,0,
        1,0,0,0,0,
        0,1,0,0,0,
        0,0,1,0,0,
        0,0,0,1,0]),
    relation(right_neighbor_ish(,), [
        0,1,1,1,1,
        0,0,1,1,1,
        0,0,0,1,1,
        0,0,0,0,1,
        0,0,0,0,0]),
    function(Blue, [0]),
    function(Electrician, [0]),
    function(Erica, [0]),
    function(Thirtynine, [2]),
        function(TwentyFive, [0]),
        function(Wrap, [0])]).
```

There are 5 women in total, so the domain size is 5. These women bought different dresses as told above and they all have different names and professions. After analyzing all the clues, one attribute from each list (color, name, profession, discount, dress type) is associated with a function. Based o the solution given by Mace4, the electrician is Erica, a 30 year old woman, who wears a blue, wrap dress.

2.0.3 Illegible Letters(Easy)

Encode and prove the following argument. You may assume that the domain of discourse consists of all the letters in the room.

1. All the dated letters in this room are written on blue paper.
2. None of them are in black ink except those that are written in the third person.
3. AI have not filed any of them that I can read.
4. None of them that are written on one sheet are undated.
5. All of them that are not crossed are in black ink. :
6. All of them written by Brown begin with "Dear Sir" :
7. All of them written on blue paper are filed. :
8. None of them written on more than one sheet are crossed.
9. None of them that begin with "Dear Sir" are written in third person.
10. Prove: The letters written by Brown cannot be read.

The output solution as given by Prover9:

```
============================== prooftrans ============================
Prover9 (32) version Dec-2007, Dec 2007.
Process 12112 was started by lexri on Veronica,
Sun Nov 29 17:36:32 2020
The command was "/cygdrive/c/Program Files (x86)/Prover9-Mace4/bin-win32/prover9".
============================== end of head ===========================

============================== end of input ==========================

============================== PROOF =================================

% -------- Comments from original proof --------
% Proof 1 at 0.03 (+ 0.00) seconds.
% Length of proof is 31.
% Level of proof is 7.
% Maximum clause weight is 0.
% Given clauses 0.
```

```
1 (all x (datedLetter(x) -> bluePaper(x))) # label(non_clause).  [assumption].
2 (all x (blackInk(x) <-> thirdPerson(x))) # label(non_clause).  [assumption].
3 (all x (read(x) -> -filed(x))) # label(non_clause).  [assumption].
4 (all x (oneSheet(x) -> datedLetter(x))) # label(non_clause).  [assumption].
5 (all x (-crossed(x) -> blackInk(x))) # label(non_clause).  [assumption].
6 (all x (writtenByBrown(x) -> beginsWithDearSir(x))) # label(non_clause).[assumption].
7 (all x (bluePaper(x) -> filed(x))) # label(non_clause).  [assumption].
8 (all x (oneSheet(x) <-> crossed(x))) # label(non_clause).  [assumption].
9 (all x (beginsWithDearSir(x) -> -thirdPerson(x))) # label(non_clause).  [assumption].
10 writtenByBrown(x) -> -read(x) # label(non_clause) # label(goal).  [goal].
11 -oneSheet(x) | datedLetter(x).  [clausify(4)].
12 -datedLetter(x) | bluePaper(x).  [clausify(1)].
14 -blackInk(x) | thirdPerson(x).  [clausify(2)].
15 crossed(x) | blackInk(x).  [clausify(5)].
16 read(c1).  [deny(10)].
17 -read(x) | -filed(x).  [clausify(3)].
18 writtenByBrown(c1).  [deny(10)].
19 -writtenByBrown(x) | beginsWithDearSir(x).  [clausify(6)].
20 -oneSheet(x) | bluePaper(x).  [resolve(11,b,12,a)].
21 -bluePaper(x) | filed(x).  [clausify(7)].
22 oneSheet(x) | -crossed(x).  [clausify(8)].
24 -oneSheet(x) | filed(x).  [resolve(20,b,21,a)].
25 beginsWithDearSir(c1).  [resolve(18,a,19,a)].
26 -beginsWithDearSir(x) | -thirdPerson(x).  [clausify(9)].
27 filed(x) | -crossed(x).  [resolve(24,a,22,a)].
28 crossed(x) | thirdPerson(x).  [resolve(15,b,14,a)].
29 filed(x) | thirdPerson(x).  [resolve(27,b,28,a)].
30 -filed(c1).  [resolve(16,a,17,a)].
31 thirdPerson(c1).  [resolve(29,a,30,a)].
32 -thirdPerson(c1).  [resolve(25,a,26,a)].
33 $F.  [resolve(31,a,32,a)].

============================== end of proof ==========================
```

2.0.4 The SeaGod and the WarGod(Easy)

Encode and prove the following argument. You may assume that the domain of discourse consists of all the letters in the room.

1. Zeus is the sky and thunder god and his brother, Poseidon, is the god of the sea.
2. Rhea was fathered by Uranus. She is also Zeus' mother.
3. The god of the war, Ares is the son of Zeus.
4. Athena and Ares are siblings.
5. Poseidon's son is Triton.
6. Prove that the god of the sea is the uncle of the god of war.

The output solution as given by Prover9:

```
============================== prooftrans ============================
Prover9 (32) version Dec-2007, Dec 2007.
Process 5280 was started by lexri on Veronica,
Mon Nov 30 13:04:56 2020
The command was "/cygdrive/c/Program Files (x86)/Prover9-Mace4/bin-win32/prover9".
============================== end of head ===========================

============================== end of input ==========================

============================== PROOF =================================

% -------- Comments from original proof --------
% Proof 1 at 0.01 (+ 0.00) seconds.
% Length of proof is 12.
% Level of proof is 4.
% Maximum clause weight is 6.
% Given clauses 12.

1 Father(x,y) | Mother(x,y) -> Parent(x,y) # label(non_clause).  [assumption].
5 Parent(x,y) & Sibling(x,z) | Sibling(z,x) & Parent(z,u) -> Cousin(y,u) # label(non_cla
6 Cousin(Atena,Triton) # label(non_clause) # label(goal).  [goal].
7 -Father(x,y) | Parent(x,y).  [clausify(1)].
10 Father(Zeus,Atena).  [assumption].
17 Sibling(Zeus,Poseidon).  [assumption].
20 -Parent(x,y) | -Sibling(x,z) | Cousin(y,u).  [clausify(5)].
30 -Cousin(Atena,Triton).  [deny(6)].
33 Parent(Zeus,Atena).  [resolve(7,a,10,a)].
38 -Parent(Zeus,x) | Cousin(x,y).  [resolve(20,b,17,a)].
48 Cousin(Atena,x).  [hyper(38,a,33,a)].
49 $F.  [resolve(48,a,30,a)].

============================== end of proof ==========================
```

Clause 31 was obtained from clause 30 by applying demodulation, using clauses 27 and 28. Demodulation is the process of using a set of oriented equations to rewrite terms. The first part, 27(1), tells that the left side of clause 27 (SeaGod) gets replaced by its right side (Poseidon) inside clause 29 and the second part, 28(2) tells that Ares is replaced by WarGod.

Chapter 3

A3: Planning

3.0.1 The All-Out Problem

The All-Out Puzzle from `https://www.mathsisfun.com` implies that we have a 5x5 matrix of coins, with their faces up or down in a given order. The goal within the puzzle is to flip the coins so that all end up facing upward, given the rule that whenever a coin is flipped, all adjacent coins to it are flipped also.

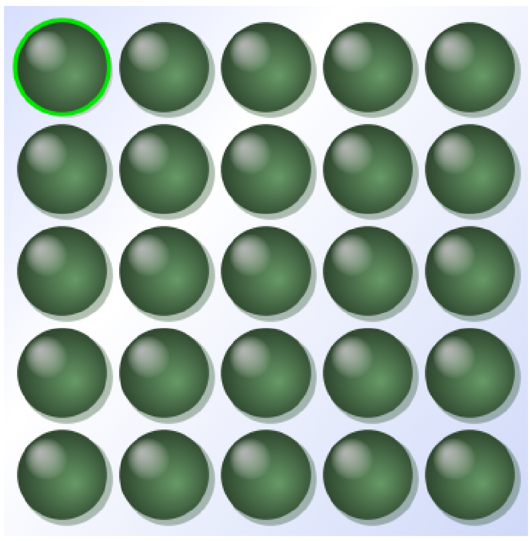


Figure 3.1: The All-Out Game

Thus, using the Planning Domain Definition Language, we have solved the problem the following way:

1. We defined the domain as having predicates for which side of the coin is up, whether the coin has two, three or four neighbours, and one representing the adjacency between coins.
2. Given the fact that there can be two, three or four adjacent coins to the main one we're flipping, we have defined three actions to treat the cases. The parameters for each action are the coins that are going to be influence by that action, so flip5 will have five parameters, flip4 will have four and so on.
3. Moving on to the definition of the problem, we define the matrix of 5x5 coins and we initialize the predicates so that the adjacency and the has-x-neighbours are correct, and the coins are all facing down. As stated above, the goal of the problem is to build a plan that will make all coins face upwards upon execution.

Three requirements were used for the sake of achieving our goals: negative-preconditions, conditional-effects and equality.

1. The negative-preconditions are required so that we can use the logical function not, for the sake of being able to apply preconditions such as "all coins must be different within the action" or "coin x can only exactly have 2 neighbours, not also exactly 3 or exactly 4", and other similar situations.
2. We require the conditional-effects requirement in order to be able to flip the coins without writing perhaps twice as much code. This way we enable ourselves to flip the coin whatever value it has, without having the need to specify it in the precondition and create several actions for each case.
3. Last but not least, the equality requirement is critical in our implementation due to the fact that we need to ensure the uniqueness of the parameters that end up being affected by the actions. We do not want the planner to pick actions where the same parameters is used twice, since that is not how the problem we're trying to solve works.

Thus the solution given by the planner when using the astar heuristic is the following plan of action:

