Department of Computer Science
Technical University of Cluj-Napoca

Artificial Intelligence

Laboratory activity

Name: Iobaj Andrei Sebastian
Group: 30433
Email: Iobaj.Se.Andrei@student.utcluj.ro

Name: Moldovan Alexandru Cristian
Group: 30433
Email: Moldovan.Io.Cristian@student.utcluj.ro

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

Contents

Table 1: Lab scheduling

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

Lab organisation.

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

Chapter 1

A1: Search

Introduction:

- This section of the lab activity is related to the development and implementation of multiple search related algorithms that will help Pacman reach different goals in more or less relaxed conditions. Also, algorithms related to the Eight Puzzle Problem will be presented as well, in this section.
- In order to achieve our goals, we have implemented multiple search algorithms that will be listed below, solutions to the Search All Food Problem and All Corners Problem, multiple heuristics and a multitude of ways for solving the Eight Puzzle Problem, plus a little easter egg that will be kept for when the time is right!

Search Algorithms:

Search problems are very common in Artificial Intelligence and involve a Search Agent, in our case, Pacman himself, to search for his goal in a maze. This is achieved via the use of Search Algorithms. Their purpose is to guide the Search Agent to the goal, through the shortest path or through the path with the smallest cost value.

In our project, we have implemented a few of these Search Algorithms. We will present them moving forward:

1. Depth First Search (DFS)

In the Depth First Search algorithm, the goal is to explore a graph by visiting a node and then, using recursion, traverse as far as possible along one of the branches. Only after we meet a visited cell or a leaf, we backtrack.

In our case, the graph is represented by a series of game states, representing Pacman's position, food dots and walls. The algorithm will try to explore as deeply as possible a path in the maze until it either collects all the food dots or meets a dead-end, backtracking to the most recently discovered path.

```
def depthFirstSearch(problem):
    """
    Search the deepest nodes in the search tree first.

    Your search algorithm needs to return a list of actions that reaches the
```

```
    goal. Make sure to implement a graph search algorithm.

    To get started, you might want to try some of these simple commands to
    understand the search problem that is being passed in:
    """

    "*** YOUR CODE HERE ***"

    moves = []
    s = []
    l = {}

    done = [False]

    l[problem.getStartState()] = True

    for v in problem.getSuccessors(problem.getStartState()):
        dfs_rec(problem, v, l, moves, done)
        if done[0]:
            break

    return moves
```

Listing 1.1: Depth First Search

2. Breadth First Search (BFS)

In the Breadth First Search Algorithm, the goal for searching a graph, starting with the root and exploring all the neighbours. The process is repeated, traversing the graph layer by layer. Unlike the DFS algorithm, no recursion is needed.

In our case, the root is the initial position of Pacman and the graph is the maze, just as before. We prioritize the exploration of all possible paths uniformly, thus ensuring that an optimal path will be found.

```
def breadthFirstSearch(problem):
    """Search the shallowest nodes in the search tree first."""
    "*** YOUR CODE HERE ***"

    moves = []
    queue = []
    l = {}
    parent = {}
    directions = {}
    goal = None

    l[problem.getStartState()] = True
    queue.append(problem.getStartState())

    parent[problem.getStartState()] = None
    directions[problem.getStartState()] = None

    while queue:
        v = queue.pop(0)
        if problem.isGoalState(v):
            goal = v
            break
        for w in problem.getSuccessors(v):
```

```
            if w[0] not in l.keys():
                l[w[0]] = True
                parent[w[0]] = v
                queue.append(w[0])
                directions[w[0]] = w[1]

    while parent[goal]:
        moves.insert(0, directions[goal])
        goal = parent[goal]

    return moves
```

Listing 1.2: Breadth First Search

3. Uniform Cost Search (UCS)

In the Uniform Cost Search algorithm, just as the previously mentioned algorithms with the goal of exploration, this one is the same with one exception, it adds a cost. As such, the algorithm will prioritise moves with the least cumulative cost.

In our case, the algorithm will focus on finding the path with the shortest total cost, while also achieving all of the given goals.

```
def uniformCostSearch(problem):
    """Search the node of least total cost first."""
    moves = []
    queue = util.PriorityQueue()
    explored = []
    cost = {}
    cost[problem.getStartState()] = 0
    queue.push(problem.getStartState(), 0)
    parent = {}
    goal = None

    while not queue.isEmpty():
        v = queue.pop()
        cost_v = cost[v]

        if problem.isGoalState(v):
            goal = v
            break

        if v not in explored:
            explored.append(v)

            for w in problem.getSuccessors(v):
                cost_w = cost_v + w[2]
                if (w[0] not in cost.keys()) or (cost[w[0]] > cost_w):
                    cost[w[0]] = cost_w
                    parent[w[0]] = (v, w[1])
                    queue.push(w[0], cost_w)

    while goal != problem.getStartState():
        v = parent[goal]
        moves.insert(0, v[1])
        goal = v[0]

```

```
    return moves
```

Listing 1.3: Uniform Cost Search

4. A Star Search (A*)

In the A Star Search algorithm, considered more complex than the other algorithms previously mentioned, the goal remains, however, like in the case of UCS, with the addition of a cost, we also add a heuristic. The goal of this heuristic is to guide the agent with an approximation cost value to achieve the end goal. The closer this value is to the true total cost, the better the heuristic.

In our case, we implemented a number of heuristics to work with this algorithm. As such, we can help Pacman with a good estimation of the total cost, prioritising those paths that return the lowest values, optimising the performance and end result finding time.

```
def aStarSearch(problem, heuristic=nullHeuristic):
    """Search the node that has the lowest combined cost and heuristic first."""
    "*** YOUR CODE HERE ***"

    openSet = util.PriorityQueue()
    cameFrom = {}

    gScore = defaultdict(lambda: float('inf'))
    gScore[problem.getStartState()] = 0

    fScore = defaultdict(lambda: float('inf'))
    fScore[problem.getStartState()] = heuristic(problem.getStartState(), problem)

    openSet.push(problem.getStartState(), fScore[problem.getStartState()])

    goal = None
    moves = []

    while not openSet.isEmpty():
        current = openSet.pop()

        if problem.isGoalState(current):
            goal = current
            break

        for neighbour in problem.getSuccessors(current):
            tentativeGScore = gScore[current] + neighbour[2]

            if tentativeGScore < gScore[neighbour[0]]:
                cameFrom[neighbour[0]] = (current, neighbour[1])
                gScore[neighbour[0]] = tentativeGScore
                fScore[neighbour[0]] = tentativeGScore + heuristic(neighbour[0], problem)
                openSet.update(neighbour[0], fScore[neighbour[0]])

    while goal != problem.getStartState():
            v = cameFrom[goal]
            moves.insert(0, v[1])
            goal = v[0]
```

```
39
40      return moves
```

Listing 1.4: A Star Search

5. Weighted A Star Search

In Weighted A Star Search algorithm, derived from the A star Search algorithm, we use the same principle as before, but we assign a weight and the goal is to find the path which minimizes the weighted sum of both the path cost and a heuristic estimate of the remaining cost to the goal.

In our case, we used the above mentioned principle and assigned the maze a weight of 1.5. We used the A* Search code for this, added the weight and as such, help the agent make better decisions that prioritize specific objectives.

```
1  def weightedAStarSearch(problem, weight = 1.5, heuristic=nullHeuristic):
2      """Search the node that has the lowest combined cost and heuristic first
       """
3      "*** YOUR CODE HERE ***"
4
5      openSet = util.PriorityQueue()
6      cameFrom = {}
7
8      gScore = defaultdict(lambda: float('inf'))
9      gScore[problem.getStartState()] = 0
10
11     fScore = defaultdict(lambda: float('inf'))
12     fScore[problem.getStartState()] = fScoreFunc(0, heuristic(problem.
       getStartState(), problem), weight)
13
14     openSet.push(problem.getStartState(), fScore[problem.getStartState()])
15
16     goal = None
17     moves = []
18
19     while not openSet.isEmpty():
20         current = openSet.pop()
21
22         if problem.isGoalState(current):
23             goal = current
24             break
25
26         for neighbour in problem.getSuccessors(current):
27             tentativeGScore = gScore[current] + neighbour[2]
28
29             if tentativeGScore < gScore[neighbour[0]]:
30                 cameFrom[neighbour[0]] = (current, neighbour[1])
31                 gScore[neighbour[0]] = tentativeGScore
32                 fScore[neighbour[0]] = fScoreFunc(tentativeGScore, heuristic
       (neighbour[0], problem), weight)
33                 openSet.update(neighbour[0], fScore[neighbour[0]])
34
35     while goal != problem.getStartState():
36             v = cameFrom[goal]
37             moves.insert(0, v[1])
38             goal = v[0]
```

```
39
40      return moves
41
42  def fScoreFunc(g, h, w):
43      if g < h:
44          return g + h
45      else:
46          return (g + (2 * w - 1) * h) / w
```

Listing 1.5: Weighted A Star Search

6. Random Search

In Random Search algorithm, the strategy is to find the goal using random moves at each step. Surprisingly, the algorithm is complete, because, if the agent is given enough time, the task will be completed.

In our case, it is as simple as it gets, Pacman will move in a random direction at each step (obviously, it being a legal move). It will randomly navigate through the maze until all the food dots are collected.

```
1   def randomSearch(problem):
2       stack = util.Stack()
3       stack.push(problem.getStartState())
4       visited = []
5       visited.append(problem.getStartState())
6       cameFrom = {}
7       moves = []
8       goal = None
9       while not stack.isEmpty():
10          currentState = stack.pop()
11          if problem.isGoalState(currentState):
12              goal = currentState
13              break
14
15          successors = problem.getSuccessors(currentState)
16          next = random.choice(successors)
17          successors.remove(next)
18
19          while next[0] in visited:
20              if len(successors) == 0:
21                  break
22              next = random.choice(successors)
23              successors.remove(next)
24
25          if next[0] not in visited:
26              cameFrom[next[0]] = (currentState, next[1])
27              stack.push(next[0])
28              visited.append(next[0])
29
30      if goal == None:
31          return []
32
33      while goal != problem.getStartState():
34          v = cameFrom[goal]
35          moves.insert(0, v[1])
36          goal = v[0]
```

```
37
38      return moves
```

Listing 1.6: Random Search

Search Problems:

More advanced search problems, such as Find all Corners Problem or Eat all Food Problem, are a good way to test the previously mentioned algorithms in more complex or dynamic scenarios and conclude if their behaviour is as good as expected.

In our work, we implemented solutions for both of the previously mentioned problems which will be presented below, with the code being provided in the Appendix A section

7. Find all Corners Problem

The Find all Corners Problem involves helping Pacman find the food situated in the 4 corners of the maze. The execution ends when Pacman successfully collected these 4 dots of food.

In our implementation we modified getStartState(), isGoalState() and getSuccessors() functions from CornersProblem and changed the state to be a tuple that contains the position and remaining corners. We modified getStartState() to return a tuple that contains the start state and the corners. In isGoalState() we simply check if there are no remaining corners and return True in that case, otherwise False. For getSuccessors() we check to see if the next position is not a wall, if is not, we compute the nextState as the next position and the remaining corners. If the next state is a corner, we remove it from the list of corners of that state. For the cornerHeuristic function we compute the distance to all remaining corners and return the maximum distance

Look at Appendix A.1 for the implementation

8. Eat all Food Problem

The Eat all Food Problem involves finding paths for Pacman to collect all the food dots within a given maze. The execution ends when all the food in the maze has been eaten.

For FoodHeuristic we used the already implemented mazeDistance() function to compute the distance between the current position and the food on the map. We then store the position to the nearest food and the position to the furthest one and also the minimum and maximum distance. We then compute the value of the heuristic as min distance + maze distance between nearestf ood and furthest food.

Look at Appendix A.2 for the implementation

9. Diagonal Search Problem

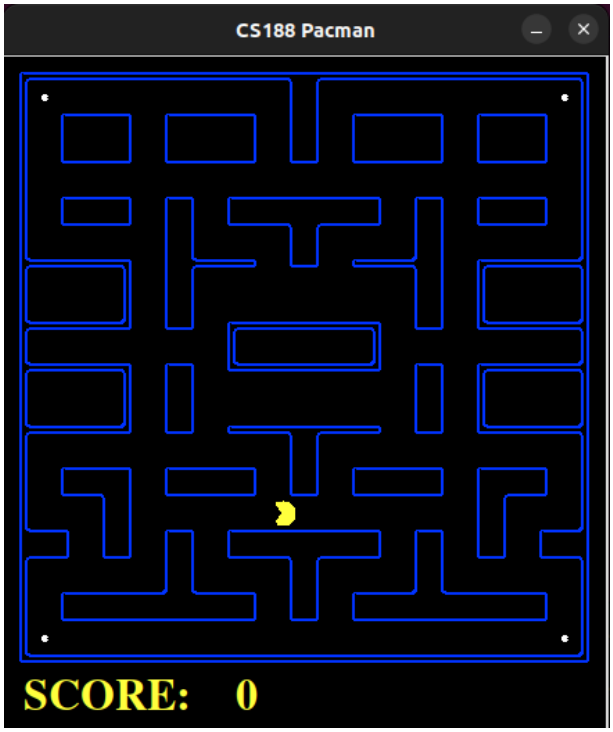
Diagonal Search Problem is a more relaxed search problem in which we allow Pacman to move not just up, down, right or left, but also in diagonal. This helps the agent find their goal faster by possibly skipping a few extra moves that would be required to achieve their goal.

