Department of Computer Science
Technical University of Cluj-Napoca

Artificial Intelligence

Laboratory activity

Name: Kovács Paul & Lupu Daniel
Group: 30232/1
Email: kovacs_paul_adrian@yahoo.com
dani.lupu2705@gmail.com

Teaching Assistant: Áron Katona
katona.io.aron@student.utcluj.ro

Contents

Chapter 1

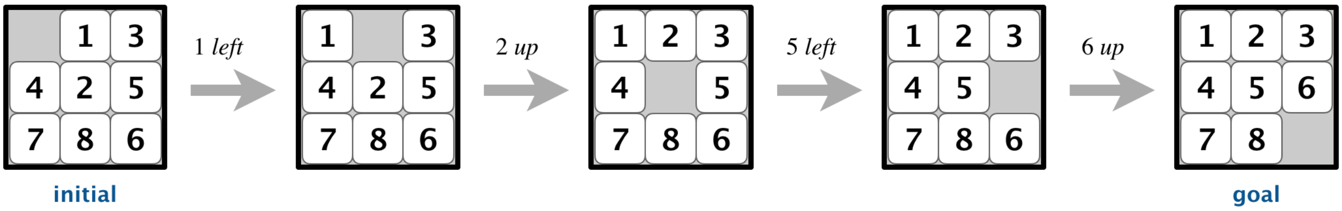
A1: Search

1.1 Introduction

The 15 puzzle is a sliding puzzle having 15 square tiles numbered 1–15 in a frame that is 4 tiles high and 4 tiles wide, leaving one unoccupied tile position. Tiles in the same row or column of the open position can be moved by sliding them horizontally or vertically, respectively. The goal of the puzzle is to place the tiles in numerical order.

Named for the number of tiles in the frame, the 15 puzzle may also be called a 16 puzzle, alluding to its total tile capacity. Similar names are used for different sized variants of the 15 puzzle, such as the 8 puzzle that has 8 tiles in a 3×3 frame.

The n puzzle is a classical problem for modelling algorithms involving heuristics. Commonly used heuristics for this problem include counting the number of misplaced tiles and finding the sum of the taxicab distances between each block and its position in the goal configuration. Note that both are admissible. That is, they never overestimate the number of moves left, which ensures optimality for certain search algorithms such as A*. [1]



1.2 Search Algorithms

1.2.1 Depth First Search

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.

So the basic idea is to start from the root or any arbitrary node and mark the node and move to the adjacent unmarked node and continue this loop until there is no unmarked adjacent node. Then backtrack and check for other unmarked nodes and traverse them. Finally, print the nodes in the path.[2]

Listing 1.1: Depth First Search

```
def depthFirstSearch(problem):
    """Search the deepest nodes in the search tree first."""
    path = []
    visited = []
    myStack = util.Stack()
    myStack.push((problem.getStartState(), []))

    while not myStack.isEmpty():
        current, path = myStack.pop()
        if current not in visited:
            visited.append(current)

            if problem.isGoalState(current):
                return path

            successors = problem.getSuccessors(current)
            for nextLocation, nextDirection, cost in successors:
                if nextLocation not in visited:
                    myStack.push((nextLocation, path + [nextDirection]))

    util.raiseNotDefined()
```

1.2.2 Breadth First Search

Breadth-first search (BFS) is an algorithm for searching a tree data structure for a node that satisfies a given property. It starts at the tree root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level. Extra memory, usually a queue, is needed to keep track of the child nodes that were encountered but not yet explored. In contrast, (plain) depth-first search, which explores the node branch as far as possible before backtracking and expanding other nodes, may get lost in an infinite branch and never make it to the solution node.[3]

Listing 1.2: Breadth First Search

```
def breadthFirstSearch(problem):
    """Search the shallowest nodes in the search tree first."""
    path = []
    visited = []
    myQueue = util.Queue()
    myQueue.push((problem.getStartState(), []))

    while not myQueue.isEmpty():
        current, path = myQueue.pop()
        if current not in visited:
            visited.append(current)
            if problem.isGoalState(current):
                return path

            successors = problem.getSuccessors(current)

            for nextLocation, nextDirection, cost in successors:
                if nextLocation not in visited:
                    myQueue.push((nextLocation, path + [nextDirection]))

    util.raiseNotDefined()
```

1.2.3 Uniform Cost Search

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. This variant of Dijkstra is useful for infinite graphs and those graph which are too large to represent in the memory. Uniform-Cost Search is mainly used in Artificial Intelligence.[4]

Listing 1.3: Uniform Cost Search

```
def uniformCostSearch(problem):
    """Search the node of least total cost first."""
    path = []
    visited = []
    myPriorityQueue = util.PriorityQueue()
    myPriorityQueue.push((problem.getStartState(), [], 0), 0)

    while not myPriorityQueue.isEmpty():
        current, path, currentCost = myPriorityQueue.pop()
        if current not in visited:
            visited.append(current)

            if problem.isGoalState(current):
                return path

            successors = problem.getSuccessors(current)
            for nextLocation, nextDirection, cost in successors:
                if nextLocation not in visited:
                    priority = currentCost + cost
                    myPriorityQueue.push((nextLocation, path + [nextDirection],
                        priority), priority)

    util.raiseNotDefined()
```

1.2.4 A* Search

A* is a graph traversal and path search algorithm, which is 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.[5]

Listing 1.4: A* Search

```python
def aStarSearch(problem, heuristic=nullHeuristic):
    """Search the node that has the lowest combined cost and heuristic first."""
    path = []
    visited = []
    myPriorityQueue = util.PriorityQueue()
    myPriorityQueue.push((problem.getStartState(), [], 0), 0)

    while not myPriorityQueue.isEmpty():
        current, path, currentCost = myPriorityQueue.pop()
        if current not in visited:
            visited.append(current)

            if problem.isGoalState(current):
                return path

            successors = problem.getSuccessors(current)
            for nextLocation, nextDirection, cost in successors:
                if nextLocation not in visited:
                    newCost = currentCost + cost
                    heuristicCost = newCost + heuristic(nextLocation, problem)
                    myPriorityQueue.push((nextLocation, path + [nextDirection],
                        newCost), heuristicCost)

    util.raiseNotDefined()
```

1.3 Heuristics

1.3.1 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.[6]

Listing 1.5: Manhattan Heuristic

```
def eightPuzzle_manhattanHeuristic(state, problem, info={}):
    "The Manhattan distance heuristic for a EightPuzzleProblem"

    sum = 0
    arrayX = list(
        itertools.chain(*[([i] * problem.size) for i in range(problem.size)])) #
            [0, 0, 0, 1, 1, 1, 2, 2, 2] for size = 3
    arrayY = range(0, problem.size) * problem.size # [0, 1, 2, 0, 1, 2, 0, 1, 2]
        for size = 3

    for row in range(problem.size):
        for col in range(problem.size):
            sum = sum + util.manhattanDistance((row, col),
                                              (arrayX[state.cells[row][col]],
                                                arrayY[state.cells[row][col]]))
    return sum
```

1.3.2 Eucledian Heuristic

If your units can move at any angle (instead of grid directions), then you should probably use a straight line distance. However, if this is the case, then you may have trouble with using A* directly because the cost function g will not match the heuristic function h. Since Euclidean distance is shorter than Manhattan or diagonal distance, you will still get shortest paths, but A* will take longer to run.[6]

Listing 1.6: Euclidean Heuristic

```
def eightPuzzle_euclidHeuristic(state, problem, info={}):
    "The Euclid distance heuristic for a EightPuzzleProblem"

    sum = 0
    arrayX = list(
        itertools.chain(*[([i] * problem.size) for i in range(problem.size)])) #
            [0, 0, 0, 1, 1, 1, 2, 2, 2] for size = 3
    arrayY = range(0, problem.size) * problem.size # [0, 1, 2, 0, 1, 2, 0, 1, 2]
        for size = 3

    for row in range(problem.size):
        for col in range(problem.size):
            sum = sum + ((row - arrayX[state.cells[row][col]]) ** 2 + (
                    col - arrayY[state.cells[row][col]]) ** 2) ** 0.5
    return sum
```

1.3.3 Displaced Heuristic

This heuristic is based on the total number of tiles that are not in the correct position. Because of this, it is an efficient heuristic for the A* algorithm on the Eight Puzzle Problem. It works for grids that are bigger than 3x3 and finds the solution rapidly.

Listing 1.7: Displaced Heuristic

```
def eightPuzzle_displacedHeuristic(state, problem, info={}):
    notInPlace = 0
    goalPuzzle = EightPuzzleState(range(0, problem.size ** 2), problem.size)
    for row in range(problem.size):
        for col in range(problem.size):
            if state.cells[row][col] != goalPuzzle.cells[row][col]:
                notInPlace = notInPlace + 1
    return notInPlace
```

1.3.4 Manhattan + Euclidean Heuristic

The Manhattan plus Euclidean heuristic is a combination of the two. It does the sum of both heuristics, therefore obtaining a result based on the state of both of them.

Listing 1.8: Manhattan + Euclidean Heuristic

```
def eightPuzzle_euclidManhattanHeuristic(state, problem, info={}):
    return eightPuzzle_euclidHeuristic(state, problem) +
        eightPuzzle_manhattanHeuristic(state, problem)
```

1.3.5 Manhattan + Displaced Heuristic

Similar to the one before, this heuristic focuses on combining the Manhattan and Displaced heuristics by, again, getting the sum of them. It is again used to get the best path to the goal state of the game from the current state.

Listing 1.9: Manhattan + Displaced Heuristic

```
def eightPuzzle_displacedManhattanHeuristic(state, problem, info={}):
    return eightPuzzle_displacedHeuristic(state, problem) +
        eightPuzzle_manhattanHeuristic(state, problem)
```

1.3.6 Think Ahead Heuristic

The Think Ahead heuristic is using the current state of the game and the one after it. It calculates the heuristic values of the next state and always choose the minimum one for a traversal that has the lowest cost.

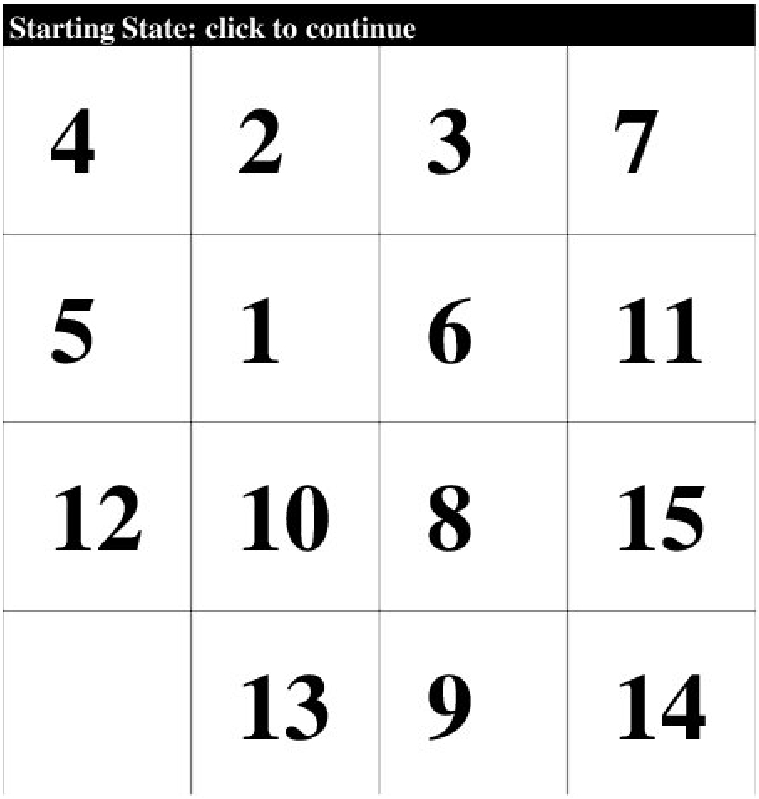
Listing 1.10: Think Ahead Heuristic

```
def eightPuzzle_thinkingAhead(state, problem, info={}):
    successors = problem.getSuccessors(state)
    h2 = 100000000000000000000
    for nextLocation, nextDirection, cost in successors:
        haux = eightPuzzle_euclidHeuristic(nextLocation, problem)
        if haux < h2:
            h2 = haux

    return eightPuzzle_euclidHeuristic(state, problem) + h2
```

1.4 Expanding the Eight Puzzle Framework

We have created an interface for the Eight Puzzle game. It not only works for the Eight version, but for every size of the game. You can create a game that runs 3x3, 4x4, 5x5 and so on. Also, the algorithms and heuristics implemented above are working for every version of the game, but the bigger the size, the slower the result will be calculated. The game is controlled by the mouse click, each mouse click moves to the next state of the application.