```
(flip5 coin33 coin23 coin32 coin34 coin43)
(flip4 coin41 coin31 coin42 coin51)
(flip5 coin34 coin24 coin33 coin35 coin44)
(flip5 coin23 coin13 coin22 coin24 coin33)
(flip5 coin24 coin14 coin23 coin25 coin34)
(flip4 coin35 coin25 coin34 coin45)
(flip4 coin45 coin35 coin44 coin55)
(flip5 coin44 coin34 coin43 coin45 coin54)
(flip4 coin52 coin42 coin51 coin53)
(flip5 coin42 coin32 coin41 coin43 coin52)
(flip3 coin51 coin41 coin52)
(flip5 coin22 coin12 coin21 coin23 coin32)
(flip4 coin13 coin12 coin14 coin23)
(flip4 coin12 coin11 coin13 coin22)
(flip3 coin15 coin14 coin25)
; cost = 15 (unit cost)
```

So by flipping the first column of coins(coin33, coin41, coin34, ... coin15) we will reach the desired state where all coins have their faces pointing upwards.

3.0.2 The Treasure Problem

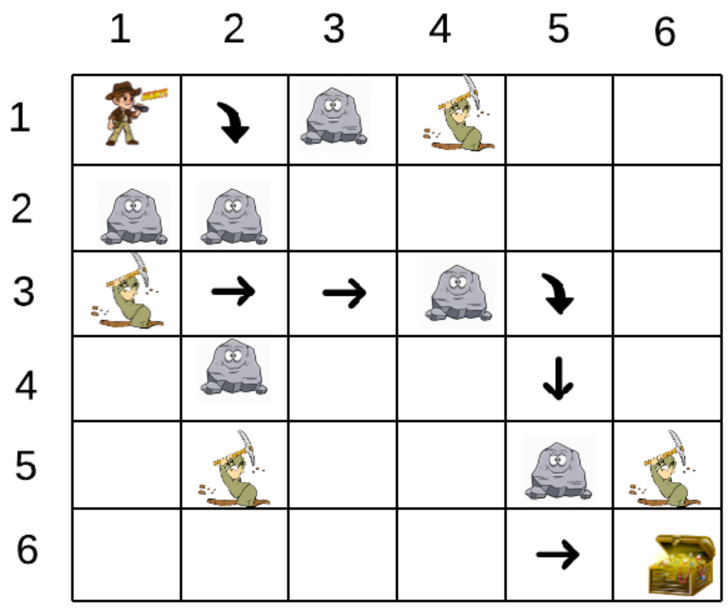
Avem următoarea problemă: un vânător de comoare vrea să găsească aurul și alte pietre prețioase, dar vrea sa sape cât mai puțin posibil. În calea lui se află pietre sau alți vânători. El poate ocoli pietrele, însă nu poate săpa într-un loc ocupat de altcineva. Așadar ocolirea pietrelor este posibilă doar atunci când în spatele acestora nu se află un alt vânător.
Fișierul treasure-domain.pddl conține 7 predicate și 8 acțiuni. Poziția vânătorului nostru este hunter-at, iar is-rock, is-another-hunter semnifică pozițiile pietrelor și a celorlalți vânători. Predicatele next-row, next-column, avoiding-row și avoiding-column reprezintă pozițiile în urma unei mișcări simple, sau a unei ocoliri.

Treasure-domain.pddl:

```
(define (domain treasure)
(:predicates
(hunter-at ?r ?c)
(is-rock ?r ?c)
(is-another-hunter ?r ?c)
(next-row ?r1 ?r2)
(next-column ?c1 ?c2)
(avoiding-row ?r1 ?r2 ?r3)
(avoiding-column ?c1 ?c2 ?c3)
)
(:action dig-up
  :parameters (?old-row ?new-row ?col)
  :precondition (and (next-row ?new-row ?old-row)
     (hunter-at ?old-row ?col)
     (not(is-rock ?new-row ?col))
     (not(is-another-hunter ?new-row ?col))
)
  :effect (and (not (hunter-at ?old-row ?col))
(hunter-at ?new-row ?col)
  )
)

(:action dig-down
  :parameters (?old-row ?new-row ?col)
  :precondition (and (next-row ?old-row ?new-row)
     (hunter-at ?old-row ?col)
     (not (is-rock ?new-row ?col))
     (not (is-another-hunter ?new-row ?col))
)
  :effect (and (not (hunter-at ?old-row ?col))
       (hunter-at ?new-row ?col)
  )
)

(:action dig-left
  :parameters (?row ?old-col ?new-col)
  :precondition (and (next-column ?new-col ?old-col)
     (hunter-at ?row ?old-col)
     (not (is-rock ?row ?new-col))
     (not (is-another-hunter ?row ?new-col))
    )
  :effect (and (not (hunter-at ?row ?old-col))
(hunter-at ?row ?new-col)

  )
)

(:action dig-right
  :parameters (?row ?old-col ?new-col)
  :precondition (and (next-column ?old-col ?new-col)
(hunter-at ?row ?old-col)
(not (is-rock ?row ?new-col))
(not (is-another-hunter ?row ?new-col)))
  :effect (and (not (hunter-at ?row ?old-col))
(hunter-at ?row ?new-col)
  )
)

(:action dig-avoid-up
  :parameters (?old-row ?new-row ?new-rowa ?col)
  :precondition (and (avoiding-row ?new-rowa ?new-row ?old-row)
     (hunter-at ?old-row ?col)
     (is-rock ?new-row ?col)
     (not (is-rock ?new-rowa ?col))
     (not (is-another-hunter ?new-rowa ?col))
)
  :effect (and (not (hunter-at ?old-row ?col))
(hunter-at ?new-rowa ?col)
  )
)

(:action dig-avoid-down
  :parameters (?old-row ?new-row ?new-rowa ?col)
  :precondition (and (not (is-rock ?new-rowa ?col))
     (avoiding-row ?old-row ?new-row ?new-rowa)
     (hunter-at ?old-row ?col)
     (is-rock ?new-row ?col)
     (not (is-another-hunter ?new-rowa ?col))
 )
  :effect (and (not (hunter-at ?old-row ?col))
       (hunter-at ?new-rowa ?col)
  )
)

(:action dig-avoid-left
  :parameters (?row ?old-col ?new-col ?new-cola)
  :precondition (and (avoiding-column ?new-cola ?new-col ?old-col)
     (hunter-at ?row ?old-col)
     (is-rock ?row ?new-col)
     (not (is-rock ?row ?new-cola))
     (not (is-another-hunter ?row ?new-cola))
)
  :effect (and (not (hunter-at ?row ?old-col))
(hunter-at ?row ?new-cola)


  )
)

(:action dig-avoid-right
  :parameters (?row ?old-col ?new-col ?new-cola)
  :precondition (and (avoiding-column ?old-col ?new-col ?new-cola)
(hunter-at ?row ?old-col)
(is-rock ?row ?new-col)
(not (is-rock ?row ?new-cola))
(not (is-another-hunter ?row ?new-cola))
)
  :effect (and (not (hunter-at ?row ?old-col))
(hunter-at ?row ?new-cola)

  )
)

)
```

Urmează acum câteva cazuri implementate.

Cazul 1:



După cum se observă din figură, avem o matrice de 6x6, poziția inițială a vânătorului principal fiind (1,1). Pentru a ajunge la comorile ascunse (poziția (6,6)), el trebuie să parcurgă un drum cât mai scurt respectând regulile menționate anterior, conform cărora, vânătorul de la poziția (3,1) face ca piatra de pe (2,1) să nu se poată ocoli. Personajul nostru este așadar nevoit să efectueze acțiunea ”dig-right”, deplasându-se astfel la dreapta. Apoi ocolește piatra (2,2) și își continuă drumul. În final ajunge la comoară, costul drumului fiind 7.
Acest caz a fost configurat în fițierul treasure1.pddl, unde am stabilit relațiile de next-row ți next-column pentru rânduri și coloane și de asemenea care sunt pozițiile în cazul unei ocoliri (avoiding-row,avoiding-column). Și nu în ultimul rând am marcat unele căsuțe cu o piatră, altele cu vânători. Personajul principal a fost plasat la poziția (1,1,) prin (hunter-at row1 col1), la fel și goal-ul cu (hunter-at row8 col8), în căsuța (8,8) aflîndu-se comoara.

Treasure1.pddl:

```
(define (problem treasure1)
  (:domain treasure)
  (:objects
     row1 row2 row3 row4 row5 row6
     col1 col2 col3 col4 col5 col6 )
  (:init

    (next-row row1 row2)         (next-column col1 col2)
    (next-row row2 row3)         (next-column col2 col3)
    (next-row row3 row4)         (next-column col3 col4)
    (next-row row4 row5)         (next-column col4 col5)
    (next-row row5 row6)         (next-column col5 col6)

    (avoiding-row row1 row2 row3)    (avoiding-column col1 col2 col3)
    (avoiding-row row2 row3 row4)    (avoiding-column col2 col3 col4)
    (avoiding-row row3 row4 row5)    (avoiding-column col3 col4 col5)
    (avoiding-row row4 row5 row6)    (avoiding-column col4 col5 col6)

    (is-rock row1 col3)
    (is-rock row2 col1)
    (is-rock row2 col2)
    (is-rock row3 col4)
    (is-rock row4 col2)
    (is-rock row5 col5)

    (is-another-hunter row1 col4)
    (is-another-hunter row5 col2)
    (is-another-hunter row5 col6)
    (is-another-hunter row3 col1)

    (hunter-at row1 col1)

  )
  (:goal (hunter-at row6 col6)
  )
)
```

Rezultate Treasure1.pddl:

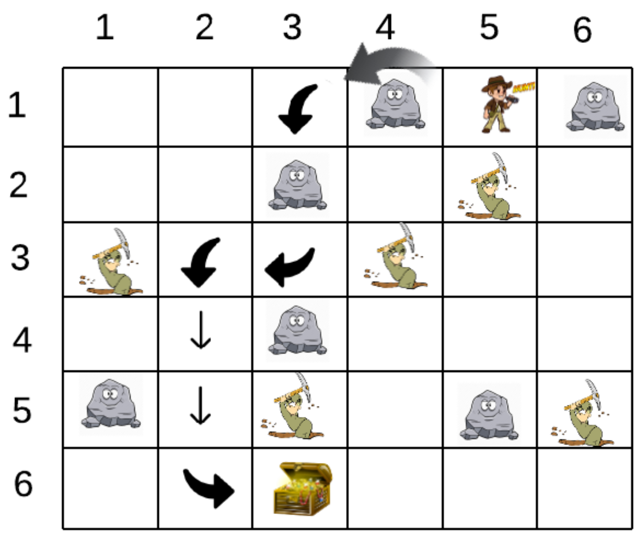
```
(dig-right row1 col1 col2)
(dig-avoid-down row1 row2 row3 col2)
(dig-right row3 col2 col3)
(dig-avoid-right row3 col3 col4 col5)
(dig-down row3 row4 col5)
(dig-avoid-down row4 row5 row6 col5)
(dig-right row6 col5 col6)
```