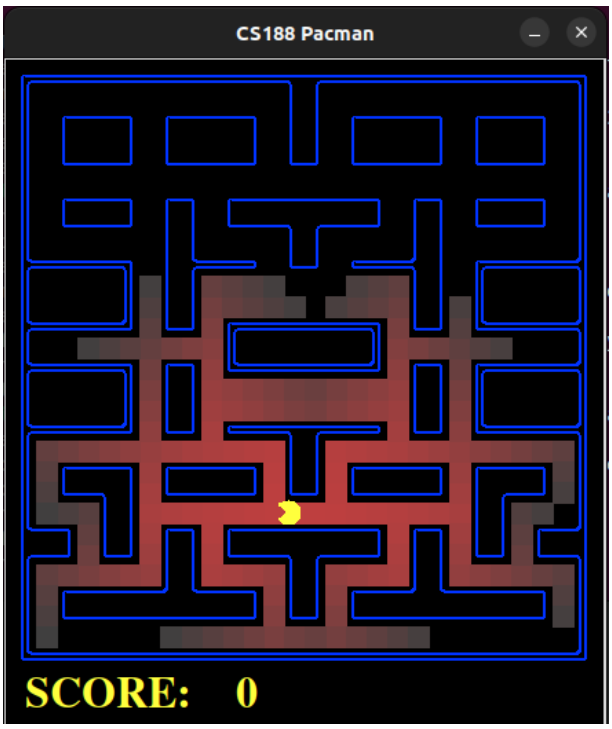
In our implementation we simply changed the way a successor state is found, by allowing Pacman to move to an increased number of neighbouring cells. After that, we let a search algorithm do their job until the goal is reached.

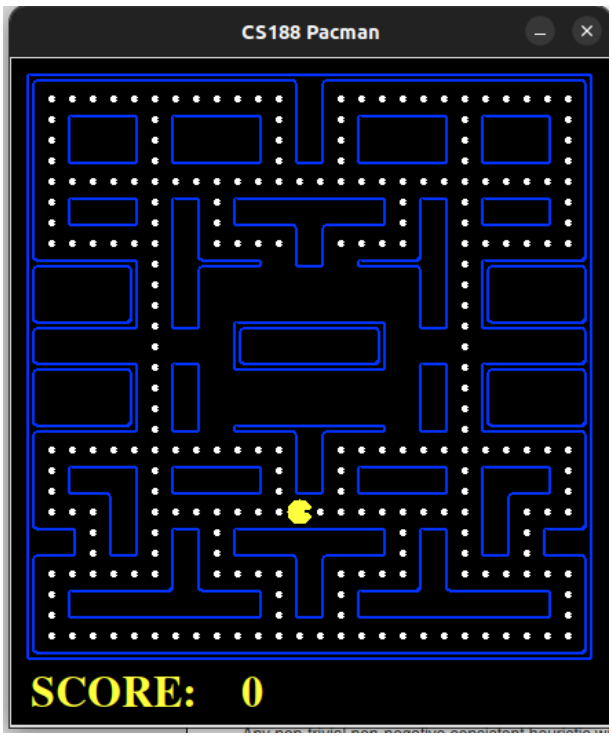
Look at Appendix A.3 for the implementation

10. Custom Maps

We decided to test the limits of our code and we put Pacman to the test on a few custom mazes we did. We managed to test his skills in a few different scenarios, showing promising results. After that, we made it more difficult by making a maze with a lot of food dots. Unfortunately, this maze takes a long time to complete and as such, we will only show the maze configurations here, but not his performance in any of them.







Eight Puzzle:

The Eight Puzzle is a classic problem in Artificial Intelligence and puzzle-solving. It consists of a 3x3 grid with eight numbered tiles and one empty space. The objective is to rearrange the tiles from an initial, presumably scrambled configuration to a well defined goal state. This problem is considered one of the most fundamental ones in AI, because it can use a multitude of search algorithms to reach the final goal.

Because we wanted to try something different, we moved on from the Pacman game and instead focused more of our time here, where we developed a multitude of ways we could solve

the Eight Puzzle Problem, that we will present below.

What we worked on?

- The 8 Puzzle Problem can be run with the A* Search or Weighted A* Search, not with just BFS
- A multitude of heuristics
- A comparison table, between all of the implemented heuristics
- The possibility to run the program from the terminal with a set or predefined settings and options
- Larger sized puzzles(16 and 25)
- The Easter egg!

Let us discuss what we did!

• Algorithm runs with the A* or Weighted A* Search algorithms

We wanted to make things interesting and as such, we decided to switch the already implemented BFS algorithm with A* for more action!

At first we thought a simple copy paste of the code would work, making our life easy from the get-go. That was not the case and a bunch of compilation errors quickly made us realise we need to adapt the data structures for the problem at hand.

After the modifications were implemented (see Listing 1.4), the problem would get solved, but the time it took to get solved would differ, it not begin very time efficient. We quickly decided that we had to implement some better heuristics than the default Null Heuristc.

• Everyone gets a heuristic!

Because we were very curious about the topic of heuristic comparison, we implemented 6 heuristics and we wanted to compare them (more on than later). As such, we did a little research and picked 6 heuristics we wanted to implement and compare:

- Null heuristic -> default heuristic that always returns 0.
- Tile Misplaced heuristic -> counts the number of tiles that are not in the correct position, but only once and returns that value
- Manhattan Distance heuristic -> adds the the distance on the oX and oY axis to reach the goal and returns the sum.
- Euclidian Distance heuristic -> computes the the distance that goes straight to the goal (straight line) and returns it.

- Out of Row Out of Column heuristic -> counts on each row and column the number of misplaced tiles, returning the sum of the 2 values.

- Swap heuristic -> it computes the number of swaps it takes to place every single tile in the correct configuration and returns that value

Every one of them works and gives more or less different times of completion, with the same grid configuration, of course. Now, let us see how they fare against each other!

• Which one is the best one?

It is now time for some statistics! Below we have included the data we collected from the test runs we did. We decided to use the puzzle [5, 0, 8, 1, 4, 2, 3, 6, 7] for our testing, on a size of 3x3.

Heuristic	Completion Time	No. Expanded Nodes
Null Heuristic	14.888	7048
Tile Misp.	0.064	422
Manh. Dist.	0.026	150
Euclid. Dist.	0.030	167
Out Row Out Col.	0.019	145
Swap	0.234	902

From the above data we can draw some conclusions:

- The best heuristic is the Out of Row Out of Column heuristic, with the smallest number of expanded nodes, this was a surprise, because we thought that the best one would be the Manhattan distance heuristic, which came in second place.

- The worst heuristic in the Null heuristic. No big surprise here, considering it always returns 0, helping with basically nothing the searching algorithm. So, informed heuristics have a big impact on the performance of the search.

- While it's true that the data does not lie, it may be possible that, on other grid configurations, another heuristic may prove more efficient than the Out of Row Out of Column heuristic. This means that every heuristic has its own purpose and the table above is specific to our example and we cannot conclude for sure, which heuristic is, objectively speaking, the best.

• Running the program from the terminal

We wanted to make the program run through the terminal. As such, we grouped the program better into functions, imported a few libraries and set the options for the run commands. It is possible to set the heuristic from the 6 possible heuristics (by default, Null Heuristic is selected) and the size of the grid (by default, 3x3 is selected) and the function (by default, A* is selected).

In order to use the command line arguments, you have to write a syntax like:

python eightpuzzle.py --heuristic <NameOfHeuristic> -s <Size as an Int> -f <Name of the function>

- Null heuristic -> by default
- Tile Misplaced heuristic -> "tileMisplaced"
- Manhattan Distance heuristic -> "manhattan"
- Euclidian Distance heuristic -> "euclidian"
- Out of Row Out of Column heuristic -> "outOfColumnRow"
- Swap heuristic -> "swap"

- A* Search -> "astar", or by default
- Weighted A* Search -> "wastar"
- Stalin Sort (more on this later) -> "StalinSort"

- Larger grids!

We wanted to see if our implementations hold true for larger puzzles. We changed some hard-coded sizes, made a few adjustments for our program to be able to take a desired size and it would appear that the code works great!

- Easter egg hunt!

The following is a PAMPHLET, treat it as such!

When we did a little research on a few more 'not so general' functions we could implement for our project, we stumbled on a very sketchy site with a few strange functions that work, but are a lot more memes than actual coding. Here, we saw a type of sorting called "Stalin Sort". Yes, it actually exists and works like this: We travel an array and if an elemnt is not in the correct position for the array to be sorted, it is simply 'executed' (taken out of the array) and you repeat this until the array becomes sorted. The number of elements left is irrelevant, just like in real life, with the political adversaries of Stalin.

Because we wanted something to make the project our own, we decide to implement this kind of logic in the Eight Puzzle, simply taking out the elements that, by default, are not in the correct position. Quick and easy! *Pew*

The code can be found in the Appendix!

Chapter 2

A2: Logics

Introduction:

- This section of the lab activity is related to the development and implementation of some problems related to Prover9/Mace4. The goal is to design and implement a project regarding an Among Us simulator based on First Order Logic and Python as a support for the game system.
- In order to achieve our goals, we have implemented multiple game systems to determine the likelihood of a person being the impostor in FOL for Mace4, then implemented in Python a game simulator to generate these messages. Then it is up to you, the player, to determine the impostor.

Knights & Knaves:

The original concept was to implement simpler concepts at first for the purpose of practicing for the real Among Us implementation and to learn the basic and more complex syntax of FOL for the Mace4 program.

We took some problems from the 382 Knights and Knaves problems site provided to us and we started working on those. Taking as examples solved problems by our professor, Adrian Groza, we quickly got to work and solved a few problems that would serve as a starting point for the Among Us simulation later to be implemented.

Problem 1:

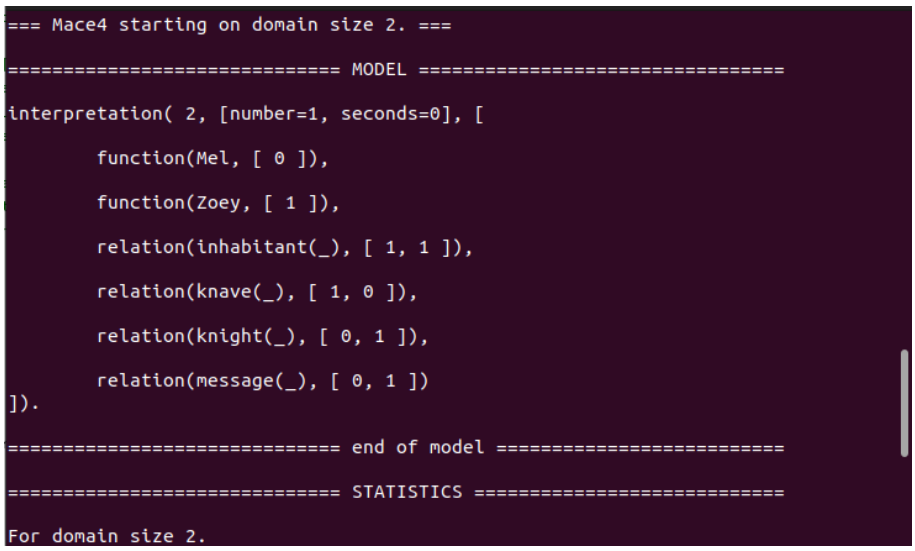
A very special island is inhabited only by knights and knaves. Knights always tell the truth, and knaves always lie.

You meet two inhabitants: Zoey and Mel. Zoey tells you that Mel is a knave. Mel says, "Neither Zoey nor I are knaves."

Can you determine who is a knight and who is a knave?

Human Solution: Before solving we determined that the knight is Zoey and the knave is Mel.

Mace4 Model:



Mace4 Solution to Problem 1

As we can see, Mace4 agrees with us on this one, determining that inhabitant 0, which is coded as Mel is a knave and inhabitant 1, coded as Zoey is a knight.

Code for Problem 1

Problem 2:

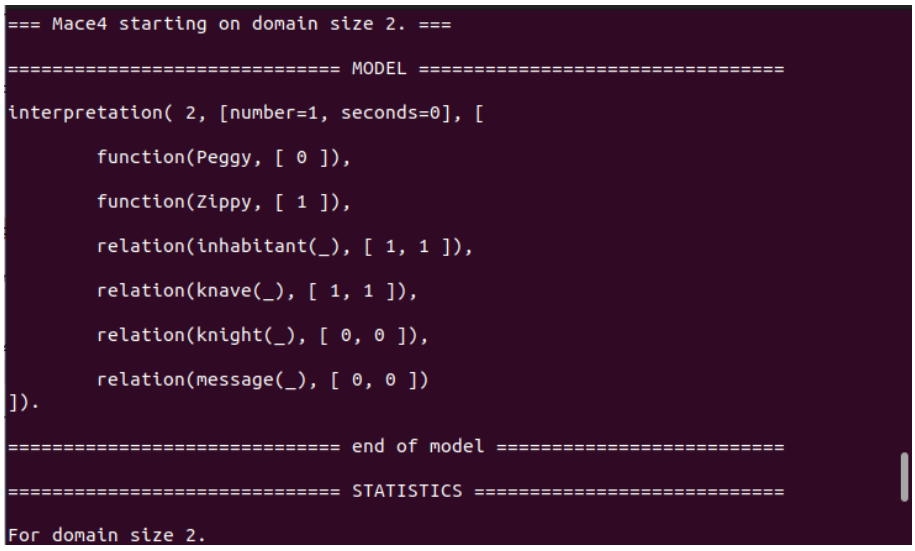
A very special island is inhabited only by knights and knaves. Knights always tell the truth, and knaves always lie.