Starting State: click to continue

4	2	3	7
5	1	6	11
12	10	8	15
	13	9	14

Also, you can change the input in the command line with different arguments for the size, algorithm, random moves etc. Here is a list:

```
Usage:
       USAGE:      python eightpuzzle.py <options>
       EXAMPLES:   (1) python eightpuzzle.py
                       - creates an 8 puzzle game with a randomly generated state
                   (2) python eightpuzzle.py --size 4 --moves 100
                   OR  python eightpuzzle.py -s 4 --moves 100
                       - starts a 15 puzzle where and the position will be shuffled with


Options:
  -h, --help            show this help message and exit
  -s SIZE, --size=SIZE  The size of the puzzle (SIZE ** 2) [Default: 3]
  -t, --textGraphics    Display output as text only [Default: False]
  -a AGENT, --agent=AGENT
                        Select the agent [Default: 0]
  --width=WIDTH         Width of the graphics display [Default: 600]
  --height=HEIGHT       Height of the graphics display [Default: 600]
  --frames              Saves each puzzle state in ./frames [Default: False]
  --load=LOAD           Loads one of 6 (0-5) 8 puzzles instead of generating a
                        random one [Default: -1]
  --moves=MOVES         Shuffles the correct puzzle solution with MOVES legal
                        moves to create random puzzle [Default: 30]
```

Chapter 2

A2: Logics

2.1 Puzzle 1: A Trip to the Zoo [7]

Details

One day, five mothers each brought their only child to the zoo. The children had a glorious time together watching the different animals and eating their favorite snacks. The kids were so good, at the end of the day each mother let her child get one item from the souvenir shop as they were leaving the zoo. Can you determine the full name of each child, each child's favorite snack and animal, and the souvenir each brought home?

1. Julia, who loves cotton candy, didn't like the elephants. Mary didn't get a caramel apple. The child who got the stuffed animal liked the giraffes best.
2. Alan Rivera, the girl who liked the lions, and the child who got the activity set didn't want to leave the zoo.
3. Neither of the boys got fried dough, but one got nachos and the other one liked the monkeys best. Tom didn't get a poster.
4. The Gomez child almost got a coloring book with Mary but finally decided on a poster.
5. Tom, whose last name isn't Lozada, got a toy gun but didn't get a caramel apple. The Rodriguez child had fried dough.
6. Beth, who didn't like the giraffes or the elephants best, got an activity set.

Implementation

Listing 2.1: A Trip to the Zoo

```
set(arithmetic).
assign(domain_size, 5).
assign(max_models, −1).

 list(distinct).       % Objects in each list are distinct.
    [Alan, Beth, Julia, Mary, Tom].                              % first name
    [Rivera, Lozada, Gomez, Rodriguez, Gonzalez].                % last name
    [nachos, caramelApple, friedDough, cottonCandy, popcorn].    % snacks
    [giraffes, seals, lions, elephants, monkeys].                % animals
    [stuffedAnimal, activitySet, poster, coloringBook, toyGun].  % souvenir
end_of_list.

formulas(assumptions).

    pair(x, y) <−> x = y.
    pair(x,y) <-> pair (y,x).
    Alan < Beth & Beth < Julia & Julia < Mary & Mary < Tom.

    %The clues.
    %1
    pair(Julia, cottonCandy).
    -pair(Julia, elephants).
    -pair(Mary, caramelApple).
    pair(stuffedAnimal, giraffes).
    %2
    pair(Alan, Rivera).
    pair(lions, Beth) | pair(lions, Julia) | pair(lions, Mary).
    -pair(activitySet, lions).
    %3
    -pair (Alan, friedDough).
    -pair (Tom, friedDough).
    (pair (Alan, nachos) & pair(Tom, monkeys)) | (pair(Tom, nachos) & pair(Alan, monkeys)).
    -pair(Tom, poster).
    %4
    pair(Gomez, poster).
    pair(Mary, coloringBook).
    %5
    -pair(Tom, Lozada).
    pair(Tom, toyGun).
    -pair(Tom, caramelApple).
    pair(Rodriguez, friedDough).
    %6
    -pair(Beth, giraffes).
    -pair(Beth, elephants).
    pair(Beth, activitySet).

end_of_list.
```

Results

```
        function(Alan, [ 0 ]),
        function(Beth, [ 1 ]),
        function(Gomez, [ 2 ]),
        function(Julia, [ 2 ]),
        function(Lozada, [ 1 ]),
        function(Mary, [ 3 ]),
        function(Rivera, [ 0 ]),
        function(Rodriguez, [ 3 ]),
        function(Tom, [ 4 ]),
        function(activitySet, [ 1 ]),
        function(caramelApple, [ 1 ]),
        function(coloringBook, [ 3 ]),
        function(cottonCandy, [ 2 ]),
        function(elephants, [ 3 ]),
        function(friedDough, [ 3 ]),
        function(giraffes, [ 0 ]),
        function(lions, [ 2 ]),
        function(monkeys, [ 4 ]),
        function(nachos, [ 0 ]),
        function(poster, [ 2 ]),
        function(stuffedAnimal, [ 0 ]),
        function(toyGun, [ 4 ]),
        function(Gonzalez, [ 4 ]),
        function(popcorn, [ 4 ]),
        function(seals, [ 1 ])
Exiting with 1 model.
```

2.2 Puzzle 2: Mystery Number 8 [8]

Details

There is a ten digit mystery number (no leading 0), represented by ABCDEFGHIJ, where each numeral, 0 through 9, is used once. Given the following clues, what is the number?

1) If A > B, then C = 5 or 7, else C = 0 or 1.
2) If B > C, then D = 1 or 2, else D = 4 or 9.
3) If C > D, then E = 6 or 9, else E = 3 or 5.
4) If D > E, then F = 2 or 4, else F = 1 or 6.
5) If E > F, then G = 5 or 6, else G = 0 or 7.
6) If F > G, then H = 1 or 4, else H = 8 or 9.
7) If G > H, then I = 0 or 8, else I = 6 or 7.
8) If H > I, then J = 3 or 8, else J = 2 or 5.
9) If I > J, then A = 3 or 7, else A = 4 or 8.
10) If J > A, then B = 0 or 9, else B = 2 or 3.

Implementation

Listing 2.2: Mystery Number 8

```
set(arithmetic).
assign(domain_size, 10).
assign(max_models, -1).

 list(distinct).        % Objects in each list are distinct.
   [A, B, C, D, E, F, G, H, I, J].
end_of_list.

formulas(assumptions).


    %The clues.
    A > B -> C = 5 | C = 7.
    A < B -> C = 0 | C = 1.
    B > C -> D = 1 | D = 7.
    B < C -> D = 4 | D = 9.
    C > D -> E = 6 | E = 9.
    C < D -> E = 3 | E = 5.
    D > E -> F = 2 | F = 4.
    D < E -> F = 1 | F = 6.
    E > F -> G = 5 | G = 6.
    E < F -> G = 0 | G = 7.
    F > G -> H = 1 | H = 4.
    F < G -> H = 8 | H = 9.
    G > H -> I = 0 | I = 8.
    G < H -> I = 6 | I = 7.
    H > I -> J = 3 | J = 8.
    G < I -> J = 2 | J = 5.
    I > J -> A = 3 | A = 7.
    I < J -> A = 4 | A = 8.
    J > A -> B = 0 | B = 9.
    J < A -> B = 2| B = 3.
end_of_list.
```

Results

```
        function(A, [ 8 ]),
        function(B, [ 2 ]),
        function(C, [ 7 ]),
        function(D, [ 4 ]),
        function(E, [ 9 ]),
        function(F, [ 6 ]),
        function(G, [ 5 ]),
        function(H, [ 1 ]),
        function(I, [ 0 ]),
        function(J, [ 3 ])
Exiting with 1 model.
```

2.3 Puzzle 3: Beethoven's wig [9]

Details

Someone has stolen Beethoven's Wig and has put it in one of four locked boxes. The boxes are numbered from 1,2,3,4 in that order. There are four different keys that each has their own color. Use the clues below to figure out which key goes in which box and to find the box where Beethoven's wig is being kept.

1. The green key goes to the third or fourth box
2. The wig is to the left of the fourth box
3. The wig is to the right of the first box
4. The yellow key is to the left of the wig
5. The blue key is to the right of the yellow key and to the left of the green key
6. The red key goes to the first box

Implementation

Listing 2.3: Beethoven's wig

```
set(arithmetic).
assign(domain_size, 5).
assign(max_models, -1).

 list ( distinct ).        % Objects in each list are distinct .
    [0, Box1, Box2, Box3, Box4]. % the boxes
    [0, Green, Yellow, Red, Blue]. % the keys
end_of_list.

```

```
formulas(assumptions).

    opens(x, y) <-> x = y.
    left (x, y) <-> x < y.
    right (x, y) <-> x > y.
    Box1 < Box2 & Box2 < Box3 & Box3 < Box4.
    theWig = Box1 | theWig = Box2 | theWig = Box3 | theWig = Box4.

    % The clues.
    opens(Green, Box3) | opens(Green, Box4).
    left (theWig, Box4).
    right (theWig, Box1).
    left (Yellow, theWig).
    right (Blue, Yellow) & left (Blue, Green).
    opens(Red, Box1).
end_of_list.
```

Results

```
        function(Blue, [ 3 ]),
        function(Box1, [ 1 ]),
        function(Box2, [ 2 ]),
        function(Box3, [ 3 ]),
        function(Box4, [ 4 ]),
        function(Green, [ 4 ]),
        function(Red, [ 1 ]),
        function(Yellow, [ 2 ]),
        function(theWig, [ 3 ]),
Exiting with 1 model.
```

2.4 Puzzle 4: Hare and Tortoise [10]

Details

Haretown and Tortoiseville are 27 miles apart. A hare travels at 7 miles per hour from Haretown to Tortoiseville, while a tortoise travels at 2 miles per hour from Tortoiseville to Haretown. If both set out at the same time, how many miles will the hare have to travel before meeting the tortoise en route?

Implementation

Listing 2.4: Hare and Tortoise

```
set(arithmetic).
assign(max_models, -1).
assign(domain_size, 100).

formulas(demodulators).
    Haretown = 0.
    Tortoisevillage = 27.
    HareSpeed = 7.
    TortoiseSpeed = 2.
end_of_list.

```

```
formulas(assumptions).
    combinedSpeed = HareSpeed + TortoiseSpeed.
    timeToMeet = (Tortoisevillage +(−Haretown)) / combinedSpeed.
    hareMiles = HareSpeed * timeToMeet.
end_of_list.
```

Results

```
        function(HareSpeed, [ 7 ]),
        function(Haretown, [ 0 ]),
        function(TortoiseSpeed, [ 2 ]),
        function(Tortoisevillage, [27 ]),
        function(combinedSpeed, [ 9 ]),
        function(hareMiles, [21 ]),
        function(timeToMeet, [ 3 ])
Exiting with 1 model.
```

2.5 Puzzle 5: How old is Hannah [11]

Details

Fiona is 4 years old. Hannah is 4 times as old as Sasha. Sasha is 5 years older than Fiona's cousin Andrew, who is 1 year older than Nick, Fiona's twin brother. How old is Hannah?

Implementation

Listing 2.5: How old is Hannah

```
set(arithmetic).
assign(max_models, −1).
assign(domain_size, 100).

formulas(demodulators).
    Fiona = 4.
end_of_list.

formulas(assumptions).
    twin(x, y) <−> x = y.

    % The clues.
    Hannah = Sasha * 4.
    Sasha = 5 + Andrew.
    Andrew = Nick + 1.
    twin(Fiona, Nick).
end_of_list.
```

Results

```
        function(Andrew, [ 5 ]),
        function(Fiona, [ 4 ]),
        function(Hannah, [40 ]),
        function(Nick, [ 4 ]),
        function(Sasha, [10 ]),
Exiting with 1 model.
```

2.6 Puzzle 6: Mystery Number 6 [12]

Details

There is a ten-digit mystery number (no leading 0), represented by ABCDEFGHIJ, where each numeral, 0 through 9, is used once. Given the following clues, what is the number?

1) Digit A is either a square number or a triangle number, but not both.
2) Digit B is either an even number or a cube number, but not both.
3) Digit C is either a cube number or a triangle number, but not both.
4) Digit D is either an odd number or a square number, but not both.
5) Digit E is either an odd number or a cube number, but not both.
6) Digit F is either an odd number or a triangle number, but not both.
7) Digit G is either an odd number or a prime number, but not both.
8) Digit H is either an even number or a square number, but not both.
9) Digit I is either a square number or a cube number, but not both.
10) Digit J is either a prime number or a triangle number, but not both.
11) A < B, C < D, E < F, G < H, I < J
12) A + B + C + D + E < F + G + H + I + J

Implementation

Listing 2.6: Mystery Number 6

```
set(arithmetic).
assign(domain_size, 10).
assign(max_models, −1).

list(distinct).        % Objects in each list are distinct.
   [A, B, C, D, E, F, G, H, I, J].
end_of_list.

formulas(utils).
    odd(x) <−> x mod 2 = 1.
    even(x) <−> −odd(x).
    prime(x) <−> x = 2 | x = 3 | x = 5 | x = 7.
    cube(x) <−> x = 0 | x = 1 | x = 8 | x = 9.
    square(x) <−> x = 0 | x = 1 | x = 4 | x = 9.
    triangle(x) <−> x = 1 | x = 3 | x = 6.
end_of_list.

formulas(assumptions).
    A != 0.

    %The clues.
    (square(A) & −triangle(A)) | (−square(A) & triangle(A)).  %1
    (even(B) & −cube(B)) | (−even(B) & cube(B)).              %2
    (cube(C) & −triangle(C)) | (−cube(C) & triangle(C)).      %3
    (odd(D) & −square(D)) | (−odd(D) & square(D)).            %4
    (odd(E) & −cube(E)) | (−odd(E) & cube(E)).                %5
    (odd(F) & −triangle(F)) | (−odd(F) & triangle(F)).        %6
    (odd(G) & −prime(G)) | (−odd(G) & prime(G)).          %7
    (even(H) & −square(H)) | (−even(H) & square(H)).      %8
    (square(I) & −cube(I)) | (−square(I) & cube(I)).      %9
    (prime(J) & −triangle(J)) | (−prime(J) & triangle(J)).   %10
    %11
    A < B & C < D & E < F & G < H & I < J.
    %12
    A + B + C + D + E < F + G + H + I + J.
end_of_list.
```

Results

```
        function(A, [ 3 ]),
        function(B, [ 6 ]),
        function(C, [ 0 ]),
        function(D, [ 5 ]),
        function(E, [ 8 ]),
        function(F, [ 9 ]),
        function(G, [ 1 ]),
        function(H, [ 2 ]),
        function(I, [ 4 ]),
        function(J, [ 7 ]),
        relation(cube(_), [ 1, 1, 0, 0, 0, 0, 0, 0, 1, 1 ]),
        relation(even(_), [ 1, 0, 1, 0, 1, 0, 1, 0, 1, 0 ]),
        relation(odd(_), [ 0, 1, 0, 1, 0, 1, 0, 1, 0, 1 ]),
        relation(prime(_), [ 0, 0, 1, 1, 0, 1, 0, 1, 0, 0 ]),
        relation(square(_), [ 1, 1, 0, 0, 1, 0, 0, 0, 0, 1 ]),
        relation(triangle(_), [ 0, 1, 0, 1, 0, 0, 1, 0, 0, 0 ])
Exiting with 1 model.
```

2.7 Puzzle 7: Snow White[13]

Details

Recently, Snow White's seven dwarfs met up with three of their friends and went to the cinema to see Bambi. From the clues below, can you determine the order in which they stood in the ticket queue?