```
; cost = 7 (unit cost)
```

Tabel cu rezultatele obținute în urma rulării cu diferiți algoritmi și diferite euristici.

Algoritm	Heuristică	Cost	Lungime	Stari expandate	Stari evaluate	Stari generate
Astar	ff()	7	7	8	15	20
Astar	cg()	7	7	10	18	26
Eager-Greedy	ff()	7	7	8	15	20
Eager-Greedy	cg()	7	7	10	18	27

Cazul 2:



În acest caz vânătorul se află pe poziția (1,5) și comoara pe poziția (6,3).

Treasure2.pddl:

```
(define (problem treasure2)
  (:domain treasure)
  (:objects
     row1 row2 row3 row4 row5 row6
     col1 col2 col3 col4 col5 col6 )
  (:init

    (next-row row1 row2)          (next-column col1 col2)
    (next-row row2 row3)         (next-column col2 col3)
    (next-row row3 row4)         (next-column col3 col4)
    (next-row row4 row5)         (next-column col4 col5)
    (next-row row5 row6)         (next-column col5 col6)
    (avoiding-row row1 row2 row3)    (avoiding-column col1 col2 col3)
    (avoiding-row row2 row3 row4)    (avoiding-column col2 col3 col4)
    (avoiding-row row3 row4 row5)    (avoiding-column col3 col4 col5)
    (avoiding-row row4 row5 row6)    (avoiding-column col4 col5 col6)

    (is-rock row1 col6)
    (is-rock row1 col4)
    (is-rock row2 col3)
    (is-rock row4 col3)
    (is-rock row5 col1)
    (is-rock row5 col5)

    (is-another-hunter row2 col5)
    (is-another-hunter row3 col1)
    (is-another-hunter row3 col4)
    (is-another-hunter row5 col3)
    (is-another-hunter row5 col6)

    (hunter-at row1 col5)

  )
  (:goal (hunter-at row6 col3)
  )
)
```

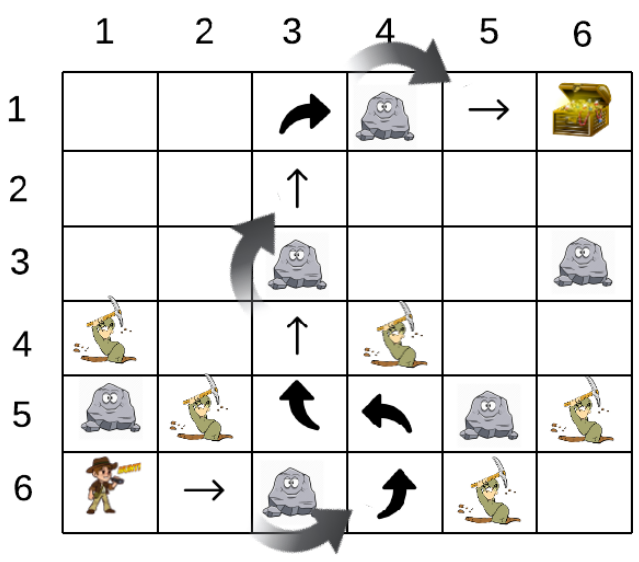
Rezultate Treasure2.pddl:

```
(dig-avoid-left row1 col5 col4 col3)
(dig-avoid-down row1 row2 row3 col3)
(dig-left row3 col3 col2)
(dig-down row3 row4 col2)
(dig-down row4 row5 col2)
(dig-down row5 row6 col2)
(dig-right row6 col2 col3)
; cost = 7 (unit cost)
```

Tabele cu rezultatele obținute în urma rulării cu diferiți algoritmi și diferite euristici.

Algoritm	Heuristică	Cost	Lungime	Stari expandate	Stari evaluate	Stari generate
Astar	ff()	7	7	8	13	18
Astar	cg()	7	7	14	18	33
Eager-Greedy	ff()	7	7	8	13	18
Eager-Greedy	cg()	7	7	14	18	33

Cazul 3:



În acest caz vânătorul se află pe poziția (6,1) și comoara pe poziția (1,6).

Treasure3.pddl:

```
(define (problem treasure3)
  (:domain treasure)
  (:objects
     row1 row2 row3 row4 row5 row6
     col1 col2 col3 col4 col5 col6 )
  (:init

    (next-row row1 row2)          (next-column col1 col2)
    (next-row row2 row3)         (next-column col2 col3)
    (next-row row3 row4)         (next-column col3 col4)
    (next-row row4 row5)         (next-column col4 col5)
    (next-row row5 row6)         (next-column col5 col6)


    (avoiding-row row1 row2 row3)    (avoiding-column col1 col2 col3)
    (avoiding-row row2 row3 row4)    (avoiding-column col2 col3 col4)
    (avoiding-row row3 row4 row5)    (avoiding-column col3 col4 col5)
    (avoiding-row row4 row5 row6)    (avoiding-column col4 col5 col6)


    (is-rock row1 col4)
    (is-rock row3 col3)
    (is-rock row3 col6)
    (is-rock row5 col1)
    (is-rock row5 col5)
    (is-rock row6 col3)


    (is-another-hunter row4 col1)
    (is-another-hunter row4 col4)
    (is-another-hunter row5 col2)
    (is-another-hunter row5 col6)
    (is-another-hunter row6 col5)

    (hunter-at row6 col1)

  )
  (:goal (hunter-at row1 col6)
  )
)
```

Rezultate Treasure3.pddl:

```
(dig-right row6 col1 col2)
(dig-avoid-right row6 col2 col3 col4)
(dig-up row6 row5 col4)
(dig-left row5 col4 col3)
(dig-up row5 row4 col3)
(dig-avoid-up row4 row3 row2 col3)
(dig-up row2 row1 col3)
(dig-avoid-right row1 col3 col4 col5)
(dig-right row1 col5 col6)
; cost = 9 (unit cost)
```

Tabel cu rezultatele obținute în urma rulării cu diferiți algoritmi și diferite euristici.

Algoritm	Heuristică	Cost	Lungime	Stari expandate	Stari evaluate	Stari generate
Astar	ff()	9	9	10	15	22
Astar	cg()	9	9	13	18	31
Eager-Greedy	ff()	9	9	10	15	22
Eager-Greedy	cg()	9	9	13	18	30

Bibliography

1. https://en.wikipedia.org/wiki/Breadth-first_search
2. https://en.wikipedia.org/wiki/Depth-first_search
3. https://github.com/aimacode/aima-pseudocode/blob/master/md/Uniform-Cost-Search.md
4. https://www.coursehero.com/file/p48anp1/II-Pseudo-code-Procedure-GreedyBFS-Insert-s
5. https://en.wikipedia.org/wiki/A*_search_algorithm
6. http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html
7. http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html
8. https://www.brainzilla.com/logic/zebra/delightful-dresses/
9. https://en.wikipedia.org/wiki/The_Hardest_Logic_Puzzle_Ever#:~:text=It%20is%20stated%20as%20follows,is%20a%20completely%20random%20matter.
10. Exercises in Logic and Knowldege Representation - Brandon Bennett
11. https://www.mathsisfun.com/games/allout.html
12. Introduction to Artificial Intelligence - Adrian Groza, Radu Razvan Slavescu and Anca Marginean

Appendix A

Your original code

Don't be a cheater! Cheating affects your colleagues, scholarships and a lot more. This section should contain only code developed by you, without any line re-used from other sources. This section helps me to correctly evaluate your amount of work and results obtained.

A.0.1 Search

```
def randomSearch(problem):
    current_state = problem.getStartState()
    actions_list = []
    from random import randrange

    while not problem.isGoalState(current_state):
        successors = problem.getSuccessors(current_state)
        next_action = successors[randrange(len(successors))]
        current_state = next_action[0]
        actions_list.append(next_action[1])
    return actions_list

def depthFirstSearch(problem):
    current_state = problem.getStartState()
    action_list = []

    visited = []
    state_queue = [current_state]
    action_lists_queue = [[]]

    while state_queue and action_lists_queue
        and not problem.isGoalState(current_state):

        current_state = state_queue.pop()
        action_list = action_lists_queue.pop()
        if current_state not in visited:
            visited.append(current_state)

        for successor in problem.getSuccessors(current_state):
            if successor[0] not in visited:
                state_queue.append(successor[0])
                action_lists_queue.append(action_list + [successor[1]])
    return action_list

def breadthFirstSearch(problem):
    current_state = problem.getStartState()
    action_list = []

    visited = []
    state_queue = [current_state]
    action_lists_queue = [[]]

    while state_queue and action_lists_queue
        and not problem.isGoalState(current_state):

        current_state = state_queue.pop(0)
        action_list = action_lists_queue.pop(0)
        if current_state not in visited:
            visited.append(current_state)

        for successor in problem.getSuccessors(current_state):
            if successor[0] not in visited:
                state_queue.append(successor[0])
                action_lists_queue.append(action_list + [successor[1]])
    return action_list

def uniformCostSearch(problem):
    from heapq import heappush, heappop

    current_state = problem.getStartState()
    action_list = []

    visited = []
    heap = []
    heappush(heap, (0, current_state, action_list))

    while heap and not problem.isGoalState(current_state):

        heap_top = heappop(heap)
        current_state, action_list = (heap_top[i] for i in [1, 2])
        if current_state not in visited:
            visited.append(current_state)

        for successor in problem.getSuccessors(current_state):
            if successor[0] not in visited:
                heappush(heap, (len(action_list) + 1,
                    successor[0], action_list + [successor[1]]))

    return action_list

def greedySearch(problem, heuristic=nullHeuristic):
    from heapq import heappush, heappop
    current_state = problem.getStartState()
    action_list = []

    visited = []
    heap = []
    heappush(heap, (0, current_state, action_list))

    while heap and not problem.isGoalState(current_state):

        heap_top = heappop(heap)
        current_state, action_list = (heap_top[i] for i in [1, 2])
        if current_state not in visited:
            visited.append(current_state)

        for successor in problem.getSuccessors(current_state):
            if successor[0] not in visited:
                heappush(heap, (heuristic(successor[0], problem),
                    successor[0], action_list + [successor[1]]))

    return action_list

def aStarSearch(problem, heuristic=nullHeuristic):
    from heapq import heappush, heappop

    current_state = problem.getStartState()
    action_list = []

    visited = []
    heap = []
    heappush(heap, (0, current_state, action_list))

    while heap and not problem.isGoalState(current_state):

        heap_top = heappop(heap)
        current_state, action_list = (heap_top[i] for i in [1, 2])
        if current_state not in visited:
            visited.append(current_state)

        for successor in problem.getSuccessors(current_state):
            if successor[0] not in visited:
                heappush(heap, (len(action_list) + 1 + heuristic(successor[0], problem),
                    successor[0], action_list + [successor[1]]))

    return action_list
```

A.0.2 Logics

1: The Hardest Logic Puzzle Ever(Hard)

```
% Saved by Prover9-Mace4 Version 0.5, December 2007.

set(ignore_option_dependencies). % GUI handles dependencies

if(Prover9). % Options for Prover9
  assign(max_seconds, 60).
end_if.

if(Mace4).   % Options for Mace4
  assign(domain_size, 3).
  assign(start_size, 3).
  assign(end_size, 3).
  assign(max_models, 70).
  assign(max_seconds, 60).
end_if.

formulas(assumptions).