You meet two inhabitants: Peggy and Zippy. Peggy tells you that "of Zippy and I, exactly one is a knight'. Zippy tells you that only a knave would say that Peggy is a knave.

Can you determine who is a knight and who is a knave?

Human Solution: Before solving we determined that both Peggy and Zippy are knaves.

Mace4 Model:



Mace4 Solution to Problem 2

As we can see, Mace4 agrees with us on this one, determining that inhabitant 0, which is coded as Peggy is a knave and inhabitant 1, coded as Zippy is also knave.

Code for Problem 2

Problem 3:

A very special island is inhabited only by knights and knaves. Knights always tell the truth, and knaves always lie.

You meet two inhabitants: Sue and Zippy. Sue says that Zippy is a knave. Zippy says, "I and Sue are knights."

Can you determine who is a knight and who is a knave?

Human Solution: Before solving we determined that Sue is a knight and Zippy is a knave.

Mace4 Model:

```
=== Mace4 starting on domain size 2. ===

============================== MODEL ==================================

interpretation( 2, [number=1, seconds=0], [

        function(Sue, [ 0 ]),

        function(Zippy, [ 1 ]),

        relation(inhabitant(_), [ 1, 1 ]),

        relation(knave(_), [ 0, 1 ]),

        relation(knight(_), [ 1, 0 ]),

        relation(message(_), [ 1, 0 ])
]).

============================ end of model ==========================

============================== STATISTICS ============================

For domain size 2.
```

Mace4 Solution to Problem 3

As we can see, Mace4 agrees with us on this one, determining that inhabitant 0, which is coded as Sue is a knight and inhabitant 1, coded as Zippy is a knave.

Code for Problem 3

Problem 4:

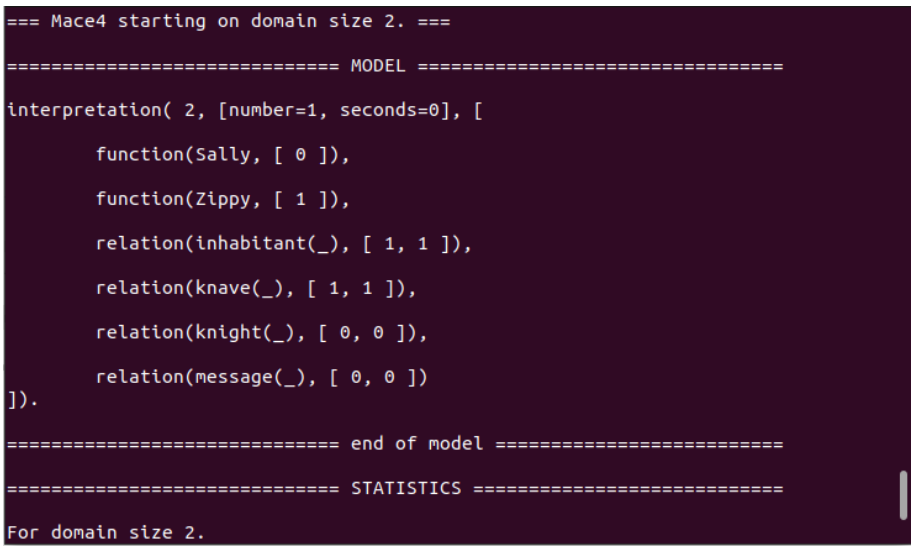
A very special island is inhabited only by knights and knaves. Knights always tell the truth, and knaves always lie.

You meet two inhabitants: Sally and Zippy. Sally claims, "I and Zippy are not the same." Zippy says, "Of I and Sally, exactly one is a knight."

Can you determine who is a knight and who is a knave?

Human Solution: Before solving we determined that both Sally and Zippy are knaves.

Mace4 Model:



Mace4 Solution to Problem 4

As we can see, Mace4 agrees with us on this one, determining that inhabitant 0, which is coded as Sally is a knave and inhabitant 1, coded as Zippy is a knave.

Code for Problem 4

Problem 5:

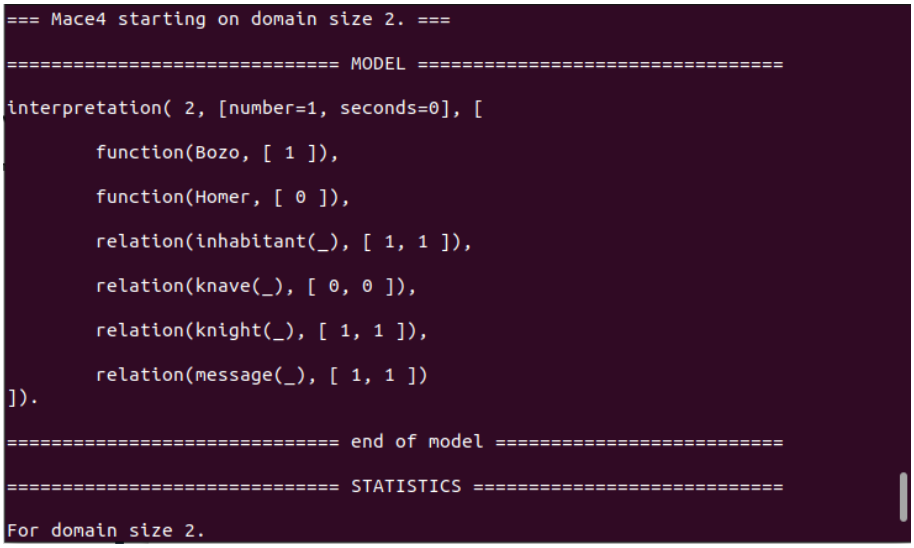
A very special island is inhabited only by knights and knaves. Knights always tell the truth, and knaves always lie.

You meet two inhabitants: Homer and Bozo. Homer tells you, "At least one of the following is true: that I am a knight or that Bozo is a knight." Bozo claims, "Homer could say that I am a knave."

Can you determine who is a knight and who is a knave?

Human Solution: Before solving we determined that both Homer and Bozo are knights.

Mace4 Model:



Mace4 Solution to Problem 5

As we can see, Mace4 agrees with us on this one, determining that inhabitant 0, which is coded as Homer is a knight and inhabitant 1, coded as Bozo is a knight.

Code for Problem 5

Problem 16:

A very special island is inhabited only by knights and knaves. Knights always tell the truth, and knaves always lie.

You meet two inhabitants: Bob and Mel. Bob tells you, "At least one of the following is true: that I am a knight or that Mel is a knave." Mel claims, "Only a knave would say that Bob is a knave."

Can you determine who is a knight and who is a knave?

Human Solution: Before solving we determined that both Bob and Mel are knights.

Mace4 Model:

```
=== Mace4 starting on domain size 2. ===

============================== MODEL ==================================

interpretation( 2, [number=1, seconds=0], [

        function(Bob, [ 0 ]),

        function(Mel, [ 1 ]),

        relation(inhabitant(_), [ 1, 1 ]),

        relation(knave(_), [ 0, 0 ]),

        relation(knight(_), [ 1, 1 ]),

        relation(message(_), [ 1, 1 ])
]).

============================== end of model ===========================

============================== STATISTICS =============================

For domain size 2.
```

Mace4 Solution to Problem 16

As we can see, Mace4 agrees with us on this one, determining that inhabitant 0, which is coded as Bob is a knight and inhabitant 1, coded as Mel is a knight.

Code for Problem 16

Problem 77:

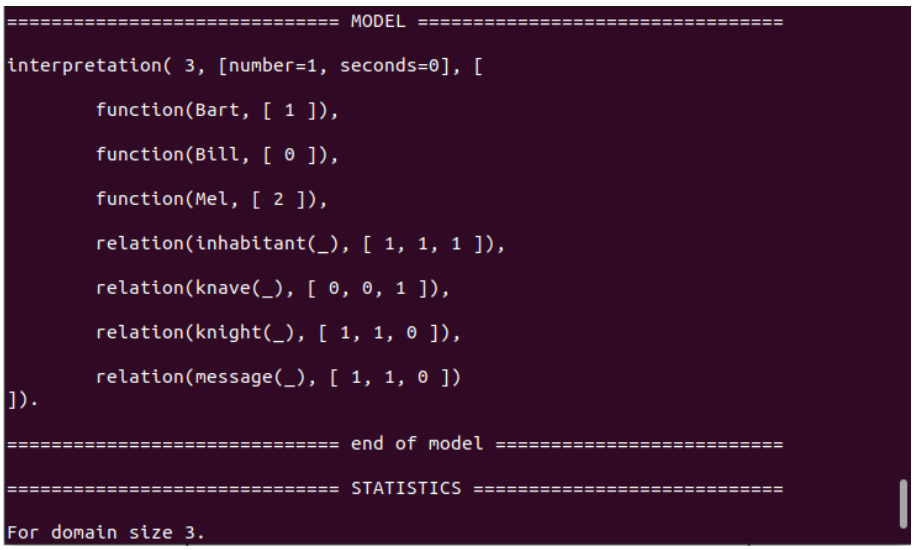
A very special island is inhabited only by knights and knaves. Knights always tell the truth, and knaves always lie.

You meet three inhabitants: Bill, Bart and Mel. Bill claims that it's false that Bart is a knave. Bart says that of Mel and I, exactly one is a knight. Mel claims, "I and Bart are both knights or both knaves."

Can you determine who is a knight and who is a knave?

Human Solution: Before solving we determined that both Bill and Bart are knights and Mel is a knave.

Mace4 Model:



Mace4 Solution to Problem 77

As we can see, Mace4 agrees with us on this one, determining that inhabitant 0, which is coded as Bill is a knight, inhabitant 1, coded as Bart is a knight and inhabitant 2, coded as Mel is a knave.

Code for Problem 77

Problem 379:

A very special island is inhabited only by knights and knaves. Knights always tell the truth, and knaves always lie.

You meet nine inhabitants: Joe, Homer, Bozo, Betty, Mel, Peggy, Rex, Carl and Marge. Joe tells you that Homer is a knave. Homer says, "It's not the case that Peggy is a knave." Bozo tells you, "Betty is a knave." Betty tells you that at least one of the following is true: that Rex is a knave or that Carl is a knave. Mel claims that Carl and Homer are knights. Peggy says that it's false that Rex is a knave. Rex claims, "Joe and Carl are knaves." Carl tells you that either Marge is a knight or Joe is a knave. Marge claims that Rex is a knave.

Can you determine who is a knight and who is a knave?

Human Solution: This problem proved to be too complicated to solve in a short time we proposed, so we didn't reach a conclusion.

Mace4 Model:

```
        function(Bozo, [ 2 ]),
        function(Carl, [ 7 ]),
        function(Homer, [ 1 ]),
        function(Joe, [ 0 ]),
        function(Marge, [ 8 ]),
        function(Mel, [ 4 ]),
        function(Peggy, [ 5 ]),
        function(Rex, [ 6 ]),
        relation(inhabitant(_), [ 1, 1, 1, 1, 1, 1, 1, 1, 1 ]),
        relation(knave(_), [ 0, 1, 1, 0, 1, 1, 1, 0, 0 ]),
        relation(knight(_), [ 1, 0, 0, 1, 0, 0, 0, 1, 1 ]),
        relation(message(_), [ 1, 0, 0, 1, 0, 0, 0, 1, 1 ])
]).
```

Mace4 Solution to Problem 379

Mace4 determined that inhabitant 0, which is coded as Joe is a knight, inhabitant 1, coded as Homer is a knave, inhabitant 2, coded as Bozo is a knave, inhabitant 3, coded as Betty is a knight, inhabitant 4, coded as Mel is a knave, inhabitant 5, coded as Peggy is a knave, inhabitant 6, coded as Rex is a knave, inhabitant 7, coded as Carl is a knight and inhabitant 8, coded as Marge is a knight.

Code for Problem 379

Knights, Knaves & Spies:

Now that the base knowledge of how Mace4 interprets lies and deception, we shall now add a new character, capable of both telling the truth to possibly lie at the same time. Mace4 needs to interpret these scenarios and determine if the spy lies or not.

We asked ChatGPT to provide us with some problems regarding 3 persons and these 3 character types. We took 2 problems and solved them in FOL and then gave them to Mace4 to see what solutions there are.

Problem 1:

We were given 3 individuals, Andrew, Billy and Charly. Andrew said that Charly is a knave, Billy said that Andrew is a knight and Charly said that they are a spy.

After coding this problem and giving it to Mace4, 8 models were resulted:

1: Everyone is a spy.
2: Billy is a knave, rest are spies.
3: Charly is a knave, rest are spies.
4: Billy and Charly are knaves, Andrew is spy.
5: Charly is a knave, Andrew is a knight and Billy is a spy.
6: Charly is a knave, rest are knights.
7: Andrew is a knave, rest are spies.
8: Andrew and Billy are knaves, Charly is spy.

Problem 2:

We were given 4 individuals, Alice, Bob, Charlie and Dylan. Alice said that either she is a knight and Charlie is knave or Bob is a knave and Dylan is a spy. Bob said that Alice is a knave and Dylan is a spy. Charlie said that she and Dylan are either knights, knaves or spies. Dylan said that Bob is either a spy or a knight.