Grumpy was in front of Dopey. Stumpy was behind Sneezy and Doc. Doc was in front of Droopy and Happy.
Sleepy was behind Stumpy, Smelly and Happy.
Happy was in front of Sleepy, Smelly and Bashful.
Bashful was behind Smelly, Droopy and Sleepy.
Sneezy was in front of Dopey. Smelly was in front of Grumpy, Stumpy and Sneezy.
Dopey was in front of Droopy.
Sleepy was in front of Grumpy and Bashful.
Dopey was behind Sneezy, Doc and Sleepy.
Stumpy was in front of Dopey. Smelly was behind Doc.

Implementation

Listing 2.7: Snow White

```
set(arithmetic).
assign(max_models, -1).
assign(domain_size, 10).

list(distinct).
    [Doc, Happy, Smelly, Sneezy, Stumpy, Sleepy, Grumpy, Dopey, Droopy, Bashful].
end_of_list.

formulas(assumptions).
    %Definitions
    in_front(x, y) <-> x < y.

    %Clues
    in_front(Grumpy, Dopey).
    -in_front(Stumpy, Sneezy).
    -in_front(Stumpy, Doc).
    in_front(Doc, Droopy).
    in_front(Doc, Happy).
    -in_front(Sleepy, Stumpy).
    -in_front(Sleepy, Smelly).
    -in_front(Sleepy, Happy).
    in_front(Happy, Sleepy).
    in_front(Happy, Smelly).
    in_front(Happy, Bashful).
    -in_front(Bashful, Smelly).
    -in_front(Bashful, Droopy).
    -in_front(Bashful, Sleepy).
    in_front(Sneezy, Dopey).
    in_front(Smelly, Grumpy).
    in_front(Smelly, Stumpy).
    in_front(Smelly, Sneezy).
    in_front(Dopey, Droopy).
    in_front(Sleepy, Grumpy).
    in_front(Sleepy, Bashful).
    -in_front(Dopey, Sneezy).
    -in_front(Dopey, Doc).
    -in_front(Dopey, Sleepy).
    in_front(Stumpy, Dopey).
    -in_front(Smelly, Doc).
end_of_list.
```

Results

```
        function(Bashful, [ 9 ]),
        function(Doc, [ 0 ]),
        function(Dopey, [ 7 ]),
        function(Droopy, [ 8 ]),
        function(Grumpy, [ 6 ]),
        function(Happy, [ 1 ]),
        function(Sleepy, [ 5 ]),
        function(Smelly, [ 2 ]),
        function(Sneezy, [ 3 ]),
        function(Stumpy, [ 4 ]),
Exiting with 1 model.
```

2.8 Puzzle 8: Einstein's Riddle[14]

Details

Einstein's Riddle: Einstein wrote the following riddle. He said that 98% of the world could not solve it. But several NIEHS scientists were able to solve it, and they said it's not all that hard if you pay attention and are very patient. Give it a try:

There are 5 houses in 5 different colors in a row. In each house lives a person with a different nationality. The 5 owners drink a certain type of beverage, smoke a certain brand of cigar, and keep a certain pet. No owners have the same pet, smoke the same brand of cigar, or drink the same beverage. Other facts:

1. The Brit lives in the red house.
2. The Swede keeps dogs as pets.
3. The Dane drinks tea.
4. The green house is on the immediate left of the white house.
5. The green house's owner drinks coffee.
6. The owner who smokes Pall Mall rears birds.
7. The owner of the yellow house smokes Dunhill.
8. The owner living in the center house drinks milk.
9. The Norwegian lives in the first house.
10. The owner who smokes Blends lives next to the one who keeps cats.
11. The owner who keeps the horse lives next to the one who smokes Dunhill.
12. The owner who smokes Bluemasters drinks beer.
13. The German smokes Prince.
14. The Norwegian lives next to the blue house.
15. The owner who smokes Blends lives next to the one who drinks water.

Implementation

Listing 2.8: Einstein's Riddle

```
set(arithmetic).
assign(max_models, -1).
assign(domain_size, 5).

 list (distinct).
    [_yellow, blue, red, green, _white].
    [norwegian, dane, brit, german, swede].
    [_water, tea, milk, coffee, beer].
    [cats, horse, birds, fish, dogs].
    [dunhill, blends, pall_malls, prince, bluemasters].
end_of_list.

formulas(assumptions).
    %Definitions
    right_neighbor(x,y) <-> x+1 = y.
        neighbors(x,y) <-> right_neighbor(x,y) | right_neighbor(y,x).

    %Clues
     brit  = red.
    swede = dogs.
    dane = tea.
    right_neighbor(green, _white).
    green = coffee.
    pall_malls = birds.
    _yellow = dunhill.
    milk = 2.
    norwegian = 0.
    neighbors(blends, cats).
    neighbors(horse, dunhill).
    bluemasters = beer.
    german = prince.
    neighbors(norwegian, blue).
    neighbors(blends, _water).
end_of_list.
```

Results

```
        function(_water, [ 0 ]),
        function(_white, [ 4 ]),
        function(_yellow, [ 0 ]),
        function(beer, [ 4 ]),
        function(birds, [ 2 ]),
        function(blends, [ 1 ]),
        function(blue, [ 1 ]),
        function(bluemasters, [ 4 ]),
        function(brit, [ 2 ]),
        function(cats, [ 0 ]),
        function(coffee, [ 3 ]),
        function(dane, [ 1 ]),
        function(dogs, [ 4 ]),
        function(dunhill, [ 0 ]),
        function(german, [ 3 ]),
        function(green, [ 3 ]),
        function(horse, [ 1 ]),
        function(milk, [ 2 ]),
        function(norwegian, [ 0 ]),
        function(pall_malls, [ 2 ]),
        function(prince, [ 3 ]),
        function(red, [ 2 ]),
        function(swede, [ 4 ]),
        function(tea, [ 1 ]),
Exiting with 1 model.
```

2.9 Puzzle 9: The Father of Algebra[15]

Details

Diophantus was a Greek mathematician who lived in the third century. He was one of the first mathematicians to use algebraic symbols.
Most of what is known about Diophantus's life comes from an algebraic riddle from around the early sixth century. The riddle states:

Diophantus's youth lasted one sixth of his life. He grew a beard after one twelfth more. After one seventh more of his life, he married. 5 years later, he and his wife had a son. The son lived exactly one half as long as his father, and Diophantus died four years after his son.
How many years did Diophantus live?

Implementation

Listing 2.9: The Father of Algebra

```
set(arithmetic).
assign(domain_size, 200).
assign(max_models, -1).

formulas(assumptions).

    %The clues.
    (d/6 + d/12 + d/7 + 5 + d/2 + 4) = d.
    d mod 6 = 0.
    d mod 12 = 0.
    d mod 7 = 0.
    d mod 2 = 0.
end_of_list.
```

Results

```
        function(d, [84 ])
Exiting with 1 model.
```

2.10 Puzzle 10: Inspector Beethoven[16]

Details

Handel has been killed and Beethoven is on the case. He has interviewed the four suspects and their statements are shown below. Each suspect has said two sentences. One sentence of each suspect is a lie and one sentence is the truth. Help Beethoven figure out who the killer is.

Joplin: I did not kill Handel. Either Grieg is the killer or none of us is.
Grieg: I did not kill Handel. Gershwin is the killer.
Strauss: I did not kill Handel. Grieg is lying when he says Gershwin is the killer.
Gershwin: I did not kill Handel. If Joplin did not kill him, then Grieg did.

Implementation

Listing 2.10: Inspector Beethoven

```
assign(max_models, −1).
assign(domain_size, 2).

formulas(assumptions).
    J1 <−> −J2.
    Gr1 <−> −Gr2.
    S1 <−> −S2.
    Ge1 <−> −Ge2.


    J1 <−> −J.
    J2 <−> Gr | (−J & −Gr & −S & −Ge).

    Gr1 <−> −Gr.
    Gr2 <−> Ge.

    S1 <−> −S.
    S2 <−> −Gr2.

    Ge1 <−> −Ge.
    Ge2 <−> −J & Gr.

end_of_list.
```

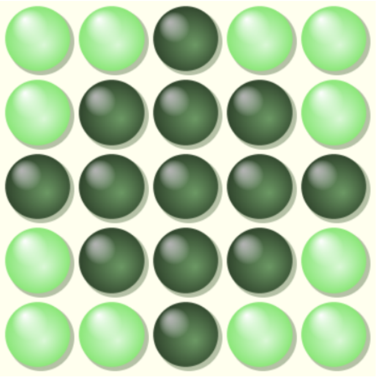
Results

```
        relation(Ge, [ 0 ]),
        relation(Ge1, [ 1 ]),
        relation(Ge2, [ 0 ]),
        relation(Gr, [ 0 ]),
        relation(Gr1, [ 1 ]),
        relation(Gr2, [ 0 ]),
        relation(J, [ 0 ]),
        relation(J1, [ 1 ]),
        relation(J2, [ 0 ]),
        relation(S, [ 1 ]),
        relation(S1, [ 0 ]),
        relation(S2, [ 1 ])
Exiting with 1 model.
```

Chapter 3

A3: Planning

3.1 Introduction



The object of the game is to turn each light on the board out. By clicking a piece on the board you will reverse the state for that piece and every adjacent piece. If they were "on" they'll switch to "off". To win you just need to make sure the entire board is ALLOUT. As another function to the game, we added the same idea but reversed, to turn all the lights on. The application is able to get as input a set of lights that are either on or off and resolve the puzzle

3.2 Implementation

When we want to change the state of the current LED we need to change the state of the adjacent LEDs but not their neighbours. In order to do that when we switch the current LED we wait for them by setting a few flags on wait then tuning them back after the adjacent LEDs finished.

For the case when the LED is on the edge of the board we made a frame of LEDs, that can't be pressed on, around it.

To switch the neighbours we check that the position next to is pressed, and that it's flag is being waited for. After that if the LED is on we turn it of otherwise if it's off we turn it on and set it's flag to not being waited for.

Listing 3.1: Domain

```
(define (domain allout)
    (:requirements :adl)

    (:types
        flag
        notpressable
        pressable
    )

    (:predicates
        (on ?row ?col)
        (next-row ?r1 ?r2)
        (next-column ?c1 ?c2)
        (pressed ?r ?c)
        (wait-for ?f)
    )

    (:action switch_neighbour_up
        :parameters (?row ?col ?down_row)
        :precondition (and
            (wait-for up)
            (next-row ?row ?down_row)
            (pressed ?down_row ?col)
        )
        :effect (and
            (when (on ?row ?col) (not (on ?row ?col)))
            (when (not (on ?row ?col)) (on ?row ?col))
            (not (wait-for up))
        )
    )

    (:action switch_neighbour_down
        :parameters (?row ?col ?up_row)
        :precondition (and
            (wait-for down)
            (next-row ?up_row ?row)
            (pressed ?up_row ?col)
        )
        :effect (and
            (when (on ?row ?col) (not (on ?row ?col)))
            (when (not (on ?row ?col)) (on ?row ?col))
            (not (wait-for down))
        )
    )

        (:action switch_neighbour_left
        :parameters (?row ?col ?right_col)
        :precondition (and
            (wait-for left)
            (next-column ?col ?right_col)
            (pressed ?row ?right_col)
        )
        :effect (and
            (when (on ?row ?col) (not (on ?row ?col)))
            (when (not (on ?row ?col)) (on ?row ?col))
            (not (wait-for left))
        )
    )

    (:action switch_neighbour_right
        :parameters (?row ?col ?left_col)
        :precondition (and
            (wait-for right)
            (next-column ?left_col ?col)
            (pressed ?row ?left_col)
        )
        :effect (and
            (when (on ?row ?col) (not (on ?row ?col)))
            (when (not (on ?row ?col)) (on ?row ?col))
            (not (wait-for right))
        )
    )


    (:action wait_for_neighbours
        :parameters (?row ?col)
        :precondition (and
            (pressed ?row ?col)
            (not (wait-for right))
            (not (wait-for left))
            (not (wait-for up))
            (not (wait-for down))
        )
        :effect (and
            (not (pressed ?row ?col))
            (not (wait-for mid))
        )
    )

    (:action switch_current--------------------------
        :parameters (?row - pressable ?col - pressable)
        :precondition (and
            (not (pressed ?row ?col))
            (not (wait-for mid))
        )
        :effect (and
            (when (on ?row ?col) (not (on ?row ?col)))
            (when (not (on ?row ?col)) (on ?row ?col))
            (pressed ?row ?col)
            (wait-for mid)
            (wait-for right)
            (wait-for left)
            (wait-for up)
            (wait-for down)
        )
    )
)
```

3.3 Results

We have made five different problems for the application to test. They are ranging from a 3x3 grid to a 5x5 grid. We also have a 3x3 grid that works the reversed way that the game was intended, because it turns on all the lights, instead of turning them off. The layouts are made so that the program can cover more cases of the problem. Below we have showcased the results to these five problems and how the program ran through the grid to accomplish its goal.