%Three gods A, B, and C are called, in no particular
%order, True, False, and Random.
%True always speaks truly, False always speaks falsely,
%but whether Random speaks truly or falsely is a
%completely random matter.
%Your task is to determine the identities of A, B, and C
%by asking three yes-no questions; each question must be
%put to exactly one god.
%The gods understand English, but will answer all
%questions in their own language, in which the words for
%yes and no are da and ja, in some order.
%You do not know which word means which.

%doua cuvinte, da si ja care inseamna da si nu(nu se
%stie care-i care)
ja <-> -da.

%trei zei care unul spune adevarul, unul minte si unul
%random
( true(A) & false(B) & rand(C) ) |
( true(A)  & rand(B) & false(C) ) |
( false(A) & true(B) & rand(C)  ) |
( false(A) & rand(B) & true(C) ) |
( rand(A) & true(B) & false(C) ) |
( rand(A) & false(B) & true(C) ).

true(A) <-> ( -true(B) & -true(C) ).
true(B) <-> ( -true(A) & -true(C) ).
true(C) <-> ( -true(A) & -true(B) ).
```

```
false(A) <-> ( -false(B) & -false(C) ).
false(B) <-> ( -false(A) & -false(C) ).
false(C) <-> ( -false(A) & -false(B) ).

rand(A) <-> ( -rand(B) & -rand(C) ).
rand(B) <-> ( -rand(A) & -rand(C) ).
rand(C) <-> ( -rand(A) & -rand(B) ).

all x ( -true(x) & -false(x) -> rand(x) ).
all x ( -true(x) & -rand(x) -> false(x) ).
all x ( -false(x) & -rand(x) -> true(x) ).

%aici avem variabilele care nu stiu(si vor sa afle) care
%este fiecare dintre zei, pe baza celor 3 intrebari care
%alcatuiesc solutia
isTrue(A) <-> ( -isTrue(B) & -isTrue(C) ).
isTrue(B) <-> ( -isTrue(A) & -isTrue(C) ).
isTrue(C) <-> ( -isTrue(A) & -isTrue(B) ).

isFalse(A) <-> ( -isFalse(B) & -isFalse(C) ).
isFalse(B) <-> ( -isFalse(A) & -isFalse(C) ).
isFalse(C) <-> ( -isFalse(A) & -isFalse(B) ).

isRand(A) <-> ( -isRand(B) & -isRand(C) ).
isRand(B) <-> ( -isRand(A) & -isRand(C) ).
isRand(C) <-> ( -isRand(A) & -isRand(B) ).

all x ( -isTrue(x) & -isFalse(x) -> isRand(x) ).
all x ( -isTrue(x) & -isRand(x) -> isFalse(x) ).
all x ( -isFalse(x) & -isRand(x) -> isTrue(x) ).

%ask god B "If I asked you 'Is A Random?', would you say
%ja?"
%results in the answer ja if the truthful answer to q1
%is yes, and the answer da if the truthful answer to Q
%is no
(-rand(B)) -> ( ( (  rand(A) & ja ) | (  rand(A) & da ) ) <-> q1_ja ).
(-rand(B)) -> ( ( ( -rand(A) & ja ) | ( -rand(A) & da ) ) <-> q1_da ).
  rand(B)  -> ( q1_ja | q1_da ).

q1_ja <-> -q1_da.

%use answer from q1 to determine who surely isn't random
q1_ja -> ( isRand(B) | isRand(A) ). %thus god C surely isnt the random
q1_da -> ( isRand(B) | isRand(C) ). %thus god A surely isnt the random

%ask god that we found isnt random "If I asked you 'Are
%you False?', would you say ja?"
%an answer of da indicates that he is True and an answer
```

```
%of ja indicates that he is False
q1_ja -> ( ( ( false(C) & ja ) | ( false(C) & da ) ) <-> q2_ja ).
q1_ja -> ( ( (  true(C) & ja ) | (  true(C) & da ) ) <-> q2_da ).
q1_da -> ( ( ( false(A) & ja ) | ( false(A) & da ) ) <-> q2_ja ).
q1_da -> ( ( (  true(A) & ja ) | (  true(A) & da ) ) <-> q2_da ).

q2_ja <-> -q2_da.

%use answer from g2 to determine who is true or who is
%false
( q1_ja & q2_ja ) -> isFalse(C).
( q1_ja & q2_da ) ->  isTrue(C).
( q1_da & q2_ja ) -> isFalse(A).
( q1_da & q2_da ) ->  isTrue(A).

%ask same god "If I asked you 'Is B Random?', would you
%say ja?"
%If the answer is ja, B is Random; if the answer is da,
%the god you have not yet spoken to is Random
( (  rand(B) & ja ) | (  rand(B) & da ) ) <-> q3_ja.
( ( -rand(B) & ja ) | ( -rand(B) & da ) ) <-> q3_da.

q3_ja <-> -q3_da.

%use answer from q3 to determine who is random
q3_ja -> isRand(B).
q3_da & q1_ja -> isRand(A).
q3_da & q1_da -> isRand(C).

end_of_list.

formulas(goals).

end_of_list.