After coding this problem and giving it to Mace4, 20 models were resulted:

1: Everyone is a spy, expect Dylan who is a knave.
2: Everyone is a spy.
3: Charlie and Dylan are knights, rest are spies.
4: Charlie is a knave, rest are spies.
5: Charlie is a knave, Dylan is a knight and rest are spies.
6: Bob and Dylan are knaves, rest are spies.
7: Bob is a knave, rest are spies.
8: Bob and Charlie are knaves, rest are spies.
9: Charlie is a knave, Alice is a knight and rest are spies.
10: Charlie is a knave, Alice and Dylan are nights and Bob is spy.
11: Bob is a knave, Alice is a knight and rest are spies.
12: Bob and Charlie are knaves, Alice is knight, Dylan is spy.
13: Alice is a knave, Dylan is a knight, rest are spies.
14: Alice is a knave, Charlie and Dylan are knights, Bob is spy.
15: Alice and Charlie are knaves, Dylan is a knight, Bob is spy.
16: Alice is a knave, rest are spies.
17: Alice and Charlie are knaves, rest are spies.
18: Alice is a knave, Bob is a knight, rest are spies.
19: Alice and Charlie are knaves, Bob is a knight, Dylan is spy.
20: Alice, Bob and Dylan are knaves, Charlie is spy.

Among FOL:

With the general knowledge gained from Knights, Knaves and Spies, we can now delve deeper into the art of mystery and deceit with Among Us! Our plan is to implement in FOL some set of commands (we shall call them systems), that help decipher the mystery of an Among Us Emergency Meeting.

The general idea is to set a few conditions that would make a person be an impostor. Obviously, we would need to simplify the problem greatly, because if we would analyze any small case if someone was or wasn't somewhere or if something happened but the people claim it didn't, we would not finish the project this academic year.

To solve this, we chose to set basic conditions for each of the system, such that the impostor can be identified via trivial mistakes they supposedly made in game, for instance, be seen near the body or claiming they made a task that in reality does not exist. Logic and simple!

Accusations:

The first system we implemented is the same accusation system present in Knights and Knaves problems. Someone claims that a player X is a certain role (in our case Crewmate or Impostor).

For the purpose of getting close to a more real life scenario, we made it so that, each Crewmate is telling the truth and each Impostor might or might not tell the truth.

We made it so that, each player can say whatever they want about whoever they see fit. The implementation can be found in the Appendix here. After testing and seeing it works fine, we moved on to the next system.

Doing Tasks:

The second system we implemented is a system of doing tasks. We make a few locations and assign a number of tasks to each location. The idea is that, some tasks are located in a specific location and some are not

If someone claims that they did a task and that task can be located in the same room they say they did the task, then they can be seen as telling the truth. If someone claims they did a task in a certain room, but that task cannot be found in that room, then they are lying and can be flagged as possible impostors.

The implementation of this system can be found in the Appendix here. We tested this system seeing that it works and then we continued onto the next.

Near Dead Body:

The third system we implemented was being in the same room as the dead body.

If a player is seen in a room by another player and there is a dead body in that room, then they can be flagged as impostors. This condition was heavily simplified due to the problem that the dead person might have been there for a while or in a room with the deceased could be a crewmate who was unfortunate enough to be there. These are cases that would take far too long to decipher, so we made it as easy and as simple as we could have.

The implementation of this system can be found in the Appendix here. We made a bunch of tests, as this system proved to be the most complicated due to the need of careful constraints considerations. After the testing proved to be satisfactory, we continued.

Venting:

The fourth and last system we implemented was the venting system. Its purpose is to let impostors use the vent between two rooms. Similar to the above system, if they are seen venting by a player and they actually did use the vent, then they are marked as an impostor.

Similarly as the Near Dead Body system, this one takes into account the locations of vents and if the person was indeed seen by a valid player. This way we can ensure that the event actually happened.

The implementation of this system can be found in the Appendix here. We tested this system and seeing it works, we considered it to be finished.

Simulation

One of the key components to make this simulator is a platform. We chose to make in Python a small game system that basically automates the job we do in Mace4.

There are 2 types of files that are used by the Python program, a template file that is a text file with the ".in" extension - it consists of the general assumptions regarding the Among Us game, and a JSON data file - it contains the data for each game such as players, tasks, locations, available messages.

The program works by first loading the data from the JSON file, after that it generates the additional assumptions that are based on the loaded data, creates the players and randomized their properties - name, location, role, task and generates their messages.

When the game is run, all the generated data is written in the template and is given to mace4 to compute the models, the output is then interpreted by the Python program and computed a statistic regarding the percentages of each player being an imposter. The user is then presented with the messages of each player and then given the chance to choose who is the imposter with the help of mace4. If the imposter is successfully detected, the crewmates win, otherwise the imposter wins.

Testing:

Now that everything is implemented and tested individually, let us see how everything fares when we put it all together!

We conducted 4 separate tests, all with 3 people involved in the game. Each and every game property is generated randomly, each player having a different status each game. All that is always the same is the map (locations) and tasks assignation (each room will always have the same tasks assigned to it).

We set the maximum number of models to be 100,000. We surely could have less or a lot more, but considering everything is randomised, we do not want Mace4 to give us all the millions of models. We want to have an approximately good idea of the probability each person is an impostor.

Test 1:


message(black)<->deadAt(green, Cafeteria) & crewmate(green) & -imposter(green) & vent(red, Weapons).
message(green)<->-imposter(green) & imposter(black) & crewmate(green) & deadAt(red, Security).
Black: 100.0% to be imposter
Red: 0.0% to be imposter
Green: 0.0% to be imposter
Vote who you think is the imposter: black

Test 2:


message(green)<->didTask(black, FixWiring) & task(Cafeteria, FixWiring) & -deadAt(black, Weapons) & -vent(black, Security) & crewm
message(black)<->-imposter(black) & -deadAt(black, Weapons) & deadAt(red, Weapons) & didTask(green, FixWiring) & task(Security, Fi
Green: 100.0% to be imposter
Red: 0.0% to be imposter
Black: 0.0% to be imposter
Vote who you think is the imposter:
message(green)<->didTask(black, FixWiring) & task(Cafeteria, FixWiring) & -deadAt(black, Weapons) & -vent(black, Security) & crewm
message(black)<->-imposter(black) & -deadAt(black, Weapons) & deadAt(red, Weapons) & didTask(green, FixWiring) & task(Security, Fi
Green: 100.0% to be imposter
Red: 0.0% to be imposter
Black: 0.0% to be imposter
Vote who you think is the imposter: red

Test 3:


message(green)<->vent(black, Security) & didTask(black, FixWiring) & task(Weapons, FixWiring) & -crewmate(black) & imposter(black)
message(black)<->-imposter(black) & -deadAt(black, Weapons) & -crewmate(green) & deadAt(red, Weapons) & vent(green, Cafeteria) & i
Green: 100.0% to be imposter
Red: 0.0% to be imposter
Black: 0.0% to be imposter
Vote who you think is the imposter:
message(green)<->vent(black, Security) & didTask(black, FixWiring) & task(Weapons, FixWiring) & -crewmate(black) & imposter(black)
message(black)<->-imposter(black) & -deadAt(black, Weapons) & -crewmate(green) & deadAt(red, Weapons) & vent(green, Cafeteria) & i
Green: 100.0% to be imposter
Red: 0.0% to be imposter
Black: 0.0% to be imposter
Vote who you think is the imposter: green

Test 4:


message(red)<->-imposter(red) & vent(green, Cafeteria).
message(black)<->-vent(black, Security) & imposter(red) & -crewmate(red) & -imposter(black) & crewmate(black) & vent(red, Cafeteria).
Red: 30.72% to be imposter
Green: 0.0% to be imposter
Black: 69.28% to be imposter
Vote who you think is the imposter:

Conclusions:

This project was really enjoyable. As hard as it was to understand the language of Mace4 at first, the outcome was very satisfactory, being able to both interpret the models and write good code for Mace4. Being able to use 2 platforms (Mace4 and Python) for achieving a great project will surely prove to be a great asset in the future. Also, working with a partner, dividing tasks, helping each other and accomplishing tasks together was extremely fun and enjoyable.

Chapter 3

A3: Planning

Bibliography

- AI Courses on Moodle from Adrian Groza
- https://en.wikipedia.org/wiki/Depth-first_search
- https://en.wikipedia.org/wiki/Breadth-first_search
- https://en.wikipedia.org/wiki/A*_search_algorithm
- http://theory.stanford.edu/~amitp/GameProgramming/Variations.html
- https://cse.iitk.ac.in/users/cs365/2009/ppt/13jan_Aman.pdf
- https://stackoverflow.com/questions/9994913/pacman-what-kinds-of-heuristics-are-ma
  36404229#36404229
- http://ai.berkeley.edu/search.html
- https://philosophy.hku.hk/think/logic/knights.php

Appendix A

Your original code

Note: Depth First Search, Breadth First Search, Uniform Cost Search, A Star Search, Weighted A Star Search and Random Search were all implemented using the pseudocode of the respective algorithm, using original work and ideas for the implementation.

```
class CornersProblem(search.SearchProblem):
    """
    This search problem finds paths through all four corners of a layout.

    You must select a suitable state space and successor function
    """

    def __init__(self, startingGameState):
        """
        Stores the walls, pacman's starting position and corners.
        """
        self.walls = startingGameState.getWalls()
        self.startingPosition = startingGameState.getPacmanPosition()
        top, right = self.walls.height-2, self.walls.width-2
        self.corners = ((1,1), (1,top), (right, 1), (right, top))
        for corner in self.corners:
            if not startingGameState.hasFood(*corner):
                print 'Warning: no food in corner ' + str(corner)
        self._expanded = 0 # DO NOT CHANGE; Number of search nodes expanded
        # Please add any code here which you would like to use
        # in initializing the problem
        "*** YOUR CODE HERE ***"

    def getStartState(self):
        """
        Returns the start state (in your state space, not the full Pacman
    state
        space)
        """
        "*** YOUR CODE HERE ***"
        return (self.startingPosition, self.corners)

    def isGoalState(self, state):
        """
        Returns whether this search state is a goal state of the problem.
        """
        "*** YOUR CODE HERE ***"

        return len(state[1]) == 0
```

```
39
40      def getSuccessors(self, state):
41          """
42          Returns successor states, the actions they require, and a cost of 1.
43
44           As noted in search.py:
45              For a given state, this should return a list of triples, (
    successor,
46              action, stepCost), where 'successor' is a successor to the
    current
47              state, 'action' is the action required to get there, and '
    stepCost'
48              is the incremental cost of expanding to that successor
49          """
50
51          successors = []
52          for action in [Directions.NORTH, Directions.SOUTH, Directions.EAST,
    Directions.WEST]:
53              # Add a successor state to the successor list if the action is
    legal
54              x,y = state[0]
55              dx, dy = Actions.directionToVector(action)
56              nextx, nexty = int(x + dx), int(y + dy)
57              hitsWall = self.walls[nextx][nexty]
58              if not hitsWall:
59                  remainingGoals = tuple(element for element in state[1] if
    element != (nextx, nexty))
60                  nextState = ((nextx, nexty), remainingGoals)
61                  cost = 1
62                  successors.append((nextState, action, cost))
63              "*** YOUR CODE HERE ***"
64
65          self._expanded += 1 # DO NOT CHANGE
66          return successors
67
68      def getCostOfActions(self, actions):
69          """
70          Returns the cost of a particular sequence of actions.  If those
    actions
71          include an illegal move, return 999999.  This is implemented for you
    .
72          """
73          if actions == None: return 999999
74          x,y= self.startingPosition
75          for action in actions:
76              dx, dy = Actions.directionToVector(action)
77              x, y = int(x + dx), int(y + dy)
78              if self.walls[x][y]: return 999999
79          return len(actions)
80
81
82      def cornersHeuristic(state, problem):
83          """
84          A heuristic for the CornersProblem that you defined.
85
86            state:   The current search state
87                     (a data structure you chose in your search problem)
88
89            problem: The CornersProblem instance for this layout.
90
```

```
        This function should always return a number that is a lower bound on the
        shortest path from the state to a goal of the problem; i.e.  it should be
        admissible (as well as consistent).
        """
        corners = problem.corners # These are the corner coordinates
        walls = problem.walls # These are the walls of the maze, as a Grid (game.py)

        "*** YOUR CODE HERE ***"

        leng = 0

        x0, y0 = state[0]
        for element in state[1]:
            x1, y1 = element
            leng = max(abs(x1 - x0) + abs(y1 - y0), leng)

        return leng
```

Listing A.1: Find all Corners Problem

```
def foodHeuristic(state, problem):
    position, foodGrid = state
    minDistance = 99999
    maxDistance = -1
    nearestFood = None
    furthestFood = None

    for element in foodGrid.asList():
        distance = mazeDistance(position, element, problem.startingGameState)
        if distance < minDistance:
            minDistance = distance
            nearestFood = element
        if distance > maxDistance:
            maxDistance = distance
            furthestFood = element

    if nearestFood == None and furthestFood == None:
        return 0
    else:
        return minDistance + mazeDistance(nearestFood, furthestFood, problem.startingGameState)
```