3.3.1 Problem 1

Listing 3.2: Problem 1

```
(define (problem allout-1)
    (:domain allout)

    (:objects
        mid up down left right - flag
        row0 row6 col0 col6 - notpressable
        row1 row2 row3 row4 row5 - pressable
        col1  col2  col3  col4  col5  - pressable)
    (: init
        (next-row row0 row1)          (next-column col0 col1)
        (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)
        (on row1 col1) (on row1 col2)     (on row1 col4) (on row1 col5)
        (on row2 col1)                                   (on row2 col5)

        (on row4 col1)                                   (on row4 col5)
        (on row5 col1) (on row5 col2)     (on row5 col4) (on row5 col5)
    )

    (:goal (and
            (not (on row1 col1)) (not (on row1 col2)) (not (on row1 col3)) (not (on row1 col4)) (not (on row1 col5))
            (not (on row2 col1)) (not (on row2 col2)) (not (on row2 col3)) (not (on row2 col4)) (not (on row2 col5))
            (not (on row3 col1)) (not (on row3 col2)) (not (on row3 col3)) (not (on row3 col4)) (not (on row3 col5))
            (not (on row4 col1)) (not (on row4 col2)) (not (on row4 col3)) (not (on row4 col4)) (not (on row4 col5))
            (not (on row5 col1)) (not (on row5 col2)) (not (on row5 col3)) (not (on row5 col4)) (not (on row5 col5))


            (not (pressed row1 col1)) (not (pressed row1 col2)) (not (pressed row1 col3)) (not (pressed row1 col4)) (not (pressed row1 col5))
            (not (pressed row2 col1)) (not (pressed row2 col2)) (not (pressed row2 col3)) (not (pressed row2 col4)) (not (pressed row2 col5))
            (not (pressed row3 col1)) (not (pressed row3 col2)) (not (pressed row3 col3)) (not (pressed row3 col4)) (not (pressed row3 col5))
            (not (pressed row4 col1)) (not (pressed row4 col2)) (not (pressed row4 col3)) (not (pressed row4 col4)) (not (pressed row4 col5))
            (not (pressed row5 col1)) (not (pressed row5 col2)) (not (pressed row5 col3)) (not (pressed row5 col4)) (not (pressed row5 col5))
        )
    )
)
```

```
step    0: SWITCH_CURRENT------------------------ ROW1 COL1
        1: SWITCH_NEIGHBOUR_LEFT ROW1 COL0 COL1
        2: SWITCH_NEIGHBOUR_UP ROW0 COL1 ROW1
        3: SWITCH_NEIGHBOUR_RIGHT ROW1 COL2 COL1
        4: SWITCH_NEIGHBOUR_DOWN ROW2 COL1 ROW1
        5: WAIT_FOR_NEIGHBOURS ROW1 COL1
        6: SWITCH_CURRENT------------------------ ROW1 COL5
        7: SWITCH_NEIGHBOUR_RIGHT ROW1 COL6 COL5
        8: SWITCH_NEIGHBOUR_UP ROW0 COL5 ROW1
        9: SWITCH_NEIGHBOUR_LEFT ROW1 COL4 COL5
       10: SWITCH_NEIGHBOUR_DOWN ROW2 COL5 ROW1
```

```
11: WAIT_FOR_NEIGHBOURS ROW1 COL5
12: SWITCH_CURRENT------------------------ ROW5 COL1
13: SWITCH_NEIGHBOUR_LEFT ROW5 COL0 COL1
14: SWITCH_NEIGHBOUR_DOWN ROW6 COL1 ROW5
15: SWITCH_NEIGHBOUR_UP ROW4 COL1 ROW5
16: SWITCH_NEIGHBOUR_RIGHT ROW5 COL2 COL1
17: WAIT_FOR_NEIGHBOURS ROW5 COL1
18: SWITCH_CURRENT------------------------ ROW5 COL5
19: SWITCH_NEIGHBOUR_RIGHT ROW5 COL6 COL5
20: SWITCH_NEIGHBOUR_DOWN ROW6 COL5 ROW5
21: SWITCH_NEIGHBOUR_UP ROW4 COL5 ROW5
22: SWITCH_NEIGHBOUR_LEFT ROW5 COL4 COL5
23: WAIT_FOR_NEIGHBOURS ROW5 COL5
```

3.3.2 Problem 2

Listing 3.3: Problem 2

```
(define (problem allout-1)
    (:domain allout)

    (:objects
        mid up down left right - flag
        row0 row5 col0 col5 - notpressable
        row1 row2 row3 row4 - pressable
        col1 col2 col3 col4 - pressable)
    (:init
        (next-row row0 row1)          (next-column col0 col1)
        (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)
        (on row3 col2) (on row3 col3) (on row3 col4)
        (on row4 col1) (on row4 col3)
    )

    (:goal (and
            (not (on row1 col1)) (not (on row1 col2)) (not (on row1 col3)) (not (on row1 col4))
            (not (on row2 col1)) (not (on row2 col2)) (not (on row2 col3)) (not (on row2 col4))
            (not (on row3 col1)) (not (on row3 col2)) (not (on row3 col3)) (not (on row3 col4))
            (not (on row4 col1)) (not (on row4 col2)) (not (on row4 col3)) (not (on row4 col4))


            (not (pressed row1 col1)) (not (pressed row1 col2)) (not (pressed row1 col3)) (not (pressed row1 col4))
            (not (pressed row2 col1)) (not (pressed row2 col2)) (not (pressed row2 col3)) (not (pressed row2 col4))
            (not (pressed row3 col1)) (not (pressed row3 col2)) (not (pressed row3 col3)) (not (pressed row3 col4))
            (not (pressed row4 col1)) (not (pressed row4 col2)) (not (pressed row4 col3)) (not (pressed row4 col4))
        )
    )
)
```

```
step    0: SWITCH_CURRENT------------------------ ROW3 COL3
        1: SWITCH_NEIGHBOUR_LEFT ROW3 COL2 COL3
        2: SWITCH_NEIGHBOUR_RIGHT ROW3 COL4 COL3
        3: SWITCH_NEIGHBOUR_DOWN ROW4 COL3 ROW3
        4: SWITCH_NEIGHBOUR_UP ROW2 COL3 ROW3
        5: WAIT_FOR_NEIGHBOURS ROW3 COL3
        6: SWITCH_CURRENT------------------------ ROW2 COL3
        7: SWITCH_NEIGHBOUR_RIGHT ROW2 COL4 COL3
        8: SWITCH_NEIGHBOUR_LEFT ROW2 COL2 COL3
        9: SWITCH_NEIGHBOUR_UP ROW1 COL3 ROW2
       10: SWITCH_NEIGHBOUR_DOWN ROW3 COL3 ROW2
       11: WAIT_FOR_NEIGHBOURS ROW2 COL3
       12: SWITCH_CURRENT------------------------ ROW2 COL2
       13: SWITCH_NEIGHBOUR_RIGHT ROW2 COL3 COL2
       14: SWITCH_NEIGHBOUR_LEFT ROW2 COL1 COL2
       15: SWITCH_NEIGHBOUR_UP ROW1 COL2 ROW2
       16: SWITCH_NEIGHBOUR_DOWN ROW3 COL2 ROW2
       17: WAIT_FOR_NEIGHBOURS ROW2 COL2
       18: SWITCH_CURRENT------------------------ ROW2 COL1
       19: SWITCH_NEIGHBOUR_LEFT ROW2 COL0 COL1
       20: SWITCH_NEIGHBOUR_RIGHT ROW2 COL2 COL1
       21: SWITCH_NEIGHBOUR_UP ROW1 COL1 ROW2
       22: SWITCH_NEIGHBOUR_DOWN ROW3 COL1 ROW2
       23: WAIT_FOR_NEIGHBOURS ROW2 COL1
       24: SWITCH_CURRENT------------------------ ROW1 COL1
       25: SWITCH_NEIGHBOUR_LEFT ROW1 COL0 COL1
       26: SWITCH_NEIGHBOUR_UP ROW0 COL1 ROW1
       27: SWITCH_NEIGHBOUR_DOWN ROW2 COL1 ROW1
       28: SWITCH_NEIGHBOUR_RIGHT ROW1 COL2 COL1
       29: WAIT_FOR_NEIGHBOURS ROW1 COL1
       30: SWITCH_CURRENT------------------------ ROW2 COL3
       31: SWITCH_NEIGHBOUR_UP ROW1 COL3 ROW2
       32: SWITCH_NEIGHBOUR_LEFT ROW2 COL2 COL3
       33: SWITCH_NEIGHBOUR_RIGHT ROW2 COL4 COL3
       34: SWITCH_NEIGHBOUR_DOWN ROW3 COL3 ROW2
       35: WAIT_FOR_NEIGHBOURS ROW2 COL3
       36: SWITCH_CURRENT------------------------ ROW3 COL1
       37: SWITCH_NEIGHBOUR_LEFT ROW3 COL0 COL1
       38: SWITCH_NEIGHBOUR_UP ROW2 COL1 ROW3
       39: SWITCH_NEIGHBOUR_RIGHT ROW3 COL2 COL1
       40: SWITCH_NEIGHBOUR_DOWN ROW4 COL1 ROW3
       41: WAIT_FOR_NEIGHBOURS ROW3 COL1
```