interpretation( 3, [number = 1,seconds = 0], [
    function(A, [0]),
    function(B, [1]),
    function(C, [2]),
    relation(da, [0]),
    relation(ja, [1]),
    relation(q1_da, [0]),
    relation(q1_ja, [1]),
    relation(q2_da, [0]),
    relation(q2_ja, [1]),
    relation(q3_da, [0]),
    relation(q3_ja, [1]),
    relation(false(_), [0,0,1]),
    relation(isFalse(_), [0,0,1]),
    relation(isRand(_), [0,1,0]),
    relation(isTrue(_), [1,0,0]),
    relation(rand(_), [0,1,0]),
    relation(true(_), [1,0,0])]).
interpretation( 3, [number = 2,seconds = 0], [
    function(A, [0]),
    function(B, [1]),
    function(C, [2]),
    relation(da, [0]),
    relation(ja, [1]),
    relation(q1_da, [0]),
    relation(q1_ja, [1]),
    relation(q2_da, [0]),
    relation(q2_ja, [1]),
    relation(q3_da, [1]),
    relation(q3_ja, [0]),
    relation(false(_), [0,0,1]),
    relation(isFalse(_), [0,0,1]),
    relation(isRand(_), [1,0,0]),
    relation(isTrue(_), [0,1,0]),
    relation(rand(_), [1,0,0]),
    relation(true(_), [0,1,0])]).
interpretation( 3, [number = 3,seconds = 0], [
    function(A, [0]),
    function(B, [1]),
    function(C, [2]),
    relation(da, [0]),
    relation(ja, [1]),
    relation(q1_da, [0]),
    relation(q1_ja, [1]),
    relation(q2_da, [1]),
    relation(q2_ja, [0]),
    relation(q3_da, [0]),
    relation(q3_ja, [1]),
    relation(false(_), [1,0,0]),
    relation(isFalse(_), [1,0,0]),
    relation(isRand(_), [0,1,0]),
    relation(isTrue(_), [0,0,1]),
    relation(rand(_), [0,1,0]),
    relation(true(_), [0,0,1])]).
interpretation( 3, [number = 4,seconds = 0], [
    function(A, [0]),
    function(B, [1]),
    function(C, [2]),
    relation(da, [0]),
    relation(ja, [1]),
    relation(q1_da, [0]),
    relation(q1_ja, [1]),
    relation(q2_da, [1]),
    relation(q2_ja, [0]),
    relation(q3_da, [1]),
    relation(q3_ja, [0]),
    relation(false(_), [0,1,0]),
    relation(isFalse(_), [0,1,0]),
    relation(isRand(_), [1,0,0]),
    relation(isTrue(_), [0,0,1]),
    relation(rand(_), [1,0,0]),
    relation(true(_), [0,0,1])]).
interpretation( 3, [number = 5,seconds = 0], [
    function(A, [0]),
    function(B, [1]),
    function(C, [2]),
    relation(da, [0]),
    relation(ja, [1]),
    relation(q1_da, [1]),
    relation(q1_ja, [0]),
    relation(q2_da, [0]),
    relation(q2_ja, [1]),
    relation(q3_da, [0]),
    relation(q3_ja, [1]),
    relation(false(_), [1,0,0]),
    relation(isFalse(_), [1,0,0]),
    relation(isRand(_), [0,1,0]),
    relation(isTrue(_), [0,0,1]),
    relation(rand(_), [0,1,0]),
    relation(true(_), [0,0,1])]).
interpretation( 3, [number = 6,seconds = 0], [
    function(A, [0]),
    function(B, [1]),
    function(C, [2]),
    relation(da, [0]),
    relation(ja, [1]),
    relation(q1_da, [1]),
    relation(q1_ja, [0]),
    relation(q2_da, [0]),
    relation(q2_ja, [1]),
    relation(q3_da, [1]),
    relation(q3_ja, [0]),
    relation(false(_), [1,0,0]),
    relation(isFalse(_), [1,0,0]),
    relation(isRand(_), [0,0,1]),
    relation(isTrue(_), [0,1,0]),
    relation(rand(_), [0,0,1]),
    relation(true(_), [0,1,0])]).
interpretation( 3, [number = 7,seconds = 0], [
    function(A, [0]),
    function(B, [1]),
    function(C, [2]),
    relation(da, [0]),
    relation(ja, [1]),
    relation(q1_da, [1]),
    relation(q1_ja, [0]),
    relation(q2_da, [1]),
    relation(q2_ja, [0]),
    relation(q3_da, [0]),
    relation(q3_ja, [1]),
    relation(false(_), [0,0,1]),
    relation(isFalse(_), [0,0,1]),
    relation(isRand(_), [0,1,0]),
    relation(isTrue(_), [1,0,0]),
    relation(rand(_), [0,1,0]),
    relation(true(_), [1,0,0])]).
interpretation( 3, [number = 8,seconds = 0], [
    function(A, [0]),
    function(B, [1]),
    function(C, [2]),
    relation(da, [0]),
    relation(ja, [1]),
    relation(q1_da, [1]),
    relation(q1_ja, [0]),
    relation(q2_da, [1]),
    relation(q2_ja, [0]),
    relation(q3_da, [1]),
    relation(q3_ja, [0]),
    relation(false(_), [0,1,0]),
    relation(isFalse(_), [0,1,0]),
    relation(isRand(_), [0,0,1]),
    relation(isTrue(_), [1,0,0]),
    relation(rand(_), [0,0,1]),
    relation(true(_), [1,0,0])]).
interpretation( 3, [number = 9,seconds = 0], [
    function(A, [0]),
    function(B, [1]),
    function(C, [2]),
    relation(da, [1]),
    relation(ja, [0]),
    relation(q1_da, [0]),
    relation(q1_ja, [1]),
    relation(q2_da, [0]),
    relation(q2_ja, [1]),
    relation(q3_da, [0]),
    relation(q3_ja, [1]),
    relation(false(_), [0,0,1]),
    relation(isFalse(_), [0,0,1]),
    relation(isRand(_), [0,1,0]),
    relation(isTrue(_), [1,0,0]),
    relation(rand(_), [0,1,0]),
    relation(true(_), [1,0,0])]).
interpretation( 3, [number = 10,seconds = 0], [
    function(A, [0]),
    function(B, [1]),
    function(C, [2]),
    relation(da, [1]),
    relation(ja, [0]),
    relation(q1_da, [0]),
    relation(q1_ja, [1]),
    relation(q2_da, [0]),
    relation(q2_ja, [1]),
    relation(q3_da, [1]),
    relation(q3_ja, [0]),
    relation(false(_), [0,0,1]),
    relation(isFalse(_), [0,0,1]),
    relation(isRand(_), [1,0,0]),
    relation(isTrue(_), [0,1,0]),
    relation(rand(_), [1,0,0]),
    relation(true(_), [0,1,0])]).
interpretation( 3, [number = 11,seconds = 0], [
    function(A, [0]),
    function(B, [1]),
    function(C, [2]),
    relation(da, [1]),
    relation(ja, [0]),
    relation(q1_da, [0]),
    relation(q1_ja, [1]),
    relation(q2_da, [1]),
    relation(q2_ja, [0]),
    relation(q3_da, [0]),
    relation(q3_ja, [1]),
    relation(false(_), [1,0,0]),
    relation(isFalse(_), [1,0,0]),
    relation(isRand(_), [0,1,0]),
    relation(isTrue(_), [0,0,1]),
    relation(rand(_), [0,1,0]),
    relation(true(_), [0,0,1])]).
interpretation( 3, [number = 12,seconds = 0], [
    function(A, [0]),
    function(B, [1]),
    function(C, [2]),
    relation(da, [1]),
    relation(ja, [0]),
    relation(q1_da, [0]),
    relation(q1_ja, [1]),
    relation(q2_da, [1]),
    relation(q2_ja, [0]),
    relation(q3_da, [1]),
    relation(q3_ja, [0]),
    relation(false(_), [0,1,0]),
    relation(isFalse(_), [0,1,0]),
    relation(isRand(_), [1,0,0]),
    relation(isTrue(_), [0,0,1]),
    relation(rand(_), [1,0,0]),
    relation(true(_), [0,0,1])]).
interpretation( 3, [number = 13,seconds = 0], [
    function(A, [0]),
    function(B, [1]),
    function(C, [2]),
    relation(da, [1]),
    relation(ja, [0]),
    relation(q1_da, [1]),
    relation(q1_ja, [0]),
    relation(q2_da, [0]),
    relation(q2_ja, [1]),
    relation(q3_da, [0]),
    relation(q3_ja, [1]),
    relation(false(_), [1,0,0]),
    relation(isFalse(_), [1,0,0]),
    relation(isRand(_), [0,1,0]),
    relation(isTrue(_), [0,0,1]),
    relation(rand(_), [0,1,0]),
    relation(true(_), [0,0,1])]).
interpretation( 3, [number = 14,seconds = 0], [
    function(A, [0]),
    function(B, [1]),
    function(C, [2]),
    relation(da, [1]),
    relation(ja, [0]),
    relation(q1_da, [1]),
    relation(q1_ja, [0]),
    relation(q2_da, [0]),
    relation(q2_ja, [1]),
    relation(q3_da, [1]),
    relation(q3_ja, [0]),
    relation(false(_), [1,0,0]),
    relation(isFalse(_), [1,0,0]),
    relation(isRand(_), [0,0,1]),
    relation(isTrue(_), [0,1,0]),
    relation(rand(_), [0,0,1]),
    relation(true(_), [0,1,0])]).
interpretation( 3, [number = 15,seconds = 0], [
    function(A, [0]),
    function(B, [1]),
    function(C, [2]),
    relation(da, [1]),
    relation(ja, [0]),
    relation(q1_da, [1]),
    relation(q1_ja, [0]),
    relation(q2_da, [1]),
    relation(q2_ja, [0]),
    relation(q3_da, [0]),
    relation(q3_ja, [1]),
    relation(false(_), [0,0,1]),
    relation(isFalse(_), [0,0,1]),
    relation(isRand(_), [0,1,0]),
    relation(isTrue(_), [1,0,0]),
    relation(rand(_), [0,1,0]),
    relation(true(_), [1,0,0])]).
interpretation( 3, [number = 16,seconds = 0], [
    function(A, [0]),
    function(B, [1]),
    function(C, [2]),
    relation(da, [1]),
    relation(ja, [0]),
    relation(q1_da, [1]),
    relation(q1_ja, [0]),
    relation(q2_da, [1]),
    relation(q2_ja, [0]),
    relation(q3_da, [1]),
    relation(q3_ja, [0]),
    relation(false(_), [0,1,0]),
    relation(isFalse(_), [0,1,0]),
    relation(isRand(_), [0,0,1]),
    relation(isTrue(_), [1,0,0]),
    relation(rand(_), [0,0,1]),
    relation(true(_), [1,0,0])]).
```

2: Delightful Dresses Zebra Puzzle(Medium)

```
% The Programmer is exactly to the right of the Surgeon.
% The woman wearing the Black dress is somewhere to the
%right of the woman who got the 10% discount.
% The Actress is next to the woman wearing the Black
%dress.
% The Psychologist is next to the woman wearing the
%Bodycon dress.
% Megan is the oldest woman.
% The lady wearing the White dress is somewhere to the
%left of the 36-year-old woman.
% The woman wearing the Sheath dress is next to the
%woman that got 15% off on her new dress.
% The woman wearing the Purple dress is somewhere
%between the youngest woman and the woman wearing the
%White dress, in that order.
% The lady who got the smallest discount is exactly to
%the left of the lady who got the 15% discount.
% The woman who got the 20% discount is somewhere to the
%right of the woman wearing the White dress.
% The Surgeon is exactly to the left of the woman
%wearing the Sundress.
% Anna is wearing the White dress.
% Megan is immediately before the woman who got the 5%
```

```
%discount.
% The Sheath dress is Red.
% At one of the ends is the 33-year-old woman.
% Sara is exactly to the right of the lady wearing the
%Bodycon dress.
% Lauren is 33 years old.
% At one of the ends is the woman wearing the Sheath
%dress.
% The woman who got the 10% discount is exactly to the
%left of the woman who got 5% off.
% The A-line dress is Purple.

set(arithmetic).          %Pentru relatiile de vecinatate.
assign(domain_size, 5).   %Sunt cinci femei in total cu indicii {0,1,2,3,4}

list(distinct).

  %Culoarea rochiei e distincta.
  [Black,Blue,White,Purple,Red].

  %Numele e distinct.
  [Anna,Erica,Lauren,Meghan,Sara].

  %Profesia e distincta.
  [Actress,Electrician,Programmer,Psychologist,Surgeon].

  %Tipul rochie e distinct.
  [Aline,Bodycon,Sheath,Sundress,Wrap].

  %Varsta e distincta.
  [Thirty,Thirtythree,Thirtysix,Thirtynine,Fourty].

  %Discount-ul e distinct.
  [Five,Ten,Fifteen,Twenty,TwentyFive].

end_of_list.

formulas(assumptions).

  % Definitii pentru vecinii exact din dreapta, exact
  %din stanga.
  right_neighbor(x,y) <-> x = y + 1.
  left_neighbor(x,y) <-> x + 1 = y.

  % Definitii pentru vecinii din dreapta sau stanga(care
  %nu neaparat sunt exact in dreapta sau stanga).
  right_neighbor_ish(x,y) <-> x < y.
  left_neighbor_ish(x,y) <-> x > y.

  % Definitie pentru relatia de vecin.
  neighbors(x,y) <-> right_neighbor(x,y) | right_neighbor(y,x).

  % Indici
  right_neighbor(Programmer,Surgeon).
  right_neighbor_ish(Ten,Black).
  neighbors(Actress,Black).
  neighbors(Psychologist,Bodycon).
  Meghan = Fourty.
  left_neighbor_ish(Thirtysix,White).
  neighbors(Sheath,Fifteen).
  Thirty < Purple & Purple < White.
  left_neighbor(Five,Fifteen).
  right_neighbor_ish(White,Twenty).
  left_neighbor(Surgeon,Sundress).
  Anna = White.
  left_neighbor(Meghan,Five).
  Sheath = Red.
  Thirtythree = 0 | Thirtythree = 4.
  right_neighbor(Sara,Bodycon).
  Lauren = Thirtythree.
  Sheath = 0 | Sheath = 4.
  left_neighbor(Ten,Five).
  Aline = Purple.

end_of_list.