Listing A.2: Eat all Food Problem

```
class DiagonalSearchProblem(search.SearchProblem):
    def __init__(self, gameState, costFn = lambda x: 1, goal=(1,1), start=None, warn=True, visualize=True):
        self.walls = gameState.getWalls()
        self.startState = gameState.getPacmanPosition()
        if start != None: self.startState = start
        self.goal = goal
        self.costFn = costFn
        self.visualize = visualize
```

```
        if warn and (gameState.getNumFood() != 1 or not gameState.hasFood(*goal)):
            print 'Warning: this does not look like a regular search maze'

        self._visited, self._visitedlist, self._expanded = {}, [], 0 # DO NOT CHANGE

    def getStartState(self):
        return self.startState

    def isGoalState(self, state):
        isGoal = state == self.goal

        # For display purposes only
        if isGoal and self.visualize:
            self._visitedlist.append(state)
            import __main__
            if '_display' in dir(__main__):
                if 'drawExpandedCells' in dir(__main__._display): # @UndefinedVariable
                    __main__._display.drawExpandedCells(self._visitedlist) # @UndefinedVariable

        return isGoal

    def getSuccessors(self, state):
        """
        Returns successor states, the actions they require, and a cost of 1.

         As noted in search.py:
             For a given state, this should return a list of triples,
         (successor, action, stepCost), where 'successor' is a
         successor to the current state, 'action' is the action
         required to get there, and 'stepCost' is the incremental
         cost of expanding to that successor
        """

        successors = []
        for action in [Directions.NORTH, Directions.SOUTH, Directions.EAST, Directions.WEST, Directions.NORTHEAST, Directions.NORTHWEST, Directions.SOUTHEAST, Directions.SOUTHWEST]:
            x,y = state
            dx, dy = Actions.directionToVector(action)
            nextx, nexty = int(x + dx), int(y + dy)
            if not self.walls[nextx][nexty]:
                nextState = (nextx, nexty)
                cost = self.costFn(nextState)
                successors.append( ( nextState, action, cost) )

        # Bookkeeping for display purposes
        self._expanded += 1 # DO NOT CHANGE
        if state not in self._visited:
            self._visited[state] = True
            self._visitedlist.append(state)

        return successors

    def getCostOfActions(self, actions):
        """
        Returns the cost of a particular sequence of actions. If those
```

```
    actions
        include an illegal move, return 999999.
        """
        if actions == None: return 999999
        x,y= self.getStartState()
        cost = 0
        for action in actions:
            # Check figure out the next state and see whether its' legal
            dx, dy = Actions.directionToVector(action)
            x, y = int(x + dx), int(y + dy)
            if self.walls[x][y]: return 999999
            cost += self.costFn((x,y))
        return cost
```

Listing A.3: Diagonal Search Problem

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                %%                %
% %%%% %%%%% %% %%%%% %%%% %
% %%%% %%%%% %% %%%%% %%%% %
% %%%% %%%%% %% %%%%% %%%% %
%                                  %
% %%%% %% %%%%%%%% %% %%%% %
% %%%% %% %%%%%%%% %% %%%% %
%        %%      %%      %%        %
%%%%%% %%%%% %% %%%%% %%%%%%
%      % %%              %% %      %
%      % %%              %% %      %
%%%%%% %% %%%%%%%% %% %%%%%%
%            %        %            %
%%%%%% %% %%%%%%%% %% %%%%%%
%      % %%              %% %      %
%      % %%              %% %      %
%%%%%% %% %%%%%%%% %% %%%%%%
%                %%                %
% %%%% %%%%% %% %%%%% %%%% %
% %%%% %%%%% %% %%%%% %%%% %
%    %%         P          %%    %
%%% %% %% %%%%%%%% %% %% %%%
%%% %% %% %%%%%%%% %% %% %%%
%        %%      %%      %%        %
% %%%%%%%%%% %% %%%%%%%%%% %
% %%%%%%%%%% %% %%%%%%%%%% %
%.                                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Listing A.4: Search Maze

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%............%%............%
%.%%%%.%%%%%.%%.%%%%%.%%%%.%
%.%%%%.%%%%%.%%.%%%%%.%%%%.%
%.%%%%.%%%%%.%%.%%%%%.%%%%.%
%..........................%
%.%%%%.%%.%%%%%%%%.%%.%%%%.%
%.%%%%.%%.%%%%%%%%.%%.%%%%.%
%......%%....%%....%%......%
```

```
%%%%%%.%%%%%  %%  %%%%%.%%%%%%
%       %.%%               %%.%       %
%       %.%%               %%.%       %
%%%%%%.%%  %%%%%%%%  %%.%%%%%%
%        .     %          %     .        %
%%%%%%.%%  %%%%%%%%  %%.%%%%%%
%       %.%%               %%.%       %
%       %.%%               %%.%       %
%%%%%%.%%  %%%%%%%%  %%.%%%%%%
%.............%%.............%
%.%%%%.%%%%%.%%.%%%%%.%%%%.%
%.%%%%.%%%%%.%%.%%%%%.%%%%.%
%...%%.......P.........%%...%
%%%.%%.%%.%%%%%%%%%.%%.%%.%%%
%%%.%%.%%.%%%%%%%%%.%%.%%.%%%
%......%%....%%....%%......%
%.%%%%%%%%%%.%%.%%%%%%%%%%.%
%.%%%%%%%%%%.%%.%%%%%%%%%%.%
%.........................%
%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Listing A.5: Food Maze

```
%%%%%%%%%%%%%%%%%%%%%%%%%%
%.                  %%                  .%
%  %%%%  %%%%%  %%  %%%%%  %%%%  %
%  %%%%  %%%%%  %%  %%%%%  %%%%  %
%  %%%%  %%%%%  %%  %%%%%  %%%%  %
%                                      %
%  %%%%  %%  %%%%%%%%  %%  %%%%  %
%  %%%%  %%  %%%%%%%%  %%  %%%%  %
%          %%       %%       %%          %
%%%%%%%  %%%%%  %%  %%%%%  %%%%%%%
%       %  %%               %%  %       %
%       %  %%               %%  %       %
%%%%%%%  %%  %%%%%%%%  %%  %%%%%%%
%              %          %              %
%%%%%%%  %%  %%%%%%%%  %%  %%%%%%%
%       %  %%               %%  %       %
%       %  %%               %%  %       %
%%%%%%%  %%  %%%%%%%%  %%  %%%%%%%
%                  %%                  %
%  %%%%  %%%%%  %%  %%%%%  %%%%  %
%  %%%%  %%%%%  %%  %%%%%  %%%%  %
%     %%           P           %%     %
%%%  %%  %%  %%%%%%%%  %%  %%  %%%
%%%  %%  %%  %%%%%%%%  %%  %%  %%%
%          %%       %%       %%          %
%  %%%%%%%%%%  %%  %%%%%%%%%%  %
%  %%%%%%%%%%  %%  %%%%%%%%%%  %
%.                                    .%
%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Listing A.6: Corners Maze

The following is the Eight Puzzle class. We modified something in every function of this class, as an example the size of the maze.

```
    Eight puzzle:

# eightpuzzle.py
# --------------
# Licensing Information:  You are free to use or extend these projects for
# educational purposes provided that (1) you do not distribute or publish
# solutions, (2) you retain this notice, and (3) you provide clear
# attribution to UC Berkeley, including a link to http://ai.berkeley.edu.
#
# Attribution Information: The Pacman AI projects were developed at UC
   Berkeley.
# The core projects and autograders were primarily created by John DeNero
# (denero@cs.berkeley.edu) and Dan Klein (klein@cs.berkeley.edu).
# Student side autograding was added by Brad Miller, Nick Hay, and
# Pieter Abbeel (pabbeel@cs.berkeley.edu).


import search
import random
import time
import sys
from optparse import OptionParser

# Module Classes

size = 3

class EightPuzzleState:
    """
    The Eight Puzzle is described in the course textbook on
    page 64.

    This class defines the mechanics of the puzzle itself.  The
    task of recasting this puzzle as a search problem is left to
    the EightPuzzleSearchProblem class.
    """

    def __init__( self, numbers ):
        """
          Constructs a new eight puzzle from an ordering of numbers.

        numbers: a list of integers from 0 to 8 representing an
          instance of the eight puzzle.  0 represents the blank
          space.  Thus, the list

            [1, 0, 2, 3, 4, 5, 6, 7, 8]

          represents the eight puzzle:
            -------------
            | 1 |   | 2 |
            -------------
            | 3 | 4 | 5 |
            -------------
            | 6 | 7 | 8 |
            ------------

        The configuration of the puzzle is stored in a 2-dimensional
        list (a list of lists) 'cells'.
```

```
        """
        self.cells = []
        numbers = numbers[:] # Make a copy so as not to cause side-effects.
        numbers.reverse()
        for row in range(size):
            self.cells.append( [] )
            for col in range(size):
                self.cells[row].append( numbers.pop() )
                if self.cells[row][col] == 0:
                    self.blankLocation = row, col

    def isGoal( self ):
        """
          Checks to see if the puzzle is in its goal state.

            -------------
            |   | 1 | 2 |
            -------------
            | 3 | 4 | 5 |
            -------------
            | 6 | 7 | 8 |
            -------------

        >>> EightPuzzleState([0, 1, 2, 3, 4, 5, 6, 7, 8]).isGoal()
        True

        >>> EightPuzzleState([1, 0, 2, 3, 4, 5, 6, 7, 8]).isGoal()
        False
        """
        current = 0
        for row in range(size):
            for col in range(size):
                if current != self.cells[row][col]:
                    return False
                current += 1
        return True

    def legalMoves( self ):
        """
          Returns a list of legal moves from the current state.

        Moves consist of moving the blank space up, down, left or right.
        These are encoded as 'up', 'down', 'left' and 'right' respectively.

        >>> EightPuzzleState([0, 1, 2, 3, 4, 5, 6, 7, 8]).legalMoves()
        ['down', 'right']
        """
        moves = []
        row, col = self.blankLocation
        if(row != 0):
            moves.append('up')
        if(row != size - 1):
            moves.append('down')
        if(col != 0):
            moves.append('left')
        if(col != size - 1):
            moves.append('right')
        return moves

    def result(self, move):
```

```
        """
          Returns a new eightPuzzle with the current state and blankLocation
        updated based on the provided move.

        The move should be a string drawn from a list returned by legalMoves
    .
        Illegal moves will raise an exception, which may be an array bounds
        exception.

        NOTE: This function *does not* change the current object.  Instead,
        it returns a new object.
        """
        row, col = self.blankLocation
        if(move == 'up'):
            newrow = row - 1
            newcol = col
        elif(move == 'down'):
            newrow = row + 1
            newcol = col
        elif(move == 'left'):
            newrow = row
            newcol = col - 1
        elif(move == 'right'):
            newrow = row
            newcol = col + 1
        else:
            raise "Illegal Move"

        # Create a copy of the current eightPuzzle
        newPuzzle = EightPuzzleState([0 for _ in range(size * size)])
        newPuzzle.cells = [values[:] for values in self.cells]
        # And update it to reflect the move
        newPuzzle.cells[row][col] = self.cells[newrow][newcol]
        newPuzzle.cells[newrow][newcol] = self.cells[row][col]
        newPuzzle.blankLocation = newrow, newcol

        return newPuzzle

    # Utilities for comparison and display
    def __eq__(self, other):
        """
            Overloads '==' such that two eightPuzzles with the same
    configuration
          are equal.

          >>> EightPuzzleState([0, 1, 2, 3, 4, 5, 6, 7, 8]) == \
              EightPuzzleState([1, 0, 2, 3, 4, 5, 6, 7, 8]).result('left')
          True
        """
        for row in range(size):
            if self.cells[row] != other.cells[row]:
                return False
        return True

    def __hash__(self):
        return hash(str(self.cells))

    def __getAsciiString(self):
        """
          Returns a display string for the maze
```

```
        """
        lines = []
        horizontalLine = ('-' * (4 * size + 1))
        lines.append(horizontalLine)
        for row in self.cells:
            rowLine = '|'
            for col in row:
                if col == 0:
                    col = ' '
                rowLine = rowLine + ' ' + col.__str__() + ' |'
            lines.append(rowLine)
            lines.append(horizontalLine)
        return '\n'.join(lines)

    def __str__(self):
        return self.__getAsciiString()

class EightPuzzleSearchProblem(search.SearchProblem):
    """
      Implementation of a SearchProblem for the  Eight Puzzle domain

      Each state is represented by an instance of an eightPuzzle.
    """
    def __init__(self,puzzle):
        "Creates a new EightPuzzleSearchProblem which stores search information."
        self.expanded = 0
        self.puzzle = puzzle

    def getStartState(self):
        return self.puzzle

    def isGoalState(self,state):
        return state.isGoal()

    def getSuccessors(self,state):
        """
          Returns list of (successor, action, stepCost) pairs where
          each succesor is either left, right, up, or down
          from the original state and the cost is 1.0 for each
        """
        self.expanded += 1
        succ = []
        for a in state.legalMoves():
            succ.append((state.result(a), a, 1))
        return succ

    def getCostOfActions(self, actions):
        """
         actions: A list of actions to take

        This method returns the total cost of a particular sequence of actions.  The sequence must
        be composed of legal moves
        """
        return len(actions)

EIGHT_PUZZLE_DATA = [[1, 0, 2, 3, 4, 5, 6, 7, 8],
                     [1, 7, 8, 2, 3, 4, 5, 6, 0],
                     [4, 3, 2, 7, 0, 5, 1, 6, 8],
```

```
                      [5, 1, 3, 4, 0, 2, 6, 7, 8],
                      [1, 2, 5, 7, 6, 8, 0, 4, 3],
                      [0, 3, 1, 6, 8, 2, 7, 5, 4]]

def loadEightPuzzle(puzzleNumber):
    """
      puzzleNumber: The number of the eight puzzle to load.