3.3.3 Problem 3

Listing 3.4: Problem 3

```
(define (problem allout−1)
    (:domain allout)

    (:objects
            mid up down left right − flag
            row0 row4 col0 col4 − notpressable
            row1 row2 row3 − pressable
            col1  col2  col3  − pressable)
    (: init
            (next−row row0 row1)          (next−column col0 col1)
            (next−row row1 row2)          (next−column col1 col2)
            (next−row row2 row3)          (next−column col2 col3)
            (next−row row3 row4)          (next−column col3 col4)
    )

    (:goal  (and
                    (on row1 col1)          (on row1 col2)          (on row1 col3)
                    (on row2 col1)   (not  (on row2 col2))          (on row2 col3)
                    (on row3 col1)          (on row3 col2)          (on row3 col3)


                (not  (pressed  row1  col1))  (not  (pressed  row1  col2))  (not  (pressed  row1  col3))
                (not  (pressed  row2  col1))  (not  (pressed  row2  col2))  (not  (pressed  row2  col3))
                (not  (pressed  row3  col1))  (not  (pressed  row3  col2))  (not  (pressed  row3  col3))
            )
    )
)
```

```
step    0: SWITCH_CURRENT------------------------ ROW1 COL1
        1: SWITCH_NEIGHBOUR_LEFT ROW1 COL0 COL1
        2: SWITCH_NEIGHBOUR_UP ROW0 COL1 ROW1
        3: SWITCH_NEIGHBOUR_RIGHT ROW1 COL2 COL1
        4: SWITCH_NEIGHBOUR_DOWN ROW2 COL1 ROW1
        5: WAIT_FOR_NEIGHBOURS ROW1 COL1
        6: SWITCH_CURRENT------------------------ ROW2 COL3
        7: SWITCH_NEIGHBOUR_RIGHT ROW2 COL4 COL3
        8: SWITCH_NEIGHBOUR_UP ROW1 COL3 ROW2
        9: SWITCH_NEIGHBOUR_DOWN ROW3 COL3 ROW2
       10: SWITCH_NEIGHBOUR_LEFT ROW2 COL2 COL3
       11: WAIT_FOR_NEIGHBOURS ROW2 COL3
       12: SWITCH_CURRENT------------------------ ROW3 COL2
       13: SWITCH_NEIGHBOUR_DOWN ROW4 COL2 ROW3
       14: SWITCH_NEIGHBOUR_UP ROW2 COL2 ROW3
       15: SWITCH_NEIGHBOUR_LEFT ROW3 COL1 COL2
       16: SWITCH_NEIGHBOUR_RIGHT ROW3 COL3 COL2
       17: WAIT_FOR_NEIGHBOURS ROW3 COL2
       18: SWITCH_CURRENT------------------------ ROW3 COL3
       19: SWITCH_NEIGHBOUR_RIGHT ROW3 COL4 COL3
       20: SWITCH_NEIGHBOUR_DOWN ROW4 COL3 ROW3
       21: SWITCH_NEIGHBOUR_LEFT ROW3 COL2 COL3
       22: SWITCH_NEIGHBOUR_UP ROW2 COL3 ROW3
       23: WAIT_FOR_NEIGHBOURS ROW3 COL3
       24: SWITCH_CURRENT------------------------ ROW2 COL3
       25: SWITCH_NEIGHBOUR_RIGHT ROW2 COL4 COL3
       26: SWITCH_NEIGHBOUR_LEFT ROW2 COL2 COL3
       27: SWITCH_NEIGHBOUR_UP ROW1 COL3 ROW2
       28: SWITCH_NEIGHBOUR_DOWN ROW3 COL3 ROW2
       29: WAIT_FOR_NEIGHBOURS ROW2 COL3
```

```
30: SWITCH_CURRENT------------------------ ROW1 COL3
31: SWITCH_NEIGHBOUR_RIGHT ROW1 COL4 COL3
32: SWITCH_NEIGHBOUR_UP ROW0 COL3 ROW1
33: SWITCH_NEIGHBOUR_LEFT ROW1 COL2 COL3
34: SWITCH_NEIGHBOUR_DOWN ROW2 COL3 ROW1
35: WAIT_FOR_NEIGHBOURS ROW1 COL3
36: SWITCH_CURRENT------------------------ ROW1 COL2
37: SWITCH_NEIGHBOUR_UP ROW0 COL2 ROW1
38: SWITCH_NEIGHBOUR_RIGHT ROW1 COL3 COL2
39: SWITCH_NEIGHBOUR_LEFT ROW1 COL1 COL2
40: SWITCH_NEIGHBOUR_DOWN ROW2 COL2 ROW1
41: WAIT_FOR_NEIGHBOURS ROW1 COL2
42: SWITCH_CURRENT------------------------ ROW1 COL1
43: SWITCH_NEIGHBOUR_LEFT ROW1 COL0 COL1
44: SWITCH_NEIGHBOUR_UP ROW0 COL1 ROW1
45: SWITCH_NEIGHBOUR_RIGHT ROW1 COL2 COL1
46: SWITCH_NEIGHBOUR_DOWN ROW2 COL1 ROW1
47: WAIT_FOR_NEIGHBOURS ROW1 COL1
48: SWITCH_CURRENT------------------------ ROW2 COL1
49: SWITCH_NEIGHBOUR_LEFT ROW2 COL0 COL1
50: SWITCH_NEIGHBOUR_RIGHT ROW2 COL2 COL1
51: SWITCH_NEIGHBOUR_UP ROW1 COL1 ROW2
52: SWITCH_NEIGHBOUR_DOWN ROW3 COL1 ROW2
53: WAIT_FOR_NEIGHBOURS ROW2 COL1
54: SWITCH_CURRENT------------------------ ROW1 COL1
55: SWITCH_NEIGHBOUR_LEFT ROW1 COL0 COL1
56: SWITCH_NEIGHBOUR_UP ROW0 COL1 ROW1
57: SWITCH_NEIGHBOUR_DOWN ROW2 COL1 ROW1
58: SWITCH_NEIGHBOUR_RIGHT ROW1 COL2 COL1
59: WAIT_FOR_NEIGHBOURS ROW1 COL1
60: SWITCH_CURRENT------------------------ ROW2 COL3
61: SWITCH_NEIGHBOUR_RIGHT ROW2 COL4 COL3
62: SWITCH_NEIGHBOUR_UP ROW1 COL3 ROW2
63: SWITCH_NEIGHBOUR_LEFT ROW2 COL2 COL3
64: SWITCH_NEIGHBOUR_DOWN ROW3 COL3 ROW2
65: WAIT_FOR_NEIGHBOURS ROW2 COL3
66: SWITCH_CURRENT------------------------ ROW3 COL1
67: SWITCH_NEIGHBOUR_LEFT ROW3 COL0 COL1
68: SWITCH_NEIGHBOUR_DOWN ROW4 COL1 ROW3
69: SWITCH_NEIGHBOUR_UP ROW2 COL1 ROW3
70: SWITCH_NEIGHBOUR_RIGHT ROW3 COL2 COL1
71: WAIT_FOR_NEIGHBOURS ROW3 COL1
```

3.3.4 Problem 4

Listing 3.5: Problem 4

```
(define (problem allout−1)
    (:domain allout)

    (:objects
        mid up down left right − flag
        row0 row6 col0 col6 − notpressable
        row1 row2 row3 row4 row5 − pressable
        col1  col2  col3  col4  col5  − pressable)
    (: init
        (next−row row0 row1)          (next−column col0 col1)
        (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)
        (on row1 col2)
        (on row2 col1) (on row2 col2) (on row2 col3)
        (on row3 col2) (on row3 col4)
        (on row4 col3) (on row4 col4) (on row4 col5)
        (on row5 col4)
    )

    (:goal (and
            (not (on row1 col1)) (not (on row1 col2)) (not (on row1 col3)) (not (on row1 col4)) (not (on row1 col5))
            (not (on row2 col1)) (not (on row2 col2)) (not (on row2 col3)) (not (on row2 col4)) (not (on row2 col5))
            (not (on row3 col1)) (not (on row3 col2)) (not (on row3 col3)) (not (on row3 col4)) (not (on row3 col5))
            (not (on row4 col1)) (not (on row4 col2)) (not (on row4 col3)) (not (on row4 col4)) (not (on row4 col5))
            (not (on row5 col1)) (not (on row5 col2)) (not (on row5 col3)) (not (on row5 col4)) (not (on row5 col5))


            (not (pressed row1 col1)) (not (pressed row1 col2)) (not (pressed row1 col3)) (not (pressed row1 col4)) (not (pressed row1 col5))
            (not (pressed row2 col1)) (not (pressed row2 col2)) (not (pressed row2 col3)) (not (pressed row2 col4)) (not (pressed row2 col5))
            (not (pressed row3 col1)) (not (pressed row3 col2)) (not (pressed row3 col3)) (not (pressed row3 col4)) (not (pressed row3 col5))
            (not (pressed row4 col1)) (not (pressed row4 col2)) (not (pressed row4 col3)) (not (pressed row4 col4)) (not (pressed row4 col5))
            (not (pressed row5 col1)) (not (pressed row5 col2)) (not (pressed row5 col3)) (not (pressed row5 col4)) (not (pressed row5 col5))
        )
    )
)
```

```
-------------------------------------------------
  0||0 --- SWITCH_CURRENT------------------------ ROW2 COL2 --- SON: 1||0
-------------------------------------------------
  1||0 --- SWITCH_NEIGHBOUR_RIGHT ROW2 COL3 COL2 --- SON: 2||0
-------------------------------------------------
  2||0 --- SWITCH_NEIGHBOUR_LEFT ROW2 COL1 COL2 --- SON: 3||0
-------------------------------------------------
  3||0 --- SWITCH_NEIGHBOUR_DOWN ROW3 COL2 ROW2 --- SON: 4||0
-------------------------------------------------
  4||0 --- SWITCH_NEIGHBOUR_UP ROW1 COL2 ROW2 --- SON: 5||0
-------------------------------------------------
  5||0 --- WAIT_FOR_NEIGHBOURS ROW2 COL2 --- SON: 6||0
```

```
-------------------------------------------------
  6||0 --- SWITCH_CURRENT------------------------ ROW4 COL4 --- SON: 7||0
-------------------------------------------------
  7||0 --- SWITCH_NEIGHBOUR_RIGHT ROW4 COL5 COL4 --- SON: 8||0
-------------------------------------------------
  8||0 --- SWITCH_NEIGHBOUR_LEFT ROW4 COL3 COL4 --- SON: 9||0
-------------------------------------------------
  9||0 --- SWITCH_NEIGHBOUR_DOWN ROW5 COL4 ROW4 --- SON: 10||0
-------------------------------------------------
 10||0 --- SWITCH_NEIGHBOUR_UP ROW3 COL4 ROW4 --- SON: 11||0
-------------------------------------------------
 11||0 --- WAIT_FOR_NEIGHBOURS ROW4 COL4 --- SON: 12||-1
-------------------------------------------------
```

3.3.5 Problem 5

Listing 3.6: Problem 5

```
(define (problem allout-1)
    (:domain allout)

    (:objects
        mid up down left right - flag
        row0 row6 col0 col6 - notpressable
        row1 row2 row3 row4 row5 - pressable
        col1 col2 col3 col4 col5 - pressable)
    (: init
        (next-row row0 row1)          (next-column col0 col1)
        (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)
        (on row1 col1) (on row1 col3) (on row1 col5)
        (on row2 col1) (on row2 col3) (on row2 col5)
        (on row4 col1) (on row4 col3) (on row4 col5)
        (on row5 col1) (on row5 col3) (on row5 col5)
    )

    (:goal (and
            (not (on row1 col1)) (not (on row1 col2)) (not (on row1 col3)) (not (on row1 col4)) (not (on row1 col5))
            (not (on row2 col1)) (not (on row2 col2)) (not (on row2 col3)) (not (on row2 col4)) (not (on row2 col5))
            (not (on row3 col1)) (not (on row3 col2)) (not (on row3 col3)) (not (on row3 col4)) (not (on row3 col5))
            (not (on row4 col1)) (not (on row4 col2)) (not (on row4 col3)) (not (on row4 col4)) (not (on row4 col5))
            (not (on row5 col1)) (not (on row5 col2)) (not (on row5 col3)) (not (on row5 col4)) (not (on row5 col5))


            (not (pressed row1 col1)) (not (pressed row1 col2)) (not (pressed row1 col3)) (not (pressed row1 col4)) (not (pressed row1 col5))
            (not (pressed row2 col1)) (not (pressed row2 col2)) (not (pressed row2 col3)) (not (pressed row2 col4)) (not (pressed row2 col5))
            (not (pressed row3 col1)) (not (pressed row3 col2)) (not (pressed row3 col3)) (not (pressed row3 col4)) (not (pressed row3 col5))
            (not (pressed row4 col1)) (not (pressed row4 col2)) (not (pressed row4 col3)) (not (pressed row4 col4)) (not (pressed row4 col5))
            (not (pressed row5 col1)) (not (pressed row5 col2)) (not (pressed row5 col3)) (not (pressed row5 col4)) (not (pressed row5 col5))
        )
    )
)
```

```
-------------------------------------------------
  0||0 --- SWITCH_CURRENT------------------------ ROW1 COL1 --- SON: 1||0
-------------------------------------------------
  1||0 --- SWITCH_NEIGHBOUR_LEFT ROW1 COL0 COL1 --- SON: 2||0
-------------------------------------------------
  2||0 --- SWITCH_NEIGHBOUR_DOWN ROW2 COL1 ROW1 --- SON: 3||0
-------------------------------------------------
  3||0 --- SWITCH_NEIGHBOUR_RIGHT ROW1 COL2 COL1 --- SON: 4||0
-------------------------------------------------
  4||0 --- SWITCH_NEIGHBOUR_UP ROW0 COL1 ROW1 --- SON: 5||0
-------------------------------------------------
  5||0 --- WAIT_FOR_NEIGHBOURS ROW1 COL1 --- SON: 6||0
-------------------------------------------------
  6||0 --- SWITCH_CURRENT------------------------ ROW1 COL3 --- SON: 7||0
-------------------------------------------------
  7||0 --- SWITCH_NEIGHBOUR_LEFT ROW1 COL2 COL3 --- SON: 8||0
-------------------------------------------------
  8||0 --- SWITCH_NEIGHBOUR_DOWN ROW2 COL3 ROW1 --- SON: 9||0
-------------------------------------------------
  9||0 --- SWITCH_NEIGHBOUR_RIGHT ROW1 COL4 COL3 --- SON: 10||0
-------------------------------------------------
 10||0 --- SWITCH_NEIGHBOUR_UP ROW0 COL3 ROW1 --- SON: 11||0
-------------------------------------------------
 11||0 --- WAIT_FOR_NEIGHBOURS ROW1 COL3 --- SON: 12||0
-------------------------------------------------
 12||0 --- SWITCH_CURRENT------------------------ ROW1 COL5 --- SON: 13||0
-------------------------------------------------
 13||0 --- SWITCH_NEIGHBOUR_RIGHT ROW1 COL6 COL5 --- SON: 14||0
-------------------------------------------------
 14||0 --- SWITCH_NEIGHBOUR_LEFT ROW1 COL4 COL5 --- SON: 15||0
-------------------------------------------------
 15||0 --- SWITCH_NEIGHBOUR_DOWN ROW2 COL5 ROW1 --- SON: 16||0
-------------------------------------------------
 16||0 --- SWITCH_NEIGHBOUR_UP ROW0 COL5 ROW1 --- SON: 17||0
-------------------------------------------------
 17||0 --- WAIT_FOR_NEIGHBOURS ROW1 COL5 --- SON: 18||0
-------------------------------------------------
 18||0 --- SWITCH_CURRENT------------------------ ROW5 COL5 --- SON: 19||0
-------------------------------------------------
 19||0 --- SWITCH_NEIGHBOUR_RIGHT ROW5 COL6 COL5 --- SON: 20||0
-------------------------------------------------
```

```
 20||0 --- SWITCH_NEIGHBOUR_DOWN ROW6 COL5 ROW5 --- SON: 21||0
-------------------------------------------------
 21||0 --- SWITCH_NEIGHBOUR_LEFT ROW5 COL4 COL5 --- SON: 22||0
-------------------------------------------------
 22||0 --- SWITCH_NEIGHBOUR_UP ROW4 COL5 ROW5 --- SON: 23||0
-------------------------------------------------
 23||0 --- WAIT_FOR_NEIGHBOURS ROW5 COL5 --- SON: 24||0
-------------------------------------------------
 24||0 --- SWITCH_CURRENT------------------------ ROW5 COL3 --- SON: 25||0
-------------------------------------------------
 25||0 --- SWITCH_NEIGHBOUR_RIGHT ROW5 COL4 COL3 --- SON: 26||0
-------------------------------------------------
 26||0 --- SWITCH_NEIGHBOUR_DOWN ROW6 COL3 ROW5 --- SON: 27||0
-------------------------------------------------
 27||0 --- SWITCH_NEIGHBOUR_LEFT ROW5 COL2 COL3 --- SON: 28||0
-------------------------------------------------
 28||0 --- SWITCH_NEIGHBOUR_UP ROW4 COL3 ROW5 --- SON: 29||0
-------------------------------------------------
 29||0 --- WAIT_FOR_NEIGHBOURS ROW5 COL3 --- SON: 30||0
-------------------------------------------------
 30||0 --- SWITCH_CURRENT------------------------ ROW5 COL1 --- SON: 31||0
-------------------------------------------------
 31||0 --- SWITCH_NEIGHBOUR_RIGHT ROW5 COL2 COL1 --- SON: 32||0
-------------------------------------------------
 32||0 --- SWITCH_NEIGHBOUR_LEFT ROW5 COL0 COL1 --- SON: 33||0
-------------------------------------------------
 33||0 --- SWITCH_NEIGHBOUR_DOWN ROW6 COL1 ROW5 --- SON: 34||0
-------------------------------------------------
 34||0 --- SWITCH_NEIGHBOUR_UP ROW4 COL1 ROW5 --- SON: 35||0
-------------------------------------------------
 35||0 --- WAIT_FOR_NEIGHBOURS ROW5 COL1 --- SON: 36||-1
-------------------------------------------------
```