=== Mace4 starting on domain size 5. ===

------ process 4012 exit (max_models) ------
interpretation( 5, [number = 1,seconds = 0], [
    function(Actress, [4]),
    function(Aline, [1]),
    function(Anna, [2]),
    function(Black, [3]),
    function(Bodycon, [2]),
    function(Fifteen, [3]),
    function(Five, [2]),
    function(Fourty, [1]),
    function(Lauren, [4]),
    function(Meghan, [1]),
    function(Programmer, [3]),
    function(Psychologist, [1]),
    function(Purple, [1]),
    function(Red, [4]),
    function(Sara, [3]),
    function(Sheath, [4]),
    function(Sundress, [3]),
    function(Surgeon, [2]),
    function(Ten, [1]),
    function(Thirty, [0]),
    function(Thirtysix, [3]),
    function(Thirtythree, [4]),
    function(Twenty, [4]),
    function(White, [2]),
    relation(left_neighbor(,), [
        0,1,0,0,0,
        0,0,1,0,0,
        0,0,0,1,0,
        0,0,0,0,1,
        0,0,0,0,0]),
    relation(left_neighbor_ish(,), [
        0,0,0,0,0,
        1,0,0,0,0,
        1,1,0,0,0,
        1,1,1,0,0,
        1,1,1,1,0]),
    relation(neighbors(,), [
        0,1,0,0,0,
        1,0,1,0,0,
        0,1,0,1,0,
        0,0,1,0,1,
        0,0,0,1,0]),
    relation(right_neighbor(,), [
        0,0,0,0,0,
        1,0,0,0,0,
        0,1,0,0,0,
        0,0,1,0,0,
        0,0,0,1,0]),
    relation(right_neighbor_ish(,), [
        0,1,1,1,1,
        0,0,1,1,1,
        0,0,0,1,1,
        0,0,0,0,1,
        0,0,0,0,0]),
    function(Blue, [0]),
    function(Electrician, [0]),
    function(Erica, [0]),
    function(Thirtynine, [2]),
    function(TwentyFive, [0]),
    function(Wrap, [0])]).
```

3: Illegible Letters(Easy)

```
% Saved by Prover9-Mace4 Version 0.5, December 2007.

set(ignore_option_dependencies). % GUI handles dependencies

if(Prover9). % Options for Prover9
  assign(max_seconds, 60).
end_if.
```

```
if(Mace4).   % Options for Mace4
  assign(max_seconds, 60).
end_if.

formulas(assumptions).

% All the dated letters in this room are written on blue paper.
all x ( datedLetter(x) -> bluePaper(x) ).

% None of them are in black ink except those that are
%written in the third person.
all x ( blackInk(x) <-> thirdPerson(x) ).

% AI have not filed any of them that I can read.
all x ( read(x) -> -filed(x) ).

% None of them that are written on one sheet are undated.
all x ( oneSheet(x) -> datedLetter(x) ).

% All of them that are not crossed are in black ink. :
all x ( -crossed(x) -> blackInk(x) ).

% All of them written by Brown begin with "Dear Sir" :
all x ( writtenByBrown(x) -> beginsWithDearSir(x) ).

% All of them written on blue paper are filed. :
all x ( bluePaper(x) -> filed(x) ).

% None of them written on more than one sheet are crossed.
all x ( oneSheet(x) <-> crossed(x) ).

% None of them that begin with "Dear Sir" are written in third person.
all x ( beginsWithDearSir(x) -> -thirdPerson(x) ).

end_of_list.

formulas(goals).

writtenByBrown(x) -> -read(x).

end_of_list.

============================== prooftrans ============================
Prover9 (32) version Dec-2007, Dec 2007.
Process 12112 was started by lexri on Veronica,
Sun Nov 29 17:36:32 2020
The command was "/cygdrive/c/Program Files (x86)/Prover9-Mace4/bin-win32/prover9".
============================== end of head ===========================
============================== end of input ==========================

============================== PROOF =================================

% -------- Comments from original proof --------
% Proof 1 at 0.03 (+ 0.00) seconds.
% Length of proof is 31.
% Level of proof is 7.
% Maximum clause weight is 0.
% Given clauses 0.

1 (all x (datedLetter(x) -> bluePaper(x))) # label(non_clause).  [assumption].
2 (all x (blackInk(x) <-> thirdPerson(x))) # label(non_clause).  [assumption].
3 (all x (read(x) -> -filed(x))) # label(non_clause).  [assumption].
4 (all x (oneSheet(x) -> datedLetter(x))) # label(non_clause).  [assumption].
5 (all x (-crossed(x) -> blackInk(x))) # label(non_clause).  [assumption].
6 (all x (writtenByBrown(x) -> beginsWithDearSir(x))) # label(non_clause).[assumption].
7 (all x (bluePaper(x) -> filed(x))) # label(non_clause).  [assumption].
8 (all x (oneSheet(x) <-> crossed(x))) # label(non_clause).  [assumption].
9 (all x (beginsWithDearSir(x) -> -thirdPerson(x))) # label(non_clause).  [assumption].
10 writtenByBrown(x) -> -read(x) # label(non_clause) # label(goal).  [goal].
11 -oneSheet(x) | datedLetter(x).  [clausify(4)].
12 -datedLetter(x) | bluePaper(x).  [clausify(1)].
14 -blackInk(x) | thirdPerson(x).  [clausify(2)].
15 crossed(x) | blackInk(x).  [clausify(5)].
16 read(c1).  [deny(10)].
17 -read(x) | -filed(x).  [clausify(3)].
18 writtenByBrown(c1).  [deny(10)].
19 -writtenByBrown(x) | beginsWithDearSir(x).  [clausify(6)].
20 -oneSheet(x) | bluePaper(x).  [resolve(11,b,12,a)].
21 -bluePaper(x) | filed(x).  [clausify(7)].
22 oneSheet(x) | -crossed(x).  [clausify(8)].
24 -oneSheet(x) | filed(x).  [resolve(20,b,21,a)].
25 beginsWithDearSir(c1).  [resolve(18,a,19,a)].
26 -beginsWithDearSir(x) | -thirdPerson(x).  [clausify(9)].
27 filed(x) | -crossed(x).  [resolve(24,a,22,a)].
28 crossed(x) | thirdPerson(x).  [resolve(15,b,14,a)].
29 filed(x) | thirdPerson(x).  [resolve(27,b,28,a)].
30 -filed(c1).  [resolve(16,a,17,a)].
31 thirdPerson(c1).  [resolve(29,a,30,a)].
32 -thirdPerson(c1).  [resolve(25,a,26,a)].
33 $F.  [resolve(31,a,32,a)].

============================== end of proof ==========================
```

4: The SeaGod and the WarGod(Easy)

```
    set(ignore_option_dependencies). % GUI handles dependencies
```

```
if(Prover9). % Options for Prover9
  set(paramodulation).
  assign(max_seconds, 60).
end_if.

if(Mace4).   % Options for Mace4
  assign(max_seconds, 60).
end_if.

if(Prover9). % Additional input for Prover9
% Zeus este zeul cerului si Poseidon este zeul oceanului. Acestia sunt frati.
% Uranus o are ca fiica pe Rhea. Rhea este mama lui Zeus.
% Ares este fiul lui Zeus si zeul razboiului. Atena este sora lui Ares.
% Iar Triton este fiul lui Poseidon.

% predicate Mother(x,y): x is motehr of y
% predicate Father(x,y): x is father of y
% predicate Sibling(x,y): x is sibling to y
% predicate Cousin(x,y): x is cousin to y
end_if.

formulas(assumptions).

Father(Uranus,Rhea).
 Mother(Rhea, Zeus).
 Mother(Rhea, Poseidon).
 Sibling(Poseidon,Zeus).
 Sibling(Zeus,Poseidon).
 Father(Poseidon,Triton).
 Father(Zeus,Atena).
 Father(Zeus,Ares).
 Sibling(Atena,Ares).
 Sibling(Ares,Atena).

 Father(x,y) | Mother(x,y) -> Parent(x,y).
 Parent(x,y) & Parent(y,z) -> Grandparent(x,z).
 Parent(x,y) & Grandparent(y,z) -> Greatgrandparent(x,z).
 Sibling(x,y) & Parent(y,z) -> Uncle(x,z).
 Parent(x,y) & Sibling(x,z) | Sibling(z,x) & Parent(z,u) -> Cousin(y,u).

 Zeus = ThunderGod.
 Poseidon = SeaGod.
 Ares = WarGod.

end_of_list.

formulas(goals).

%Greatgrandparent(Uranus,WarGod).
   %Uncle(ThunderGod,Triton).
   %Uncle(SeaGod,Atena).
   Cousin(Atena,Triton).

end_of_list.

============================== prooftrans ============================
Prover9 (32) version Dec-2007, Dec 2007.
Process 19128 was started by lexri on Veronica,
Mon Nov 30 13:08:33 2020
The command was "/cygdrive/c/Program Files (x86)/Prover9-Mace4/bin-win32/prover9".
============================== end of head ===========================

============================== end of input ==========================

============================== PROOF =================================

% -------- Comments from original proof --------
% Proof 1 at 0.03 (+ 0.00) seconds.
% Length of proof is 12.
% Level of proof is 4.
% Maximum clause weight is 6.
% Given clauses 12.

1 Father(x,y) | Mother(x,y) -> Parent(x,y) # label(non_clause).  [assumption].
5 Parent(x,y) & Sibling(x,z) | Sibling(z,x) & Parent(z,u) -> Cousin(y,u) # label(non_cla
6 Cousin(Atena,Triton) # label(non_clause) # label(goal).  [goal].
7 -Father(x,y) | Parent(x,y).  [clausify(1)].
10 Father(Zeus,Atena).  [assumption].
17 Sibling(Zeus,Poseidon).  [assumption].
20 -Parent(x,y) | -Sibling(x,z) | Cousin(y,u).  [clausify(5)].
30 -Cousin(Atena,Triton).  [deny(6)].
33 Parent(Zeus,Atena).  [resolve(7,a,10,a)].
38 -Parent(Zeus,x) | Cousin(x,y).  [resolve(20,b,17,a)].
48 Cousin(Atena,x).  [hyper(38,a,33,a)].
49 $F.  [resolve(48,a,30,a)].