      Returns an eight puzzle object generated from one of the
      provided puzzles in EIGHT_PUZZLE_DATA.

      puzzleNumber can range from 0 to 5.

      >>> print loadEightPuzzle(0)
      -------------
      | 1 |   | 2 |
      -------------
      | 3 | 4 | 5 |
      -------------
      | 6 | 7 | 8 |
      -------------
    """
    return EightPuzzleState(EIGHT_PUZZLE_DATA[puzzleNumber])

def createRandomEightPuzzle(moves=100):
    """
      moves: number of random moves to apply

      Creates a random eight puzzle by applying
      a series of 'moves' random moves to a solved
      puzzle.
    """
    puzzle = EightPuzzleState([i for i in range(size * size)])
    for i in range(moves):
        # Execute a random legal move
        puzzle = puzzle.result(random.sample(puzzle.legalMoves(), 1)[0])
    return puzzle

def nullHeuristic(state, problem=None):
    """
    A heuristic function estimates the cost from the current state to the
   nearest
    goal in the provided SearchProblem.  This heuristic is trivial.
    """

    return 0

def tileMisplacedHeuristic(state, problem = None):
    count = 0
    current = 0
    for row in range(size):
        for col in range(size):
            if current != state.cells[row][col]:
                count += 1
            current += 1
    return count

def manhattanDistance(position1, position2):
    xy1 = position1
    xy2 = position2
```

```
293     return abs(xy1[0] - xy2[0]) + abs(xy1[1] - xy2[1])
294 
295 def manhattanDistanceToCorrectPositionHeuristic(state, problem = None):
296     coordinates = [(x, y) for x in range(size) for y in range(size)]
297 
298     total_distance = 0
299 
300     current = 0
301     for row in range(size):
302         for col in range(size):
303             if current != state.cells[row][col]:
304                 total_distance += manhattanDistance(coordinates[state.cells[
    row][col]], (row, col))
305             current += 1
306 
307     return total_distance
308 
309 def outOfColumnRowHeuristic(state, problem = None):
310     coordinates = [(x, y) for x in range(size) for y in range(size)]
311 
312     outOfRow = 0
313     outOfColumn = 0
314 
315     current = 0
316     for row in range(size):
317         for col in range(size):
318             if current != state.cells[row][col]:
319                 if row != coordinates[state.cells[row][col]][0]:
320                     outOfRow += 1
321                 if col != coordinates[state.cells[row][col]][1]:
322                     outOfColumn += 1
323 
324     return outOfRow + outOfColumn
325 
326 def euclideanDistanceToCorrectPositionHeuristic(state, problem = None):
327     total_distance = 0
328     current = 0
329     for row in range (size):
330         for col in range (size):
331             if current != state.cells[row][col] :
332                 goal_row, goal_col = divmod(state.cells[row][col], 3)
333                 total_distance += ((row - goal_row) ** 2 + (col - goal_col)
    ** 2) ** 0.5
334             current += 1
335     return total_distance
336 
337 def swapHeuristic(state, problem):
338     coordinates = [(x, y) for x in range(size) for y in range(size)]
339 
340     total_cost = 0
341     cells2 = [row [:] for row in state.cells]
342 
343     for row in range (size):
344         for col in range (size):
345             x,y = coordinates[cells2[row][col]]
346             if (x != row) or (y != col):
347                 aux = cells2[row][col]
348                 cells2[row][col] = cells2[x][y]
349                 cells2[x][y] = aux
350                 total_cost += 1
```

```
351     return total_cost
352 
353 def StalinSort():
354     puzzle = createRandomEightPuzzle(100)
355     print('A random puzzle:')
356     print(puzzle)
357 
358     problem = EightPuzzleSearchProblem(puzzle)
359     current = 0
360     for row in range (size):
361         for col in range (size):
362             if current != puzzle.cells[row][col]:
363                 puzzle.cells[row][col] = 0
364             else:
365                 puzzle.cells[row][col] = current
366             current += 1
367 
368     print("\nStalin's hand falls upon this puzzle and it now becomes:")
369     print(puzzle)
370 
371 def readCommand(args):
372     parser = OptionParser()
373     parser.add_option("--heuristic", dest='heuristic', action="store", type=
    "string", default="null")
374     parser.add_option("-s", "--size", dest="size", action="store", type="int
    ", default=3)
375     parser.add_option("-f", "--function", dest="function", action="store",
    type="string", default="astar")
376 
377     options, arg = parser.parse_args(args)
378 
379     if len(arg) != 0:
380         print("Commands not understood")
381         return
382 
383     heuristic = None
384 
385     if options.heuristic == "manhattan":
386         heuristic = manhattanDistanceToCorrectPositionHeuristic
387     elif options.heuristic == "euclidian":
388         heuristic = euclideanDistanceToCorrectPositionHeuristic
389     elif options.heuristic == "tileMisplaced":
390         heuristic = tileMisplacedHeuristic
391     elif options.heuristic == "outOfColumnRow":
392         heuristic = outOfColumnRowHeuristic
393     elif options.heuristic == "swap":
394         heuristic = swapHeuristic
395     elif options.heuristic == "null":
396         heuristic = nullHeuristic
397     else:
398         print("No such heuristic found")
399         return
400 
401     func = None
402     if options.function == "StalinSort":
403         StalinSort()
404         return
405     elif options.function == "wastar":
406         func = search.weightedAStarSearch
407     elif options.function == "astar":
```

```
        func = search.aStarSearch
    else:
        print("That function doesn't exist")
        return

    global size
    size = options.size

    runGame(func, heuristic)

def runGame(func, heuristic):
    puzzle = createRandomEightPuzzle(30)
    #puzzle = EightPuzzleState([7, 2, 4, 5, 0, 6, 8, 3, 1])
    # puzzle = EightPuzzleState([8, 7, 6, 5, 4, 3, 2, 1, 0])
    # puzzle = EightPuzzleState([5, 0, 8, 1, 4, 2, 3, 6, 7])
    print(puzzle)
    problem = EightPuzzleSearchProblem(puzzle)
    start = time.time()
    path = func(problem=problem, heuristic=heuristic)
    end = time.time()
    print(end - start)
    print("Expanded nodes: %d" % problem.expanded)
    print('A* found a path of %d moves: %s' % (len(path), str(path)))
    curr = puzzle
    i = 1
    for a in path:
        curr = curr.result(a)
        print('After %d move%s: %s' % (i, ("", "s")[i>1], a))
        print(curr)

        raw_input("Press return for the next state...")   # wait for key
    stroke
        i += 1

if __name__ == '__main__':
    args = readCommand(sys.argv[1:])
```

Listing A.7: Eight Puzzle Problem Class

```
  class Directions:
    NORTH = 'North'
    SOUTH = 'South'
    EAST = 'East'
    WEST = 'West'
    STOP = 'Stop'
    NORTHEAST = 'NorthEast'
    NORTHWEST = 'NorthWest'
    SOUTHEAST = 'SouthEast'
    SOUTHWEST = 'SouthWest'

    LEFT =       {NORTH: WEST,
                  SOUTH: EAST,
                  EAST:  NORTH,
                  WEST:  SOUTH,
                  STOP:  STOP}

    RIGHT =      dict([(y,x) for x, y in LEFT.items()])

    REVERSE = {NORTH: SOUTH,
```

```
                SOUTH: NORTH,
                EAST: WEST,
                WEST: EAST,
                STOP: STOP}

class Actions:
    """
    A collection of static methods for manipulating move actions.
    """
    # Directions
    _directions = {Directions.NORTH: (0, 1),
                   Directions.SOUTH: (0, -1),
                   Directions.EAST:  (1, 0),
                   Directions.WEST:  (-1, 0),
                   Directions.STOP:  (0, 0),
                   Directions.NORTHEAST: (1, 1),
                   Directions.NORTHWEST: (-1, 1),
                   Directions.SOUTHEAST: (1, -1),
                   Directions.SOUTHWEST: (-1, -1)}

    _directionsAsList = _directions.items()

    ............
```

Listing A.8: game.py file modifications

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

formulas(vrajeala).
    all x (inhabitant(x) -> knight(x) | knave(x)).
    all x ((knight(x) -> -knave(x)) & (knave(x) -> -knight(x))).
    knight(x) -> message(x).
    knave(x) -> -message(x).
end_of_list.

formulas(nu_vrajeala).
    % Zoey = a & Mel = b
    inhabitant(Zoey) & inhabitant(Mel).
    message(Zoey) <-> knave(Mel).
    message(Mel) <-> (-knave(Mel) & -knave(Zoey)).

    % inhabitant(z) & inhabitant(m).
    % message(z) <-> knave(m).
    % message(m) <-> (-knave(m) & -knave(z)).
end_of_list.
```

Listing A.9: KK Problem1

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

formulas(vrajeala).
    all x (inhabitant(x) -> knight(x) | knave(x)).
```

```
    all x ((knight(x) -> -knave(x)) & (knave(x) -> -knight(x))).
    knight(x) -> message(x).
    knave(x) -> -message(x).
end_of_list.

formulas(nu_vrajeala).
    inhabitant(Peggy) & inhabitant(Zippy).
    Peggy=0 & Zippy=1.
    message(Peggy) <-> (knight(Peggy) & knave(Zippy)) | (knight(Zippy) &
   knave(Peggy)).
    message(Zippy) <-> ((message(Zippy) <-> knave(Peggy)) <-> knave(Zippy))
   & ((message(Peggy) <-> knave(Peggy)) <-> knave(Peggy)).
    % message(Zippy) <-> (all x ((message(x) <-> knave(Peggy)) -> knave(x)))
    .
end_of_list.
```

Listing A.10: KK Problem2

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

formulas(vrajeala).
    all x (inhabitant(x) -> knight(x) | knave(x)).
    all x ((knight(x) -> -knave(x)) & (knave(x) -> -knight(x))).
    knight(x) -> message(x).
    knave(x) -> -message(x).
end_of_list.

formulas(nu_vrajeala).
    inhabitant(Sue) & inhabitant(Zippy).
    Sue=0 & Zippy=1.
    message(Sue) <-> knave(Zippy).
    message(Zippy) <-> knight(Zippy) & knight(Sue).
end_of_list.
```

Listing A.11: KK Problem3

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

formulas(vrajeala).
    all x (inhabitant(x) -> knight(x) | knave(x)).
    all x ((knight(x) -> -knave(x)) & (knave(x) -> -knight(x))).
    knight(x) -> message(x).
    knave(x) -> -message(x).
end_of_list.

formulas(nu_vrajeala).
    inhabitant(Sally) & inhabitant(Zippy).
    Sally=0 & Zippy=1.
    message(Sally) <-> (knight(Sally) & -knight(Zippy)) | (knight(Zippy) & -
   knight(Sally)).
    message(Zippy) <-> (knight(Sally) & knave(Zippy)) | (knight(Zippy) &
   knave(Sally)).
end_of_list.
```

Listing A.12: KK Problem4

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

formulas(vrajeala).
    all x (inhabitant(x) -> knight(x) | knave(x)).
    all x ((knight(x) -> -knave(x)) & (knave(x) -> -knight(x))).
    knight(x) -> message(x).
    knave(x) -> -message(x).
end_of_list.

formulas(nu_vrajeala).
    inhabitant(Homer) & inhabitant(Bozo).
    Homer=0 & Bozo=1.
    message(Homer) <-> (knight(Homer) | knight(Bozo) | (knight(Bozo) &
   knight(Homer))).
    message(Bozo) <-> (message(Homer) <-> knave(Bozo)) | (-message(Homer)
   <-> knave(Bozo)).
end_of_list.
```

Listing A.13: KK Problem5

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

formulas(vrajeala).
    all x (inhabitant(x) -> knight(x) | knave(x)).
    all x ((knight(x) -> -knave(x)) & (knave(x) -> -knight(x))).
    knight(x) -> message(x).
    knave(x) -> -message(x).
end_of_list.

formulas(nu_vrajeala).
    inhabitant(Bob) & inhabitant(Mel).
    Bob=0 & Mel=1.
    message(Bob) <-> knight(Bob) | knave(Mel) | (knight(Bob) & knave(Mel)).
    message(Mel) <-> ((message(Bob) <-> knave(Bob)) <-> knave(Bob)) & ((
   message(Mel) <-> knave(Bob)) <-> knave(Mel)).
end_of_list.
```

Listing A.14: KK Problem16

```
assign(max_models, -1).
assign(domain_size, 3).

formulas(vrajeala).
    all x (inhabitant(x) -> knight(x) | knave(x)).
    all x ((knight(x) -> -knave(x)) & (knave(x) -> -knight(x))).
    knight(x) -> message(x).
    knave(x) -> -message(x).
end_of_list.

formulas(nu_vrajeala).
    inhabitant(Bill) & inhabitant(Bart) & inhabitant(Mel).
    Bill=0 & Bart=1 & Mel=2.
    message(Bill) <-> -knave(Bart).
    message(Bart) <-> (knight(Bart) & knave(Mel)) | (knight(Mel) & knave(
   Bart)).
```