Bibliography

[1] https://en.wikipedia.org/wiki/15_puzzle *accessed on November 6, 2022*

[2] https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/ *accessed on November 6, 2022*

[3] https://en.wikipedia.org/wiki/Breadth-first_search *accessed on November 6, 2022*

[4] https://www.geeksforgeeks.org/uniform-cost-search-dijkstra-for-large-graphs/ *accessed on November 6, 2022*

[5] https://en.wikipedia.org/wiki/A*_search_algorithm *accessed on November 6, 2022*

[6] http://theory.stanford.edu/ amitp/GameProgramming/Heuristics.html *accessed on November 6, 2022*

Puzzles

[7] https://www.braingle.com/brainteasers/26603/a-trip-to-the-zoo.html *accessed on December 8, 2022*

[8] https://www.braingle.com/brainteasers/49152/mystery-number-8.html *accessed on December 8, 2022*

[9] https://www.braingle.com/brainteasers/23965/beethovens-wig.html *accessed on December 9, 2022*

[10] https://www.braingle.com/brainteasers/1471/hare-and-tortoise.html *accessed on December 9, 2022*

[11] https://www.braingle.com/brainteasers/45521/how-old-is-hannah.html *accessed on December 9, 2022*

[12] https://www.braingle.com/brainteasers/49152/mystery-number-8.html *accessed on December 9, 2022*

[13] https://www.braingle.com/brainteasers/23384/snow-white.html *accessed on December 9, 2022*

[14] https://www.braingle.com/brainteasers/602/einsteins-riddle.html *accessed on December 9, 2022*

[15] https://www.braingle.com/brainteasers/35386/the-father-of-algebra.html *accessed on December 9, 2022*

[16] https://www.braingle.com/brainteasers/28115/inspector-beethoven-iii.html *accessed on December 9, 2022*

Appendix A

Original code for A1

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.1 Heuristics

Listing A.1: Manhattan Heuristic

```
def eightPuzzle_manhattanHeuristic(state, problem, info={}):
    "The Manhattan distance heuristic for a EightPuzzleProblem"

    sum = 0
    arrayX = list(
        itertools.chain(*[([i] * problem.size) for i in range(problem.size)])) #
            [0, 0, 0, 1, 1, 1, 2, 2, 2] for size = 3
    arrayY = range(0, problem.size) * problem.size # [0, 1, 2, 0, 1, 2, 0, 1, 2]
        for size = 3

    for row in range(problem.size):
        for col in range(problem.size):
            sum = sum + util.manhattanDistance((row, col),
                                           (arrayX[state.cells[row][col]],
                                              arrayY[state.cells[row][col]]))
    return sum
```

Listing A.2: Euclidean Heuristic

```
def eightPuzzle_euclidHeuristic(state, problem, info={}):
    "The Euclid distance heuristic for a EightPuzzleProblem"

    sum = 0
    arrayX = list(
        itertools.chain(*[([i] * problem.size) for i in range(problem.size)])) #
            [0, 0, 0, 1, 1, 1, 2, 2, 2] for size = 3
    arrayY = range(0, problem.size) * problem.size # [0, 1, 2, 0, 1, 2, 0, 1, 2]
        for size = 3

    for row in range(problem.size):
        for col in range(problem.size):
            sum = sum + ((row - arrayX[state.cells[row][col]]) ** 2 + (
                    col - arrayY[state.cells[row][col]]) ** 2) ** 0.5
    return sum
```

Listing A.3: Displaced Heuristic

```
def eightPuzzle_displacedHeuristic(state, problem, info={}):
    notInPlace = 0
    goalPuzzle = EightPuzzleState(range(0, problem.size ** 2), problem.size)
    for row in range(problem.size):
        for col in range(problem.size):
            if state.cells[row][col] != goalPuzzle.cells[row][col]:
                notInPlace = notInPlace + 1
    return notInPlace
```

Listing A.4: Manhattan + Euclidean Heuristic

```
def eightPuzzle_euclidManhattanHeuristic(state, problem, info={}):
    return eightPuzzle_euclidHeuristic(state, problem) +
        eightPuzzle_manhattanHeuristic(state, problem)
```

Listing A.5: Manhattan + Displaced Heuristic

```
def eightPuzzle_displacedManhattanHeuristic(state, problem, info={}):
    return eightPuzzle_displacedHeuristic(state, problem) +
        eightPuzzle_manhattanHeuristic(state, problem)
```

Listing A.6: hink Ahead Heuristic

```
def eightPuzzle_thinkingAhead(state, problem, info={}):
    successors = problem.getSuccessors(state)
    h2 = 10000000000000000000
    for nextLocation, nextDirection, cost in successors:
        haux = eightPuzzle_euclidHeuristic(nextLocation, problem)
        if haux < h2:
            h2 = haux

    return eightPuzzle_euclidHeuristic(state, problem) + h2
```

A.2 Search Algorithms

Listing A.7: Depth First Search

```
def depthFirstSearch(problem):
    """Search the deepest nodes in the search tree first."""
    path = []
    visited = []
    myStack = util.Stack()
    myStack.push((problem.getStartState(), []))

    while not myStack.isEmpty():
        current, path = myStack.pop()
        if current not in visited:
            visited.append(current)

            if problem.isGoalState(current):
                return path

            successors = problem.getSuccessors(current)
            for nextLocation, nextDirection, cost in successors:
                if nextLocation not in visited:
                    myStack.push((nextLocation, path + [nextDirection]))

    util.raiseNotDefined()
```

Listing A.8: Breadth First Search

```
def breadthFirstSearch(problem):
    """Search the shallowest nodes in the search tree first."""
    path = []
    visited = []
    myQueue = util.Queue()
    myQueue.push((problem.getStartState(), []))

    while not myQueue.isEmpty():
        current, path = myQueue.pop()
        if current not in visited:
            visited.append(current)
            if problem.isGoalState(current):
                return path

            successors = problem.getSuccessors(current)

            for nextLocation, nextDirection, cost in successors:
                if nextLocation not in visited:
                    myQueue.push((nextLocation, path + [nextDirection]))

    util.raiseNotDefined()
```

Listing A.9: Uniform Cost Search

```
def uniformCostSearch(problem):
    """Search the node of least total cost first."""
    path = []
    visited = []
    myPriorityQueue = util.PriorityQueue()
    myPriorityQueue.push((problem.getStartState(), [], 0), 0)

    while not myPriorityQueue.isEmpty():
        current, path, currentCost = myPriorityQueue.pop()
        if current not in visited:
            visited.append(current)

            if problem.isGoalState(current):
                return path
            successors = problem.getSuccessors(current)
            for nextLocation, nextDirection, cost in successors:
                if nextLocation not in visited:
                    priority = currentCost + cost
                    myPriorityQueue.push((nextLocation, path + [nextDirection],
                        priority), priority)

    util.raiseNotDefined()
```

Listing A.10: A* Search

```
def aStarSearch(problem, heuristic=nullHeuristic):
    """Search the node that has the lowest combined cost and heuristic first."""
    path = []
    visited = []
    myPriorityQueue = util.PriorityQueue()
    myPriorityQueue.push((problem.getStartState(), [], 0), 0)

    while not myPriorityQueue.isEmpty():
        current, path, currentCost = myPriorityQueue.pop()
        if current not in visited:
            visited.append(current)

            if problem.isGoalState(current):
                return path

            successors = problem.getSuccessors(current)
            for nextLocation, nextDirection, cost in successors:
                if nextLocation not in visited:
                    newCost = currentCost + cost
                    heuristicCost = newCost + heuristic(nextLocation, problem)
                    myPriorityQueue.push((nextLocation, path + [nextDirection],
                        newCost), heuristicCost)

    util.raiseNotDefined()
```

A.3 Display

Listing A.11: Display

```
from graphicsUtils import *
from eightpuzzle import *

BACKGROUND_COLOR = formatColor(0, 0, 0)
PUZZLE_TEXT_COLOR = formatColor(0, 0, 0)
TEXT_COLOR = formatColor(1, 1, 1)
SQUARE_COLOR = formatColor(1, 1, 1)


class Graphics:

    def __init__(self, width=600, height=600, size=3):
       self.size = size
       self.messageBoxHeight = 30
       self.enableMessageBox = 1
       self.windowWidth = width
       self.windowHeight = height
       self.make_window(self.windowWidth, self.windowHeight +
           self.getMessageBoxHeight())

       self.drawSquares()

   def make_window(self, width, height):
       begin_graphics(width, height,
                      BACKGROUND_COLOR,
                     "AI: Eight Puzzle Problem")

   def getMessageBoxHeight(self):
       return self.messageBoxHeight * self.enableMessageBox

   def drawMessage(self, message, refreshGraphics=1, ):
       text((5, 5), TEXT_COLOR, message, "Times", 24, "bold")
       if refreshGraphics: refresh()

   def drawSquares(self, refreshGraphics=1):
       normalizedSize = min(self.windowHeight, self.windowWidth)
       startX = normalizedSize / (self.size * 2)
       startY = normalizedSize / (self.size * 2) + self.getMessageBoxHeight()
       position = [startX, startY]
       increment = normalizedSize / self.size
       for i in range(self.size):
           for j in range(self.size):
               square(self.to_screen(position), increment / 2 - 1, SQUARE_COLOR, 1)
               position[0] += increment
           position[1] += increment
           position[0] = startX
       if refreshGraphics: refresh()

   def drawState(self, state, refreshGraphics=1):
       normalizedSize = min(self.windowHeight, self.windowWidth)
       startX = normalizedSize / (self.size ** 2)
       startY = normalizedSize / (self.size ** 2) + self.getMessageBoxHeight()
       position = [startX, startY]
       increment = normalizedSize / self.size

       for i in range(self.size):
           for j in range(self.size):
               if state.cells[i][j] != 0:
                  text(self.to_screen(position), PUZZLE_TEXT_COLOR,
                      str(state.cells[i][j]), "Times", increment / 2, "bold")
               position[0] += increment
           position[1] += increment
           position[0] = startX
       if refreshGraphics: refresh()
   def updatePuzzleGraphics(self, state, message):
       clear_screen()
       self.drawSquares(0)
       self.drawState(state, 0)
       self.drawMessage(message, 0)
       refresh()

   def to_screen(self, point):
       return (point[0], point[1])

   def finish(self):
       end_graphics()

SAVE_POSTSCRIPT = True
POSTSCRIPT_OUTPUT_DIR = 'frames'
FRAME_NUMBER = 0

def saveFrame():
   "Saves the current graphical output as a postscript file"
   global SAVE_POSTSCRIPT, FRAME_NUMBER, POSTSCRIPT_OUTPUT_DIR
   if not SAVE_POSTSCRIPT: return
   if not os.path.exists(POSTSCRIPT_OUTPUT_DIR): os.mkdir(POSTSCRIPT_OUTPUT_DIR)
   name = os.path.join(POSTSCRIPT_OUTPUT_DIR, 'frame_%08d.ps' % FRAME_NUMBER)
   FRAME_NUMBER += 1
   writePostscript(name) # writes the current canvas
```