============================== end of proof ==========================
```

A.0.3 Planning

1: The All-Out Problem

```
(define (domain all-out-d)
    (:requirements :negative-preconditions
    :conditional-effects :equality)
    (:predicates
        (is-up ?coin)
        (has-two-neighbours ?coin)
        (has-three-neighbours ?coin)
        (has-four-neighbours ?coin)
        (is-adjacent ?coin1 ?coin2)
```

```
)
(:action flip5
    :parameters (
        ?coinC
        ?coinT
        ?coinB
        ?coinL
        ?coinR
    )
    :precondition (and
     (not (= ?coinC ?coinT))
     (not (= ?coinC ?coinB))
     (not (= ?coinC ?coinL))
     (not (= ?coinC ?coinR))
     (not (= ?coinT ?coinB))
     (not (= ?coinT ?coinL))
     (not (= ?coinT ?coinR))
     (not (= ?coinB ?coinL))
     (not (= ?coinB ?coinR))
     (not (= ?coinL ?coinR))
     (not (has-two-neighbours ?coinC))
     (not (has-three-neighbours ?coinC))
     (has-four-neighbours ?coinC)
     (is-adjacent ?coinC ?coinT)
     (is-adjacent ?coinC ?coinB)
     (is-adjacent ?coinC ?coinL)
     (is-adjacent ?coinC ?coinR)

    )
    :effect (and
     (when (is-up ?coinC) (not (is-up ?coinC)))
     (when (not (is-up ?coinC)) (is-up ?coinC))
     (when (is-up ?coinT) (not (is-up ?coinT)))
     (when (not (is-up ?coinT)) (is-up ?coinT))
     (when (is-up ?coinB) (not (is-up ?coinB)))
     (when (not (is-up ?coinB)) (is-up ?coinB))
     (when (is-up ?coinL) (not (is-up ?coinL)))
     (when (not (is-up ?coinL)) (is-up ?coinL))
     (when (is-up ?coinR) (not (is-up ?coinR)))
     (when (not (is-up ?coinR)) (is-up ?coinR))
    )
)
(:action flip4
    :parameters (
        ?coinC
        ?coin1
        ?coin2
        ?coin3
    )
    :precondition (and
            (not (= ?coinC ?coin1))
            (not (= ?coinC ?coin2))
            (not (= ?coinC ?coin3))
            (not (= ?coin1 ?coin2))
            (not (= ?coin1 ?coin3))
            (not (= ?coin2 ?coin3))
            (not (has-two-neighbours ?coinC))
            (has-three-neighbours ?coinC)
            (not (has-four-neighbours ?coinC))
            (is-adjacent ?coinC ?coin1)
            (is-adjacent ?coinC ?coin2)
            (is-adjacent ?coinC ?coin3)
        )
        :effect (and
            (when (is-up ?coinC) (not (is-up ?coinC)))
            (when (not (is-up ?coinC)) (is-up ?coinC))
            (when (is-up ?coin1) (not (is-up ?coin1)))
            (when (not (is-up ?coin1)) (is-up ?coin1))
            (when (is-up ?coin2) (not (is-up ?coin2)))
            (when (not (is-up ?coin2)) (is-up ?coin2))
            (when (is-up ?coin3) (not (is-up ?coin3)))
            (when (not (is-up ?coin3)) (is-up ?coin3))
        )
    )
    (:action flip3
        :parameters (
            ?coinC
            ?coin1
            ?coin2
        )
        :precondition (and
            (not (= ?coinC ?coin1))
            (not (= ?coinC ?coin2))
            (not (= ?coin1 ?coin2))
            (has-two-neighbours ?coinC)
            (not (has-three-neighbours ?coinC))
            (not (has-four-neighbours ?coinC))
            (is-adjacent ?coinC ?coin1)
            (is-adjacent ?coinC ?coin2)
        )
        :effect (and
            (when (is-up ?coinC) (not (is-up ?coinC)))
            (when (not (is-up ?coinC)) (is-up ?coinC))
            (when (is-up ?coin1) (not (is-up ?coin1)))
            (when (not (is-up ?coin1)) (is-up ?coin1))
            (when (is-up ?coin2) (not (is-up ?coin2)))
            (when (not (is-up ?coin2)) (is-up ?coin2))
        )
    )
)
```

```
(define (problem all-out-p)
    (:domain all-out-d)
    (:objects
        coin11 coin12 coin13 coin14 coin15
        coin21 coin22 coin23 coin24 coin25
        coin31 coin32 coin33 coin34 coin35
        coin41 coin42 coin43 coin44 coin45
        coin51 coin52 coin53 coin54 coin55
    )
    (:init
        ; init adjacency
        ; row 1
        ; column 1
        (not (is-adjacent coin11 coin11))
        (is-adjacent coin11 coin12)
        (is-adjacent coin11 coin21)
        (has-two-neighbours coin11)
        (not (has-three-neighbours coin11))
        (not (has-four-neighbours coin11))
        ; column 2
        (not (is-adjacent coin12 coin12))
        (is-adjacent coin12 coin11)
        (is-adjacent coin12 coin22)
        (is-adjacent coin12 coin13)
        (not (has-two-neighbours coin12))
        (has-three-neighbours coin12)
        (not (has-four-neighbours coin12))
        ; column 3
        (not (is-adjacent coin13 coin13))
        (is-adjacent coin13 coin12)
        (is-adjacent coin13 coin23)
        (is-adjacent coin13 coin14)
        (not (has-two-neighbours coin13))
        (has-three-neighbours coin13)
        (not (has-four-neighbours coin13))
        ; column 4
        (not (is-adjacent coin14 coin14))
        (is-adjacent coin14 coin13)
        (is-adjacent coin14 coin24)
        (is-adjacent coin14 coin15)
        (not (has-two-neighbours coin14))
        (has-three-neighbours coin14)
        (not (has-four-neighbours coin14))
        ; column 5
        (not (is-adjacent coin15 coin15))
        (is-adjacent coin15 coin14)
        (is-adjacent coin15 coin25)
        (has-two-neighbours coin15)
        (not (has-three-neighbours coin15))
```

```
(not (has-four-neighbours coin15))

; row 2
; column 1
(not (is-adjacent coin21 coin21))
(is-adjacent coin21 coin11)
(is-adjacent coin21 coin22)
(is-adjacent coin21 coin31)
(not (has-two-neighbours coin21))
(has-three-neighbours coin21)
(not (has-four-neighbours coin21))
; column 2
(not (is-adjacent coin22 coin22))
(is-adjacent coin22 coin21)
(is-adjacent coin22 coin12)
(is-adjacent coin22 coin32)
(is-adjacent coin22 coin23)
(not (has-two-neighbours coin22))
(not (has-three-neighbours coin22))
(has-four-neighbours coin22)
; column 3
(not (is-adjacent coin23 coin23))
(is-adjacent coin23 coin22)
(is-adjacent coin23 coin13)
(is-adjacent coin23 coin33)
(is-adjacent coin23 coin24)
(not (has-two-neighbours coin23))
(not (has-three-neighbours coin23))
(has-four-neighbours coin23)
; column 4
(not (is-adjacent coin24 coin24))
(is-adjacent coin24 coin23)
(is-adjacent coin24 coin14)
(is-adjacent coin24 coin34)
(is-adjacent coin24 coin25)
(not (has-two-neighbours coin24))
(not (has-three-neighbours coin24))
(has-four-neighbours coin24)
; column 5
(not (is-adjacent coin25 coin25))
(is-adjacent coin25 coin15)
(is-adjacent coin25 coin24)
(is-adjacent coin25 coin35)
(not (has-two-neighbours coin25))
(has-three-neighbours coin25)
(not (has-four-neighbours coin25))

; row 3
; column 1
(not (is-adjacent coin31 coin31))
```

```
(is-adjacent coin31 coin21)
(is-adjacent coin31 coin32)
(is-adjacent coin31 coin41)
(not (has-two-neighbours coin31))
(has-three-neighbours coin31)
(not (has-four-neighbours coin31))
; column 2
(not (is-adjacent coin32 coin32))
(is-adjacent coin32 coin31)
(is-adjacent coin32 coin22)
(is-adjacent coin32 coin42)
(is-adjacent coin32 coin33)
(not (has-two-neighbours coin32))
(not (has-three-neighbours coin32))
(has-four-neighbours coin32)
; column 3
(not (is-adjacent coin33 coin33))
(is-adjacent coin33 coin32)
(is-adjacent coin33 coin23)
(is-adjacent coin33 coin43)
(is-adjacent coin33 coin34)
(not (has-two-neighbours coin33))
(not (has-three-neighbours coin33))
(has-four-neighbours coin33)
; column 4
(not (is-adjacent coin34 coin34))
(is-adjacent coin34 coin24)
(is-adjacent coin34 coin33)
(is-adjacent coin34 coin44)
(is-adjacent coin34 coin35)
(not (has-two-neighbours coin34))
(not (has-three-neighbours coin34))
(has-four-neighbours coin34)
; column 5
(not (is-adjacent coin35 coin35))
(is-adjacent coin35 coin25)
(is-adjacent coin35 coin34)
(is-adjacent coin35 coin45)
(not (has-two-neighbours coin35))
(has-three-neighbours coin35)
(not (has-four-neighbours coin35))

; row 4
; column 1
(not (is-adjacent coin41 coin41))
(is-adjacent coin41 coin31)
(is-adjacent coin41 coin42)
(is-adjacent coin41 coin51)
(not (has-two-neighbours coin41))
(has-three-neighbours coin41)
```

```
(not (has-four-neighbours coin41))
; column 2
(not (is-adjacent coin42 coin42))
(is-adjacent coin42 coin41)
(is-adjacent coin42 coin32)
(is-adjacent coin42 coin43)
(is-adjacent coin42 coin52)
(not (has-two-neighbours coin42))
(not (has-three-neighbours coin42))
(has-four-neighbours coin42)
; column 3
(not (is-adjacent coin43 coin43))
(is-adjacent coin43 coin42)
(is-adjacent coin43 coin33)
(is-adjacent coin43 coin44)
(is-adjacent coin43 coin53)
(not (has-two-neighbours coin43))
(not (has-three-neighbours coin43))
(has-four-neighbours coin43)
; column 4
(not (is-adjacent coin44 coin44))
(is-adjacent coin44 coin43)
(is-adjacent coin44 coin34)
(is-adjacent coin44 coin45)
(is-adjacent coin44 coin54)
(not (has-two-neighbours coin44))
(not (has-three-neighbours coin44))
(has-four-neighbours coin44)
; column 5
(not (is-adjacent coin45 coin45))
(is-adjacent coin45 coin35)
(is-adjacent coin45 coin44)
(is-adjacent coin45 coin55)
(not (has-two-neighbours coin45))
(has-three-neighbours coin45)
(not (has-four-neighbours coin45))

; row 5
; column 1
(not (is-adjacent coin51 coin51))
(is-adjacent coin51 coin41)
(is-adjacent coin51 coin52)
(has-two-neighbours coin51)
(not (has-three-neighbours coin51))
(not (has-four-neighbours coin51))
; column 2
(not (is-adjacent coin52 coin52))
(is-adjacent coin52 coin51)
(is-adjacent coin52 coin42)
(is-adjacent coin52 coin53)
```