```
    message(Mel) <-> (knight(Bart) & knight(Mel)) | (knave(Bart) & knave(Mel
   )).
end_of_list.
```

Listing A.15: KK Problem77

```
assign(max_models, -1).
assign(domain_size, 9).

formulas(vrajeala).
    all x (inhabitant(x) -> knight(x) | knave(x)).
    all x ((knight(x) -> -knave(x)) & (knave(x) -> -knight(x))).
    knight(x) -> message(x).
    knave(x) -> -message(x).
end_of_list.

formulas(nu_vrajeala).
    inhabitant(Joe) & inhabitant(Homer) & inhabitant(Bozo) & inhabitant(
   Betty) & inhabitant(Mel) & inhabitant(Peggy) & inhabitant(Rex) &
   inhabitant(Carl) & inhabitant(Marge).
    Joe=0 & Homer=1 & Bozo=2 & Betty=3 & Mel=4 & Peggy=5 & Rex=6 & Carl=7 &
   Marge=8.
    message(Joe) <-> knave(Homer).
    message(Homer) <-> (-knave(Peggy)).
    message(Bozo) <-> knave(Betty).
    message(Betty) <-> (knave(Rex) | knave(Carl) | (knave(Rex) & knave(Carl)
   )).
    message(Mel) <-> (knight(Carl) & knight(Homer)).
    message(Peggy) <-> (-knave(Rex)).
    message(Rex) <-> knave(Joe) & knave(Carl).
    message(Carl) <-> (knight(Marge) | knave(Joe)).
    message(Marge) <-> knave(Rex).
end_of_list.
```

Listing A.16: KK Problem379

```
assign(max_models, -1).
assign(domain_size, 3).

formulas(vrajeala).
    all x (inhabitant(x) -> knight(x) | knave(x) | spy(x)).
    all x ((knight(x) -> -knave(x) & -spy(x)) & (knave(x) -> -knight(x) & -
   spy(x)) & (spy(x) -> -knight(x) & -knave(x))).
    knight(x) -> message(x).
    knave(x) -> -message(x).
    spy(x) -> message(x) | (-message(x)).
end_of_list.

formulas(nu_vrajeala).
    inhabitant(Andrew) & inhabitant(Billy) & inhabitant(Charly).
    Andrew=0 & Billy=1 & Charly=2.
    message(Andrew) <-> knave(Charly).
    message(Billy) <-> knight(Andrew).
    message(Charly) <-> spy(Charly).

    %message(Andrew) <-> spy(Billy).
    %message(Charly) <-> knave(Andrew).
```

```
end_of_list.
```

Listing A.17: KKS Problem1

```
assign(max_models, -1).
assign(domain_size, 4).

formulas(vrajeala).
    all x (inhabitant(x) -> knight(x) | knave(x) | spy(x)).
    all x ((knight(x) -> -knave(x) & -spy(x)) & (knave(x) -> -knight(x) & -spy(x)) & (spy(x) -> -knight(x) & -knave(x))).
    knight(x) -> message(x).
    knave(x) -> -message(x).
    spy(x) -> message(x) | (-message(x)).
end_of_list.

formulas(nu_vrajeala).
    inhabitant(Alice) & inhabitant(Bob) & inhabitant(Charlie) & inhabitant(Dylan).
    Alice=0 & Bob=1 & Charlie=2 & Dylan=3.

    message(Alice) <-> (knight(Alice) & knave(Charlie)) | (knave(Bob) & spy(Dylan)).
    message(Bob) <-> knave(Alice) & spy(Dylan).
    message(Charlie) <-> (knight(Charlie) & knight(Dylan)) | (knave(Charlie) & knave(Dylan)) | (spy(Charlie) & spy(Dylan)).
    message(Dylan) <-> spy(Bob) | knight(Bob).
end_of_list.
```

Listing A.18: KKS Problem2

```

all x (player(x) -> crewmate(x) | impostor(x)).
all x ((crewmate(x) -> -impostor(x)) & (impostor(x) -> -crewmate(x))).

player(player0) & player(player1).
player0 = 0 & player1 = 1.

crewmate(x) -> message(x).
impostor(x) -> -message(x) | message(x).

%Case 1: We try with messages of accusation
message(player0) <-> impostor(player1).
message(player1) <-> (-impostor(player1) & -impostor(player0)).
```

Listing A.19: Accusation System + Testing

```

all x (location(x) -> task(x, y)).

Cafeteria = 0 & Weapons = 1.
FixWiring = 0 & EmptyGarbage = 1.

location(Cafeteria) | location(Weapons).

```

```
task(Cafeteria, FixWiring) & task(Cafeteria, EmptyGarbage).
task(Weapons, FixWiring) & -task(Weapons, EmptyGarbage).

all x (crewmate(x) -> didTask(x, Location)).
all x (impostor(x) -> didTask(x, Location) | -didTask(x, Location)).

didTask(player0, Cafeteria) & -didTask(player0, Weapons) | -didTask(player0, Cafeteria) & didTask(player0, Weapons).
didTask(player1, Cafeteria) & -didTask(player1, Weapons) | -didTask(player1, Cafeteria) & didTask(player1, Weapons).

%Case 2: We try with tasks
message(player0) <-> didTask(player0, EmptyGarbage) & task(Cafeteria, EmptyGarbage).
message(player1) <-> didTask(player1, FixWiring) & task(Cafeteria, FixWiring).
```

Listing A.20: Doing Tasks System + Testing

```

all x (player(x) -> seenAt(x, Location)).

(seenAt(player0, Cafeteria) & -seenAt(player0, Weapons)) | (-seenAt(player0, Cafeteria) & seenAt(player0, Weapons)).
(seenAt(player1, Cafeteria) & -seenAt(player1, Weapons)) | (-seenAt(player1, Cafeteria) & seenAt(player1, Weapons)).

all x exists y ((deadAt(y, Location) & seenAt(x, Location)) -> impostor(x)).

%Case 3: We try with a dead body at a certain location and if the imposor is or not there
deadAt(player0, Weapons).
deadAt(player0, Cafeteria).
-deadAt(player1, Weapons).
-deadAt(player1, Cafeteria).
seenAt(player1, Weapons).
seenAt(player0, Weapons).

didTask(player1, Cafeteria) & task(Cafeteria, FixWiring).
didTask(player0, Weapons) & task(Weapons, FixWiring).
```

Listing A.21: Near Dead Body System + Testing

```

all x (player(x) -> vent(x, Location) | -vent(x, Location)).

vent(player0, Cafeteria) | vent(player0, Weapons) | -vent(player0, Cafeteria) | -vent(player0, Weapons).
vent(player1, Cafeteria) | vent(player1, Weapons) | -vent(player1, Cafeteria) | -vent(player1, Weapons).

all x ((vent(x, Location) & seenVenting(x, Location)) -> impostor(x)).
seenVenting(player0, Cafeteria) | seenVenting(player0, Weapons) | -seenVenting(player0, Cafeteria) | -seenVenting(player0, Weapons).
seenVenting(player1, Cafeteria) | seenVenting(player1, Weapons) | -seenVenting(player1, Cafeteria) | -seenVenting(player1, Weapons).

```

```
%Case 4: We try with vents.
didTask(player0, Cafeteria) & task(Cafeteria, FixWiring). % scriem astea ca sa nu fie mai multe scenarii in care ba is task-uri facute, ba nu
didTask(player1, Weapons) & task(Weapons, FixWiring).

-deadAt(player0, Weapons).
-deadAt(player0, Cafeteria).
-deadAt(player1, Weapons).
-deadAt(player1, Cafeteria). %asta e una -vedem de fixam, daca nu lasam in python sa fie facute

-seenVenting(player0, Cafeteria).
-seenVenting(player0, Weapons).
-seenVenting(player1, Cafeteria).
seenVenting(player1, Weapons).
-vent(player1, Cafeteria).
vent(player1, Weapons).
-vent(player0, Cafeteria).
-vent(player0, Weapons).
```

Listing A.22: Venting System + Testing

```
import random
import json
import subprocess
import re

TITLE = """
 _____ __  __ _____ ____    ____ _____ _    _  ______    __
| ____|  \\/  | ____|  _ \\ / ___| ____| \\ | |/ ___\\ \\ / /
|  _| | |\\/| |  _| | |_) | |  _|  _| |  \\| | |    \\ V /
| |___| |  | | |___|  _ <| |_| | |___| |\\  | |___  | |
|_____|_|  |_|_____|_| \\_\\\\____|_____|_| \\_|\\____| |_|


 __  __ _____ _____ _____ ___ _    _  ____
|  \\/  | ____| ____|_   _|_ _| \\ | |/ ___|
| |\\/| |  _| |  _|   | |  | ||  \\| | |  _
| |  | | |___| |___  | |  | || |\\  | |_| |
|_|  |_|_____|_____| |_| |___|_| \\_|\\____|
"""

CREWMATES_WIN = """
  ____ ____  _______         ____  __    _  _____ _____ ____   __
    _____ _   _
 / ___|  _ \| ____\ \      / |  \/  |  / \|_   _| ____/ ___|  \ \      / |_
   _| \ | |
| |   | |_) |  _|  \ \ /\ / /| |\/| | / _ \ | | |  _| \___ \   \ \ /\ / / |
   ||  \| |
| |___|  _ <| |___  \ V  V / | |  | |/ ___ \| | | |___ ___) |   \ V  V /  |
   || |\  |
 \____|_| \_|_____|  \_/\_/  |_|  |_/_/   \_|_| |_____|____/     \_/\_/  |
   ___|_| \_|
"""

IMPOSTER_WINS = """
 ___ __  __ ____    ___ ____ _____ _____ ____   __       _____ _   _ ____
|_ _|  \/  |  _ \ / _ \/ ___|_   _| ____|  _ \  \ \      / |_ _| \ | / ___|
 | || |\/| | |_) | | | \___ \ | | |  _| | |_) |  \ \ /\ / / | ||  \| \___ \
```

```
 | || |  | |  __/| |_| |___) || | | |___|  _ <    \ V  V / | || |\  |___) |
|___|_|  |_|_|    \___/|____/ |_| |_____|_| \_\    \_/\_/  |___|_| \_|____/
"""

game = None

class Location:
    def __init__(self, name, tasks) -> None:
        self.name = name
        self.tasks = tasks

    def __eq__(self, __value: object) -> bool:
        return self.name == __value.name

class Player:
    def __init__(self, name) -> None:
        self.name = name
        self.location = None
        self.task = None

    def get_name(self):
        return self.name

    def get_message(self):
        final_message = "message({player})<->".format(player = self.name)
        global game

        messages = random.sample(game.get_messages(), random.randint(2, min
    (6, len(game.get_messages()))))

        for i in range(len(messages)):
            final_message += self.interpret_message(messages[i])

            if i == len(messages) - 1:
                final_message += ".\n\t"
            else:
                final_message += " & "

        return final_message

    def interpret_message(self, message):
        return ""

    def get_role(self):
        return ""

class Crewmate(Player):
    def __init__(self, name) -> None:
        super().__init__(name)

    def interpret_message(self, message):
        match message:
            case "crewmate({player})" | "-imposter({player})":
                return message.format(player = (random.choice([p for p in
    game.players if isinstance(p, Crewmate)])).get_name())
            case "imposter({player})" | "-crewmate({player})":
                return message.format(player = (random.choice([p for p in
    game.players if isinstance(p, Imposter)])).get_name())
            case "didTask({player}, {task}) & task({location}, {task})":
                player = random.choice([p for p in game.players if not
```

```
    isinstance(p, Body)])
90                  return message.format(player = player.get_name(), task =
    player.task, location = player.location.name)
91              case "deadAt({player}, {location})":
92                  player = [p for p in game.players if isinstance(p, Body)][0]
93                  return message.format(player = player.get_name(), location =
     player.location.name)
94              case "-deadAt({player}, {location})":
95                  player = random.choice([p for p in game.players if not
    isinstance(p, Body)])
96                  return message.format(player = player.get_name(), location =
     player.location.name)
97              case "vent({player}, {location})":
98                  player = random.choice([p for p in game.players if
    isinstance(p, Imposter)])
99                  return message.format(player = player.name, location =
    player.location.name)
100             case "-vent({player}, {location})":
101                 player = random.choice([p for p in game.players if
    isinstance(p, Crewmate)])
102                 return message.format(player = player.name, location =
    player.location.name)
103             case _:
104                 return ""
105
106     def get_role(self):
107         return "Crewmate"
108
109 class Imposter(Player):
110     def __init__(self, name) -> None:
111         super().__init__(name)
112
113     def interpret_message(self, message):
114         match message:
115             case "crewmate({player})" | "imposter({player})" | "-crewmate({
    player})" | "-imposter({player})":
116                 return message.format(player = (random.choice([p for p in
    game.players if not isinstance(p, Body)])).get_name())
117             case "didTask({player}, {task}) & task({location}, {task})":
118                 player = random.choice([p for p in game.players if not
    isinstance(p, Body)])
119                 return message.format(player = player.get_name(), task =
    random.choice(game.tasks), location = random.choice(game.locations).name)
120             case "deadAt({player}, {location})" | "-deadAt({player}, {
    location})":
121                 return message.format(player = random.choice(game.players).
    name, location = random.choice(game.locations).name)
122             case "vent({player}, {location})" | "-vent({player}, {location})
    ":
123                 return message.format(player = random.choice(game.players).
    name, location = random.choice(game.locations).name)
124             case _:
125                 return ""
126             # case "crewmate({player})" | "-imposter({player})":
127             #     return message.format(player = (random.choice([p for p in
    game.players if isinstance(p, Imposter)])).get_name())
128             # case "imposter({player})" | "-crewmate({player})":
129             #     return message.format(player = (random.choice([p for p in
    game.players if isinstance(p, Crewmate)])).get_name())
130
```