A.4 Game Initialization

Listing A.12: Rungame

```
def readCommand(argv):
    """
    Processes the command used to run eightpuzzle from the command line.
    """
    from optparse import OptionParser
    usageStr = """
       USAGE:     python eightpuzzle.py <options>
       EXAMPLES:  (1) python eightpuzzle.py
                      - creates an 8 puzzle game with a randomly generated state
                  (2) python eightpuzzle.py --size 4 --moves 100
                  OR  python eightpuzzle.py -s 4 --moves 100
                      - starts a 15 puzzle where and the position will be shuffled
                          with 100 legal moves
       """
    parser = OptionParser(usageStr)

    parser.add_option('-s', '--size', type='int', dest='size',
                      help=default('The size of the puzzle (SIZE ** 2)'),
                      default=3)
    parser.add_option('-t', '--textGraphics', action='store_true',
        dest='textGraphics',
                      help=default('Display output as text only'),
                      default=False)
    parser.add_option(’-a’, ’--agent’, type=’int’, dest=’agent’,
                      help=default(’Select the agent’),
                      default=0)
    parser.add_option(’--width’, type=’int’, dest=’width’,
                      help=default(’Width of the graphics display’),
                      default=600)
    parser.add_option(’--height’, type=’int’, dest=’height’,
                      help=default(’Height of the graphics display’),
                      default=600)
    parser.add_option(’--frames’, action=’store_true’, dest=’frames’,
                      help=default(’Saves each puzzle state in ./frames’),
                      default=False)
    parser.add_option(’--load’, type=’int’, dest=’load’,
                      help=default(’Loads one of 6 (0-5) 8 puzzles instead of
                          generating a random one’),
                      default=-1)
    parser.add_option(’--moves’, type=’int’, dest=’moves’,
                      help=default(’Shuffles the correct puzzle solution with MOVES
                          legal moves to create random puzzle’),
                      default=30)
    options, otherjunk = parser.parse_args(argv)
    if len(otherjunk) != 0:
        raise Exception(’Command line input not understood: ’ + str(otherjunk))
    args = dict()

    if options.size < 2: parser.error(’Size must be > 1’)
    if options.width > 7680: parser.error(’Try a width <= 7680’)
    if options.width < 300: parser.error(’Try a width >= 300’)
    if options.height > 4320: parser.error(’Try a height <= 4320’)
    if options.height < 300: parser.error(’Try a height >= 300’)
    if options.moves < 0: parser.error(’The number of moves should be positive’)
    if options.size != 3 and options.load >= 0: parser.error(’size must be 3 to use
        the stored puzzles’)
    if options.load > 5: parser.error(’There are 6 puzzle stored, numbered from 0
        to 5’)
    if options.agent < 0: parser.error(’Agent is a value between 0 and 8’)
    if options.agent > 8: parser.error(’Agent is a value between 0 and 8’)

    args[’size’] = options.size
    args[’width’] = options.width
    args[’height’] = options.height
    args[’frames’] = options.frames
    args[’textGraphics’] = options.textGraphics
    args[’load’] = options.load
    args[’moves’] = options.moves
    args[’agent’] = options.agent

    return args


def runGame(size, width, height, frames, textGraphics, load, moves, agent):
    # create or load a puzzle
    if load >= 0: puzzle = loadEightPuzzle(load)
    else: puzzle = createRandomEightPuzzle(moves, size)

    # initialize display
    if not textGraphics:
        display = eightPuzzleDisplay.Graphics(width, height, size)
        display.updatePuzzleGraphics(puzzle, "Starting State: click to continue")

    # saves the current state in ./frame as a ps file
    if frames: eightPuzzleDisplay.saveFrame()

    # find the solution to the puzzle
    problem = EightPuzzleSearchProblem(puzzle, size)
    # path = search.aStarSearch(problem, problem.eightPuzzle_euclidHeuristic)

    path = eightPuzzleAgents.EightPuzzleAgent(problem, agent).searchFunction

    print(’The algorithm found a path of %d moves: %s’ % (len(path), str(path)))

    i = 1
    if textGraphics: raw_input("Press return for the next state...") # wait for key
        stroke
    else: graphicsUtils.wait_for_click() # wait for click
    for a in path:
        puzzle = puzzle.result(a)
        s = (’After %d move%s: %s’ % (i, ("", "s")[i > 1], a))

        if textGraphics:
            print s
            print(puzzle)
            raw_input("Press return for the next state...") # wait for key stroke
        else:
            display.updatePuzzleGraphics(puzzle, s)
            graphicsUtils.wait_for_click() # wait for click

        if frames: eightPuzzleDisplay.saveFrame()
        i += 1


if __name__ == ’__main__’:
    """
    The main function called when eightpuzzle.py is run
    from the command line:
    > python eightpuzzle.py

    See the usage string for more details.
    > python eightpuzzle.py --help
    """
    args = readCommand(sys.argv[1:]) # Get game components based on input
    runGame(**args)
```

Appendix B

Original code for A2

Listing B.1: A Trip to the Zoo

```
set(arithmetic).
assign(domain_size, 5).
assign(max_models, −1).

list(distinct).      % Objects in each list are distinct.
   [Alan, Beth, Julia, Mary, Tom].                                 % first name
   [Rivera, Lozada, Gomez, Rodriguez, Gonzalez].                   % last name
   [nachos, caramelApple, friedDough, cottonCandy, popcorn].       % snacks
   [giraffes, seals, lions, elephants, monkeys].                   % animals
   [stuffedAnimal, activitySet, poster, coloringBook, toyGun].     % souvenir
end_of_list.

formulas(assumptions).

    pair(x, y) <−> x = y.
    pair(x,y) <−> pair (y,x).
    Alan < Beth & Beth < Julia & Julia < Mary & Mary < Tom.

    %The clues.
    %1
    pair(Julia, cottonCandy).
    −pair(Julia, elephants).
    −pair(Mary, caramelApple).
    pair(stuffedAnimal, giraffes).
    %2
    pair(Alan, Rivera).
    pair(lions, Beth) | pair(lions, Julia) | pair(lions, Mary).
    −pair(activitySet, lions).
    %3
    −pair (Alan, friedDough).
    −pair (Tom, friedDough).
    (pair (Alan, nachos) & pair(Tom, monkeys)) | (pair(Tom, nachos) & pair(Alan, monkeys)).
    −pair(Tom, poster).
    %4
    pair(Gomez, poster).
    pair(Mary, coloringBook).
    %5
    −pair(Tom, Lozada).
    pair(Tom, toyGun).
    −pair(Tom, caramelApple).
    pair(Rodriguez, friedDough).
    %6
    −pair(Beth, giraffes).
    -pair(Beth, elephants).
    pair(Beth, activitySet).

end_of_list.
```

Listing B.2: Mystery Number 8

```
set(arithmetic).
assign(domain_size, 10).
assign(max_models, -1).

list(distinct).        % Objects in each list are distinct.
   [A, B, C, D, E, F, G, H, I, J].
end_of_list.

formulas(assumptions).


    %The clues.
    A > B -> C = 5 | C = 7.
    A < B -> C = 0 | C = 1.
    B > C -> D = 1 | D = 7.
    B < C -> D = 4 | D = 9.
    C > D -> E = 6 | E = 9.
    C < D -> E = 3 | E = 5.
    D > E -> F = 2 | F = 4.
    D < E -> F = 1 | F = 6.
    E > F -> G = 5 | G = 6.
    E < F -> G = 0 | G = 7.
    F > G -> H = 1 | H = 4.
    F < G -> H = 8 | H = 9.
    G > H -> I = 0 | I = 8.
    G < H -> I = 6 | I = 7.
    H > I -> J = 3 | J = 8.
    G < I -> J = 2 | J = 5.
    I > J -> A = 3 | A = 7.
    I < J -> A = 4 | A = 8.
    J > A -> B = 0 | B = 9.
    J < A -> B = 2| B = 3.
end_of_list.
```

Listing B.3: Beethoven’s wig

```
set(arithmetic).
assign(domain_size, 5).
assign(max_models, -1).

list(distinct).        % Objects in each list are distinct.
    [0, Box1, Box2, Box3, Box4]. % the boxes
    [0, Green, Yellow, Red, Blue]. % the keys
end_of_list.

formulas(assumptions).

    opens(x, y) <-> x = y.
    left(x, y) <-> x < y.
    right(x, y) <-> x > y.
    Box1 < Box2 & Box2 < Box3 & Box3 < Box4.
    theWig = Box1 | theWig = Box2 | theWig = Box3 | theWig = Box4.

    % The clues.
    opens(Green, Box3) | opens(Green, Box4).
    left(theWig, Box4).
    right(theWig, Box1).
    left(Yellow, theWig).
    right(Blue, Yellow) & left(Blue, Green).
    opens(Red, Box1).
end_of_list.
```

Listing B.4: Hare and Tortoise

```
set(arithmetic).
assign(max_models, -1).
assign(domain_size, 100).

formulas(demodulators).
    Haretown = 0.
    Tortoisevillage = 27.
    HareSpeed = 7.
    TortoiseSpeed = 2.
end_of_list.

formulas(assumptions).
    combinedSpeed = HareSpeed + TortoiseSpeed.
    timeToMeet = (Tortoisevillage +(-Haretown)) / combinedSpeed.
    hareMiles = HareSpeed * timeToMeet.
end_of_list.
```

Listing B.5: How old is Hannah

```
set(arithmetic).
assign(max_models, -1).
assign(domain_size, 100).

formulas(demodulators).
    Fiona = 4.
end_of_list.

formulas(assumptions).
    twin(x, y) <-> x = y.

    % The clues.
    Hannah = Sasha * 4.
    Sasha = 5 + Andrew.
    Andrew = Nick + 1.
    twin(Fiona, Nick).
end_of_list.
```

Listing B.6: Mystery Number 6

```
set(arithmetic).
assign(domain_size, 10).
assign(max_models, -1).

list(distinct).        % Objects in each list are distinct.
   [A, B, C, D, E, F, G, H, I, J].
end_of_list.

formulas(utils).
```

```
    odd(x) <-> x mod 2 = 1.
    even(x) <-> -odd(x).
    prime(x) <-> x = 2 | x = 3 | x = 5 | x = 7.
    cube(x) <-> x = 0 | x = 1 | x = 8 | x = 9.
    square(x) <-> x = 0 | x = 1 | x = 4 | x = 9.
    triangle(x) <-> x = 1 | x = 3 | x = 6.
end_of_list.

formulas(assumptions).
    A != 0.

    %The clues.
    (square(A) & -triangle(A)) | (-square(A) & triangle(A)).  %1
    (even(B) & -cube(B)) | (-even(B) & cube(B)).              %2
    (cube(C) & -triangle(C)) | (-cube(C) & triangle(C)).      %3
    (odd(D) & -square(D)) | (-odd(D) & square(D)).            %4
    (odd(E) & -cube(E)) | (-odd(E) & cube(E)).                %5
    (odd(F) & -triangle(F)) | (-odd(F) & triangle(F)).        %6
    (odd(G) & -prime(G)) | (-odd(G) & prime(G)).              %7
    (even(H) & -square(H)) | (-even(H) & square(H)).          %8
    (square(I) & -cube(I)) | (-square(I) & cube(I)).          %9
    (prime(J) & -triangle(J)) | (-prime(J) & triangle(J)).    %10
    %11
    A < B & C < D & E < F & G < H & I < J.
    %12
    A + B + C + D + E < F + G + H + I + J.
end_of_list.
```

Listing B.7: Snow White

```
set(arithmetic).
assign(max_models, -1).
assign(domain_size, 10).

list(distinct).
   [Doc, Happy, Smelly, Sneezy, Stumpy, Sleepy, Grumpy, Dopey, Droopy, Bashful].
end_of_list.

formulas(assumptions).
   %Definitions
   in_front(x, y) <-> x < y.

   %Clues
   in_front(Grumpy, Dopey).
   -in_front(Stumpy, Sneezy).
   -in_front(Stumpy, Doc).
   in_front(Doc, Droopy).
   in_front(Doc, Happy).
   -in_front(Sleepy, Stumpy).
   -in_front(Sleepy, Smelly).
   -in_front(Sleepy, Happy).
   in_front(Happy, Sleepy).
   in_front(Happy, Smelly).
   in_front(Happy, Bashful).
   -in_front(Bashful, Smelly).
   -in_front(Bashful, Droopy).
   -in_front(Bashful, Sleepy).
   in_front(Sneezy, Dopey).
   in_front(Smelly, Grumpy).
   in_front(Smelly, Stumpy).
```

```
    in_front(Smelly, Sneezy).
    in_front(Dopey, Droopy).
    in_front(Sleepy, Grumpy).
    in_front(Sleepy, Bashful).
    -in_front(Dopey, Sneezy).
    -in_front(Dopey, Doc).
    -in_front(Dopey, Sleepy).
    in_front(Stumpy, Dopey).
    -in_front(Smelly, Doc).
end_of_list.
```