```
(not (has-two-neighbours coin52))
(has-three-neighbours coin52)
(not (has-four-neighbours coin52))
; column 3
(not (is-adjacent coin53 coin53))
(is-adjacent coin53 coin52)
(is-adjacent coin53 coin43)
(is-adjacent coin53 coin54)
(not (has-two-neighbours coin53))
(has-three-neighbours coin53)
(not (has-four-neighbours coin53))
; column 4
(not (is-adjacent coin54 coin54))
(is-adjacent coin54 coin53)
(is-adjacent coin54 coin44)
(is-adjacent coin54 coin55)
(not (has-two-neighbours coin54))
(has-three-neighbours coin54)
(not (has-four-neighbours coin54))
;column 5
(not (is-adjacent coin55 coin55))
(is-adjacent coin55 coin54)
(is-adjacent coin55 coin45)
(has-two-neighbours coin55)
(not (has-three-neighbours coin55))
(not (has-four-neighbours coin55))

; init all coin faces down
(not (is-up coin11))
(not (is-up coin12))
(not (is-up coin13))
(not (is-up coin14))
(not (is-up coin15))
; row 2
(not (is-up coin21))
(not (is-up coin22))
(not (is-up coin23))
(not (is-up coin24))
(not (is-up coin25))
; row 3
(not (is-up coin31))
(not (is-up coin32))
(not (is-up coin33))
(not (is-up coin34))
(not (is-up coin35))
; row 4
(not (is-up coin41))
(not (is-up coin42))
(not (is-up coin43))
(not (is-up coin44))
```

```
        (not (is-up coin45))
        ; row 5
        (not (is-up coin51))
        (not (is-up coin52))
        (not (is-up coin53))
        (not (is-up coin54))
        (not (is-up coin55))
    )
    (:goal
        (and
            ; all coin faces up
            ; row 1
            (is-up coin11)
            (is-up coin12)
            (is-up coin13)
            (is-up coin14)
            (is-up coin15)
            ; row 2
            (is-up coin21)
            (is-up coin22)
            (is-up coin23)
            (is-up coin24)
            (is-up coin25)
            ; row 3
            (is-up coin31)
            (is-up coin32)
            (is-up coin33)
            (is-up coin34)
            (is-up coin35)
            ; row 4
            (is-up coin41)
            (is-up coin42)
            (is-up coin43)
            (is-up coin44)
            (is-up coin45)
            ; row 5
            (is-up coin51)
            (is-up coin52)
            (is-up coin53)
            (is-up coin54)
            (is-up coin55)
        )
    )
)

(flip5 coin22 coin12 coin21 coin23 coin32)
(flip4 coin34 coin24 coin33 coin44)
(flip4 coin42 coin43 coin43 coin43)
(flip3 coin41 coin31 coin31)
(flip4 coin13 coin12 coin12 coin14)
```

```
(flip3 coin11 coin12 coin12)
; cost = 6 (unit cost)
```

2: Treasure

```
(define (domain treasure)
    (:predicates
        (hunter-at ?r ?c)
        (is-rock ?r ?c)
        (is-another-hunter ?r ?c)
        (next-row ?r1 ?r2)
        (next-column ?c1 ?c2)
        (avoiding-row ?r1 ?r2 ?r3)
        (avoiding-column ?c1 ?c2 ?c3)
    )
    (:action dig-up
      :parameters (?old-row ?new-row ?col)
      :precondition (and (next-row ?new-row ?old-row)
          (hunter-at ?old-row ?col)
          (not(is-rock ?new-row ?col))
          (not(is-another-hunter ?new-row ?col))
     )
      :effect (and (not (hunter-at ?old-row ?col))
     (hunter-at ?new-row ?col)
       )
    )

    (:action dig-down
      :parameters (?old-row ?new-row ?col)
      :precondition (and (next-row ?old-row ?new-row)
          (hunter-at ?old-row ?col)
          (not (is-rock ?new-row ?col))
          (not (is-another-hunter ?new-row ?col))
     )
      :effect (and (not (hunter-at ?old-row ?col))
            (hunter-at ?new-row ?col)
       )
    )

    (:action dig-left
      :parameters (?row ?old-col ?new-col)
      :precondition (and (next-column ?new-col ?old-col)
          (hunter-at ?row ?old-col)
          (not (is-rock ?row ?new-col))
          (not (is-another-hunter ?row ?new-col))
         )
      :effect (and (not (hunter-at ?row ?old-col))
     (hunter-at ?row ?new-col)

       )
```

```
)

(:action dig-right
  :parameters (?row ?old-col ?new-col)
  :precondition (and (next-column ?old-col ?new-col)
 (hunter-at ?row ?old-col)
 (not (is-rock ?row ?new-col))
 (not (is-another-hunter ?row ?new-col)))
  :effect (and (not (hunter-at ?row ?old-col))
 (hunter-at ?row ?new-col)
   )
)

(:action dig-avoid-up
  :parameters (?old-row ?new-row ?new-rowa ?col)
  :precondition (and (avoiding-row ?new-rowa ?new-row ?old-row)
      (hunter-at ?old-row ?col)
      (is-rock ?new-row ?col)
      (not (is-rock ?new-rowa ?col))
      (not (is-another-hunter ?new-rowa ?col))
 )
  :effect (and (not (hunter-at ?old-row ?col))
 (hunter-at ?new-rowa ?col)
   )
)

(:action dig-avoid-down
  :parameters (?old-row ?new-row ?new-rowa ?col)
  :precondition (and (not (is-rock ?new-rowa ?col))
      (avoiding-row ?old-row ?new-row ?new-rowa)
      (hunter-at ?old-row ?col)
      (is-rock ?new-row ?col)
      (not (is-another-hunter ?new-rowa ?col))
  )
  :effect (and (not (hunter-at ?old-row ?col))
        (hunter-at ?new-rowa ?col)
   )
)

(:action dig-avoid-left
  :parameters (?row ?old-col ?new-col ?new-cola)
  :precondition (and (avoiding-column ?new-cola ?new-col ?old-col)
      (hunter-at ?row ?old-col)
      (is-rock ?row ?new-col)
      (not (is-rock ?row ?new-cola))
      (not (is-another-hunter ?row ?new-cola))
 )
  :effect (and (not (hunter-at ?row ?old-col))
 (hunter-at ?row ?new-cola)
```

```
        )
     )

     (:action dig-avoid-right
       :parameters (?row ?old-col ?new-col ?new-cola)
       :precondition (and (avoiding-column ?old-col ?new-col ?new-cola)
      (hunter-at ?row ?old-col)
      (is-rock ?row ?new-col)
      (not (is-rock ?row ?new-cola))
      (not (is-another-hunter ?row ?new-cola))
      )
       :effect (and (not (hunter-at ?row ?old-col))
      (hunter-at ?row ?new-cola)

        )
     )
)
(define (problem treasure1)
     (:domain treasure)
     (:objects
        row1 row2 row3 row4 row5 row6
        col1 col2 col3 col4 col5 col6 )
     (:init

       (next-row row1 row2)          (next-column col1 col2)
       (next-row row2 row3)          (next-column col2 col3)
       (next-row row3 row4)          (next-column col3 col4)
       (next-row row4 row5)          (next-column col4 col5)
       (next-row row5 row6)          (next-column col5 col6)

       (avoiding-row row1 row2 row3)    (avoiding-column col1 col2 col3)
       (avoiding-row row2 row3 row4)    (avoiding-column col2 col3 col4)
       (avoiding-row row3 row4 row5)    (avoiding-column col3 col4 col5)
       (avoiding-row row4 row5 row6)    (avoiding-column col4 col5 col6)

       (is-rock row1 col3)
       (is-rock row2 col1)
       (is-rock row2 col2)
       (is-rock row3 col4)
       (is-rock row4 col2)
       (is-rock row5 col5)

       (is-another-hunter row1 col4)
       (is-another-hunter row5 col2)
       (is-another-hunter row5 col6)
       (is-another-hunter row3 col1)

       (hunter-at row1 col1)
```

```
    )
    (:goal (hunter-at row6 col6))
)
(define (problem treasure2)
    (:domain treasure)
    (:objects
       row1 row2 row3 row4 row5 row6
       col1 col2 col3 col4 col5 col6 )
    (:init

      (next-row row1 row2)          (next-column col1 col2)
      (next-row row2 row3)         (next-column col2 col3)
      (next-row row3 row4)         (next-column col3 col4)
      (next-row row4 row5)         (next-column col4 col5)
      (next-row row5 row6)         (next-column col5 col6)


      (avoiding-row row1 row2 row3)    (avoiding-column col1 col2 col3)
      (avoiding-row row2 row3 row4)    (avoiding-column col2 col3 col4)
      (avoiding-row row3 row4 row5)    (avoiding-column col3 col4 col5)
      (avoiding-row row4 row5 row6)    (avoiding-column col4 col5 col6)


      (is-rock row1 col6)
      (is-rock row1 col4)
      (is-rock row2 col3)
      (is-rock row4 col3)
      (is-rock row5 col1)
      (is-rock row5 col5)


      (is-another-hunter row2 col5)
      (is-another-hunter row3 col1)
      (is-another-hunter row3 col4)
      (is-another-hunter row5 col3)
      (is-another-hunter row5 col6)

      (hunter-at row1 col5)

    )
    (:goal (hunter-at row6 col3))
)
(define (problem treasure3)
    (:domain treasure)
    (:objects
       row1 row2 row3 row4 row5 row6
       col1 col2 col3 col4 col5 col6 )
    (:init

      (next-row row1 row2)          (next-column col1 col2)
      (next-row row2 row3)          (next-column col2 col3)
      (next-row row3 row4)          (next-column col3 col4)
      (next-row row4 row5)          (next-column col4 col5)
      (next-row row5 row6)          (next-column col5 col6)


      (avoiding-row row1 row2 row3)    (avoiding-column col1 col2 col3)
      (avoiding-row row2 row3 row4)    (avoiding-column col2 col3 col4)
      (avoiding-row row3 row4 row5)    (avoiding-column col3 col4 col5)
      (avoiding-row row4 row5 row6)    (avoiding-column col4 col5 col6)


      (is-rock row1 col4)
      (is-rock row3 col3)
      (is-rock row3 col6)
      (is-rock row5 col1)
      (is-rock row5 col5)
      (is-rock row6 col3)


      (is-another-hunter row4 col1)
      (is-another-hunter row4 col4)
      (is-another-hunter row5 col2)
      (is-another-hunter row5 col6)
      (is-another-hunter row6 col5)

      (hunter-at row6 col1)

    )
    (:goal (hunter-at row1 col6)
    )
)
```