Listing B.8: Einstein's Riddle

```
set(arithmetic).
assign(max_models, -1).
assign(domain_size, 5).

 list (distinct).
    [_yellow, blue, red, green, _white].
    [norwegian, dane, brit, german, swede].
    [_water, tea, milk, coffee, beer].
    [cats, horse, birds, fish, dogs].
    [dunhill, blends, pall_malls, prince, bluemasters].
end_of_list.

formulas(assumptions).
    %Definitions
    right_neighbor(x,y) <-> x+1 = y.
        neighbors(x,y) <-> right_neighbor(x,y) | right_neighbor(y,x).

    %Clues
     brit = red.
    swede = dogs.
    dane = tea.
    right_neighbor(green, _white).
    green = coffee.
    pall_malls = birds.
    _yellow = dunhill.
    milk = 2.
    norwegian = 0.
    neighbors(blends, cats).
    neighbors(horse, dunhill).
    bluemasters = beer.
    german = prince.
    neighbors(norwegian, blue).
    neighbors(blends, _water).
end_of_list.
```

Listing B.9: The Father of Algebra

```
set(arithmetic).
assign(domain_size, 200).
assign(max_models, -1).

formulas(assumptions).

     %The clues.
     (d/6 + d/12 + d/7 + 5 + d/2 + 4) = d.
     d mod 6 = 0.
     d mod 12 = 0.
```

```
    d mod 7 = 0.
    d mod 2 = 0.
end_of_list.
```

Listing B.10: Inspector Beethoven

```
assign(max_models, -1).
assign(domain_size, 2).

formulas(assumptions).
    J1 <-> -J2.
    Gr1 <-> -Gr2.
    S1 <-> -S2.
    Ge1 <-> -Ge2.


    J1 <-> -J.
    J2 <-> Gr | (-J & -Gr & -S & -Ge).

    Gr1 <-> -Gr.
    Gr2 <-> Ge.

    S1 <-> -S.
    S2 <-> -Gr2.

    Ge1 <-> -Ge.
    Ge2 <-> -J & Gr.

end_of_list.
```

Appendix C

Original code for A3

Listing C.1: Domain

```
(define (domain allout)
    (:requirements :adl)

    (:types
        flag
        notpressable
        pressable
    )

    (:predicates
        (on ?row ?col)
        (next-row ?r1 ?r2)
        (next-column ?c1 ?c2)
        (pressed ?r ?c)
        (wait-for ?f)
    )

    (:action switch_neighbour_up
        :parameters (?row ?col ?down_row)
        :precondition (and
            (wait-for up)
            (next-row ?row ?down_row)
            (pressed ?down_row ?col)
        )
        :effect (and
            (when (on ?row ?col) (not (on ?row ?col)))
            (when (not (on ?row ?col)) (on ?row ?col))
            (not (wait-for up))
        )
    )

    (:action switch_neighbour_down
        :parameters (?row ?col ?up_row)
        :precondition (and
            (wait-for down)
            (next-row ?up_row ?row)
            (pressed ?up_row ?col)
        )
        :effect (and
            (when (on ?row ?col) (not (on ?row ?col)))
            (when (not (on ?row ?col)) (on ?row ?col))
            (not (wait-for down))
        )
    )

        (:action switch_neighbour_left
        :parameters (?row ?col ?right_col)
        :precondition (and
            (wait-for left )
            (next-column ?col ?right_col)
            (pressed ?row ?right_col)
        )
        : effect (and
            (when (on ?row ?col) (not (on ?row ?col)))
            (when (not (on ?row ?col)) (on ?row ?col))
            (not (wait-for left ))
        )
    )

    (:action switch_neighbour_right
        :parameters (?row ?col ?left_col)
        :precondition (and
            (wait-for right)
            (next-column ?left_col ?col)
            (pressed ?row ?left_col)
        )
        : effect (and
            (when (on ?row ?col) (not (on ?row ?col)))
            (when (not (on ?row ?col)) (on ?row ?col))
            (not (wait-for right))
        )
    )


    (:action wait_for_neighbours
        :parameters (?row ?col)
        :precondition (and
            (pressed ?row ?col)
            (not (wait-for right))
            (not (wait-for left ))
            (not (wait-for up))
            (not (wait-for down))
        )
        : effect (and
            (not (pressed ?row ?col))
            (not (wait-for mid))
        )
    )

    (:action switch_current-------------------------
        :parameters (?row - pressable ?col - pressable)
        :precondition (and
            (not (pressed ?row ?col))
            (not (wait-for mid))
        )
        : effect (and
            (when (on ?row ?col) (not (on ?row ?col)))
            (when (not (on ?row ?col)) (on ?row ?col))
            (pressed ?row ?col)
            (wait-for mid)
            (wait-for right)
            (wait-for left )
            (wait-for up)
            (wait-for down)
        )
    )
)
```

Listing C.2: Problem 1

```
(define (problem allout-1)
    (:domain allout)

    (:objects
        mid up down left right - flag
        row0 row6 col0 col6 - notpressable
        row1 row2 row3 row4 row5 - pressable
        col1 col2 col3 col4 col5 - pressable)
    (: init
        (next-row row0 row1)        (next-column col0 col1)
        (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)
        (on row1 col1) (on row1 col2)    (on row1 col4) (on row1 col5)
        (on row2 col1)                                  (on row2 col5)

        (on row4 col1)                                  (on row4 col5)
        (on row5 col1) (on row5 col2)    (on row5 col4) (on row5 col5)
    )

    (:goal (and
            (not (on row1 col1)) (not (on row1 col2)) (not (on row1 col3)) (not (on row1 col4)) (not (on row1 col5))
            (not (on row2 col1)) (not (on row2 col2)) (not (on row2 col3)) (not (on row2 col4)) (not (on row2 col5))
            (not (on row3 col1)) (not (on row3 col2)) (not (on row3 col3)) (not (on row3 col4)) (not (on row3 col5))
            (not (on row4 col1)) (not (on row4 col2)) (not (on row4 col3)) (not (on row4 col4)) (not (on row4 col5))
            (not (on row5 col1)) (not (on row5 col2)) (not (on row5 col3)) (not (on row5 col4)) (not (on row5 col5))


            (not (pressed row1 col1)) (not (pressed row1 col2)) (not (pressed row1 col3)) (not (pressed row1 col4)) (not (pressed row1 col5))
            (not (pressed row2 col1)) (not (pressed row2 col2)) (not (pressed row2 col3)) (not (pressed row2 col4)) (not (pressed row2 col5))
            (not (pressed row3 col1)) (not (pressed row3 col2)) (not (pressed row3 col3)) (not (pressed row3 col4)) (not (pressed row3 col5))
            (not (pressed row4 col1)) (not (pressed row4 col2)) (not (pressed row4 col3)) (not (pressed row4 col4)) (not (pressed row4 col5))
            (not (pressed row5 col1)) (not (pressed row5 col2)) (not (pressed row5 col3)) (not (pressed row5 col4)) (not (pressed row5 col5))
        )
    )
)
```

Listing C.3: Problem 2

```
(define (problem allout-1)
    (:domain allout)

    (:objects
        mid up down left right − flag
        row0 row5 col0 col5 − notpressable
        row1 row2 row3 row4 − pressable
        col1 col2 col3 col4 − pressable)
    (:init
        (next−row row0 row1)         (next−column col0 col1)
        (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)
        (on row3 col2) (on row3 col3) (on row3 col4)
        (on row4 col1) (on row4 col3)
    )

    (:goal (and
            (not (on row1 col1)) (not (on row1 col2)) (not (on row1 col3)) (not (on row1 col4))
            (not (on row2 col1)) (not (on row2 col2)) (not (on row2 col3)) (not (on row2 col4))
            (not (on row3 col1)) (not (on row3 col2)) (not (on row3 col3)) (not (on row3 col4))
            (not (on row4 col1)) (not (on row4 col2)) (not (on row4 col3)) (not (on row4 col4))


            (not (pressed row1 col1)) (not (pressed row1 col2)) (not (pressed row1 col3)) (not (pressed row1 col4))
            (not (pressed row2 col1)) (not (pressed row2 col2)) (not (pressed row2 col3)) (not (pressed row2 col4))
            (not (pressed row3 col1)) (not (pressed row3 col2)) (not (pressed row3 col3)) (not (pressed row3 col4))
            (not (pressed row4 col1)) (not (pressed row4 col2)) (not (pressed row4 col3)) (not (pressed row4 col4))
        )
    )
)
```

Listing C.4: Problem 3

```
(define (problem allout−1)
    (:domain allout)

    (:objects
        mid up down left right − flag
        row0 row4 col0 col4 − notpressable
        row1 row2 row3 − pressable
        col1 col2 col3 − pressable)
    (:init
        (next−row row0 row1)         (next−column col0 col1)
        (next−row row1 row2)         (next−column col1 col2)
        (next−row row2 row3)         (next−column col2 col3)
        (next−row row3 row4)         (next−column col3 col4)
    )

    (:goal (and
                (on row1 col1)       (on row1 col2)        (on row1 col3)
                (on row2 col1)  (not (on row2 col2))       (on row2 col3)
                (on row3 col1)       (on row3 col2)        (on row3 col3)


            (not (pressed row1 col1)) (not (pressed row1 col2)) (not (pressed row1 col3))
            (not (pressed row2 col1)) (not (pressed row2 col2)) (not (pressed row2 col3))
            (not (pressed row3 col1)) (not (pressed row3 col2)) (not (pressed row3 col3))
        )
    )
)
```

Listing C.5: Problem 4

```
(define (problem allout-1)
    (:domain allout)

    (:objects
        mid up down left right - flag
        row0 row6 col0 col6 - notpressable
        row1 row2 row3 row4 row5 - pressable
        col1 col2 col3 col4 col5 - pressable)
    (: init
        (next-row row0 row1)        (next-column col0 col1)
        (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)
        (on row1 col2)
        (on row2 col1) (on row2 col2) (on row2 col3)
        (on row3 col2) (on row3 col4)
        (on row4 col3) (on row4 col4) (on row4 col5)
        (on row5 col4)
    )

    (:goal (and
            (not (on row1 col1)) (not (on row1 col2)) (not (on row1 col3)) (not (on row1 col4)) (not (on row1 col5))
            (not (on row2 col1)) (not (on row2 col2)) (not (on row2 col3)) (not (on row2 col4)) (not (on row2 col5))
            (not (on row3 col1)) (not (on row3 col2)) (not (on row3 col3)) (not (on row3 col4)) (not (on row3 col5))
            (not (on row4 col1)) (not (on row4 col2)) (not (on row4 col3)) (not (on row4 col4)) (not (on row4 col5))
            (not (on row5 col1)) (not (on row5 col2)) (not (on row5 col3)) (not (on row5 col4)) (not (on row5 col5))


            (not (pressed row1 col1)) (not (pressed row1 col2)) (not (pressed row1 col3)) (not (pressed row1 col4)) (not (pressed row1 col5))
            (not (pressed row2 col1)) (not (pressed row2 col2)) (not (pressed row2 col3)) (not (pressed row2 col4)) (not (pressed row2 col5))
            (not (pressed row3 col1)) (not (pressed row3 col2)) (not (pressed row3 col3)) (not (pressed row3 col4)) (not (pressed row3 col5))
            (not (pressed row4 col1)) (not (pressed row4 col2)) (not (pressed row4 col3)) (not (pressed row4 col4)) (not (pressed row4 col5))
            (not (pressed row5 col1)) (not (pressed row5 col2)) (not (pressed row5 col3)) (not (pressed row5 col4)) (not (pressed row5 col5))
        )
    )
)
```

Listing C.6: Problem 5

```
(define (problem allout-1)
    (:domain allout)

    (:objects
        mid up down left right − flag
        row0 row6 col0 col6 − notpressable
        row1 row2 row3 row4 row5 − pressable
        col1  col2  col3  col4  col5  − pressable)
    (: init
        (next−row row0 row1)          (next−column col0 col1)
        (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)
        (on row1 col1) (on row1 col3) (on row1 col5)
        (on row2 col1) (on row2 col3) (on row2 col5)
        (on row4 col1) (on row4 col3) (on row4 col5)
        (on row5 col1) (on row5 col3) (on row5 col5)
    )

    (:goal (and
            (not (on row1 col1)) (not (on row1 col2)) (not (on row1 col3)) (not (on row1 col4)) (not (on row1 col5))
            (not (on row2 col1)) (not (on row2 col2)) (not (on row2 col3)) (not (on row2 col4)) (not (on row2 col5))
            (not (on row3 col1)) (not (on row3 col2)) (not (on row3 col3)) (not (on row3 col4)) (not (on row3 col5))
            (not (on row4 col1)) (not (on row4 col2)) (not (on row4 col3)) (not (on row4 col4)) (not (on row4 col5))
            (not (on row5 col1)) (not (on row5 col2)) (not (on row5 col3)) (not (on row5 col4)) (not (on row5 col5))


            (not (pressed row1 col1)) (not (pressed row1 col2)) (not (pressed row1 col3)) (not (pressed row1 col4)) (not (pressed row1 col5))
            (not (pressed row2 col1)) (not (pressed row2 col2)) (not (pressed row2 col3)) (not (pressed row2 col4)) (not (pressed row2 col5))
            (not (pressed row3 col1)) (not (pressed row3 col2)) (not (pressed row3 col3)) (not (pressed row3 col4)) (not (pressed row3 col5))
            (not (pressed row4 col1)) (not (pressed row4 col2)) (not (pressed row4 col3)) (not (pressed row4 col4)) (not (pressed row4 col5))
            (not (pressed row5 col1)) (not (pressed row5 col2)) (not (pressed row5 col3)) (not (pressed row5 col4)) (not (pressed row5 col5))
        )
    )
)
```