```
131
132     def get_role(self):
133         return "Imposter"
134
135 class Body(Player):
136     def __init__(self, name) -> None:
137         super().__init__(name)
138
139     def get_message(self):
140         return ""
141
142 class Game:
143     def __init__(self, path) -> None:
144         self.load_data(path)
145
146     def init_conditions(self):
147         message = ""
148
149         players_list1 = ""
150         players_list2 = ""
151
152         for i in range(len(self.players)):
153             players_list1 += "player({p_name})".format(p_name = self.players
    [i].get_name())
154             players_list2 += "{p_name} = {v}".format(p_name = self.players[i
    ].get_name(), v = i)
155
156             if i == len(self.players) - 1:
157                 players_list1 += ".\n\t"
158                 players_list2 += ".\n\t"
159             else:
160                 players_list1 += " & "
161                 players_list2 += " & "
162
163         message += players_list1 + players_list2
164
165         if hasattr(self, "locations"):
166             locations_list1 = ""
167             locations_list2 = ""
168             task_list1 = ""
169             for i in range(len(self.tasks)):
170                 task_list1 += "{task} = {v}".format(task = self.tasks[i], v
    = i)
171
172                 if i == len(self.tasks) - 1:
173                     task_list1 += ".\n\t"
174                 else:
175                     task_list1 += " & "
176
177             message += task_list1
178
179             for i in range(len(self.locations)):
180                 locations_list1 += "{location} = {v}".format(location = self
    .locations[i].name, v = i)
181                 locations_list2 += "location({location})".format(location =
    self.locations[i].name)
182                 if i == len(self.locations) - 1:
183                     locations_list1 += ".\n\t"
184                     locations_list2 += ".\n\t"
185                 else:
```

```
                    locations_list1 += " & "
                    locations_list2 += " | "

                task_list2 = ""

                for j in range(len(self.tasks)):
                    if self.tasks[j] in self.locations[i].tasks:
                        task_list2 += "task({location}, {task})".format(
location = self.locations[i].name, task = self.tasks[j])
                    else:
                        task_list2 += "-task({location}, {task})".format(
location = self.locations[i].name, task = self.tasks[j])

                    if j == len(self.tasks) - 1:
                        task_list2 += ".\n\t"
                    else:
                        task_list2 += " & "

                message += task_list2

            message += locations_list1 + locations_list2

        if hasattr(self, "dead_player"):
            for p in self.players:
                seen_messages = ""
                task_messages = ""
                for i in range(len(self.locations)):
                    seen_messages += "("
                    task_messages += "("
                    for j in range(len(self.locations)):
                        if self.locations[i] == self.locations[j]:
                            seen_messages += "seenAt({player}, {location})".
format(player = p.name, location = self.locations[j].name)
                            task_messages += "didTask({player}, {location})"
.format(player = p.name, location = self.locations[j].name)
                        else:
                            seen_messages += "-seenAt({player}, {location})"
.format(player = p.name, location = self.locations[j].name)
                            task_messages += "-didTask({player}, {location})
".format(player = p.name, location = self.locations[j].name)
                        if j != len(self.locations) - 1:
                            seen_messages += " & "
                            task_messages += " & "
                    seen_messages += ")"
                    task_messages += ")"
                    if i == len(self.locations) - 1:
                        seen_messages += ".\n\t"
                        task_messages += ".\n\t"
                    else:
                        seen_messages += " | "
                        task_messages += " | "
                message += seen_messages
                message += task_messages

            for p in self.players:
                seen_messages = ""
                dead_messages = ""
                for i in range(len(self.locations)):
                    loc = self.locations[i]

```

```
                    if p.location == loc:
                        seen_messages += "seenAt({player}, {location})".
    format(player = p.name, location = loc.name)
                    else:
                        seen_messages += "-seenAt({player}, {location})".
    format(player = p.name, location = loc.name)

                    if isinstance(p, Body) and p.location == loc:
                        dead_messages += "deadAt({player}, {location})".
    format(player = p.name, location = loc.name)
                    else:
                        dead_messages += "-deadAt({player}, {location})".
    format(player = p.name, location = loc.name)

                    if i == len(self.locations) - 1:
                        seen_messages += ".\n\t"
                        dead_messages += ".\n\t"
                    else:
                        seen_messages += " & "
                        dead_messages += " & "

                message += seen_messages
                message += dead_messages

        if hasattr(self, "vents"):
            for p in self.players:
                vent_messages = ""
                seen_messages = ""
                for i in range(len(self.locations)):
                    seen_messages += "seenVenting({player}, {location}) | -
    seenVenting({player}, {location})".format(player = p.name, location =
    self.locations[i].name)
                    vent_messages += "vent({player}, {location}) | -vent({
    player}, {location})".format(player = p.name, location = self.locations[i
    ].name)

                    if i == len(self.locations) - 1:
                        seen_messages += ".\n\t"
                        vent_messages += ".\n\t"
                    else:
                        seen_messages += " | "
                        vent_messages += " | "

                message += seen_messages
                message += vent_messages

            for p in self.players:
                seen_messages = ""
                for i in range(len(self.locations)):
                    if isinstance(p, Imposter) and p.location.name == self.
    locations[i].name:
                        seen_messages += "seenVenting({player}, {location})"
    .format(player = p.name, location = self.locations[i].name)
                    else:
                        seen_messages += "-seenVenting({player}, {location})
    ".format(player = p.name, location = self.locations[i].name)

                    if i == len(self.locations) - 1:
                        seen_messages += ".\n\t"
                    else:
```

```
                        seen_messages += " & "

                message += seen_messages

        return message

    def get_messages(self):
        return self.messages

    def print_players(self, percentages):
        for i in range(len(self.players)):
            print("{p_name}: {percent}% to be imposter".format(p_name = self
    .players[i].get_name().capitalize(), percent=percentages[i]))

    def load_data(self, path):
        f = open(path)
        data = json.load(f)
        f.close()

        self.max_models = data["max_models"]

        if "tasks" in data:
            self.tasks = data["tasks"]

        if "locations" in data:
            self.locations = []
            for loc in data["locations"]:
                new_loc = Location(loc["name"], loc["tasks"])
                self.locations.append(new_loc)

        players = random.sample(data["players"], len(data["players"]))

        self.messages = data["messages"]

        self.players = []

        self.players.append(Imposter(players.pop()))

        if data["dead_player"]:
            self.players.append(Body(players.pop()))
            self.dead_player = True

        for i in range(len(players)):
            self.players.append(Crewmate(players.pop()))

        for p in self.players:
            if "locations" in data:
                location = random.choice(self.locations)
                task = random.choice(location.tasks)

                p.location = location
                p.task = task

        if data["vents"]:
            self.vents = True

        f = open(data["template"], "r")
        self.template = f.read()
        f.close()

```

```
    def run_game(self):
        print(TITLE)

        conditions = self.init_conditions()
        messages = ""

        for p in self.players:
            m = p.get_message()
            print(m)
            #print("{player} is {role}".format(player = p.get_name(), role = p.get_role()))
            messages += m

        self.template = self.template.format(max_models = self.max_models, domain_size = len(self.players), game = conditions, messages = messages)

        f = open("out.in", "w")
        f.write(self.template)
        f.close()

        result = run_mace4()

        game.print_players(result)

        vote = input("Vote who you think is the imposter: ")
        while vote.lower() not in [player.get_name() for player in game.players]:
            vote = input("Not a player. Try again: ")

        game.players = [player for player in game.players if player.get_name() != vote.lower()]

        if any(isinstance(player, Imposter) for player in game.players):
            print(IMPOSTER_WINS)
        else:
            print(CREWMATES_WIN)

def run_mace4():
    result = subprocess.run(["mace4 | interpformat standard"], input=game.template, text=True, capture_output=True, shell=True)

    output = result.stdout

    line_imposter_regex = "relation\(imposter\(_\), \[(?:\d+,)+\d\]\)"
    array_regex = "\[\d+(?:,\d+)*\]"

    lines = re.findall(line_imposter_regex, output)
    result = [0] * len(game.players)

    for line in lines:
        array = eval(re.findall(array_regex, line)[0])
        result = [elem1 + elem2 for elem1, elem2 in zip(result, array)]

    result = [v * 100 / len(lines) for v in result]

    return result

if __name__ == "__main__":
    game = Game("data5.json")
```

```
403     game.run_game()
```

Listing A.23: game.py File

```
assign(max_models, {max_models}).
assign(domain_size, {domain_size}).

formulas(scenario).
    all x (player(x) -> crewmate(x) | imposter(x)).
    all x ((crewmate(x) -> -imposter(x)) & (imposter(x) -> -crewmate(x))).

    imposter(x) & imposter(y) -> x = y.
    (exists x crewmate(x)) & (exists x imposter(x)).

    crewmate(x) -> message(x).
    imposter(x) -> -message(x) | message(x).
end_of_list.

formulas(game).
    {game}
end_of_list.

formulas(messages).
    {messages}
end_of_list.
```

Listing A.24: template1.in File

```
{
    "max_models": -1,
    "dead_player": false,
    "vents": false,
    "players": ["red", "black"],
    "template": "template1.in",
    "messages": [
        "crewmate({player})",
        "imposter({player})",
        "-crewmate({player})",
        "-imposter({player})"
    ]
}
```

Listing A.25: data1.json File

```
assign(max_models, {max_models}).
assign(domain_size, {domain_size}).

formulas(scenario).
    all x (player(x) -> crewmate(x) | imposter(x)).
    all x ((crewmate(x) -> -imposter(x)) & (imposter(x) -> -crewmate(x))).
    all x (location(x) -> task(x, y)).

    imposter(x) & imposter(y) -> x = y.
    (exists x crewmate(x)) & (exists x imposter(x)).

```

```
    crewmate(x) -> message(x).
    imposter(x) -> -message(x) | message(x).
end_of_list.

formulas(game).
    {game}
end_of_list.

formulas(messages).
    {messages}
end_of_list.
```

Listing A.26: template2.in File

```
{
    "max_models": -1,
    "dead_player": false,
    "players": ["red", "black"],
    "vents": false,
    "tasks": [
        "EmptyGarbage",
        "FixWiring"
    ],
    "locations": [
        {
            "name": "Cafeteria",
            "tasks": [
                "EmptyGarbage",
                "FixWiring"
            ]
        },
        {
            "name": "Security",
            "tasks": [
                "FixWiring"
            ]
        }
    ],
    "template": "template2.in",
    "messages": [
        "didTask({player}, {task}) & task({location}, {task})"
    ]
}
```

Listing A.27: data2.json File

```
assign(max_models, {max_models}).
assign(domain_size, {domain_size}).

formulas(scenario).
    all x (player(x) -> crewmate(x) | imposter(x)).
    all x ((crewmate(x) -> -imposter(x)) & (imposter(x) -> -crewmate(x))).
    all x (location(x) -> task(x, Location)).

    imposter(x) & imposter(y) -> x = y.
    (exists x crewmate(x)) & (exists x imposter(x)).

```

```
    crewmate(x) -> message(x).
    imposter(x) -> -message(x) | message(x).

    all x (player(x) -> seenAt(x, Location) | -seenAt(x, Location)).

    all x exists y ((deadAt(y, Location) & seenAt(x, Location)) -> impostor(
   x)).

    all x (crewmate(x) -> didTask(x, Location)).
    all x (imposter(x) -> didTask(x, Location) | -didTask(x, Location)).
end_of_list.

formulas(game).
    {game}
end_of_list.

formulas(messages).
    {messages}
end_of_list.
```

Listing A.28: template3.in File

```
{
    "max_models": -1,
    "dead_player": true,
    "players": ["red", "black", "green"],
    "vents" : false,
    "tasks": [
        "EmptyGarbage",
        "FixWiring",
        "SwipeCard"
    ],
    "locations": [
        {
            "name": "Cafeteria",
            "tasks": [
                "EmptyGarbage",
                "FixWiring"
            ]
        },
        {
            "name": "Security",
            "tasks": [
                "FixWiring"
            ]
        },
        {
            "name": "Admin",
            "tasks": [
                "FixWiring",
                "SwipeCard"
            ]
        }
    ],
    "template": "template3.in",
    "messages": [
        "deadAt({player}, {location})",
        "-deadAt({player}, {location})",
        "didTask({player}, {task}) & task({location}, {task})"
```

```
    ]
}
```

Listing A.29: data3.json File

```
assign(max_models, {max_models}).
assign(domain_size, {domain_size}).

formulas(scenario).
    all x (player(x) -> crewmate(x) | imposter(x)).
    all x ((crewmate(x) -> -imposter(x)) & (imposter(x) -> -crewmate(x))).
    all x (location(x) -> task(x, Location)).

    imposter(x) & imposter(y) -> x = y.
    (exists x crewmate(x)) & (exists x imposter(x)).

    crewmate(x) -> message(x).
    imposter(x) -> -message(x) | message(x).

    all x (player(x) -> vent(x, Location) | -vent(x, Location)).
    all x ((vent(x, Location) & seenVenting(x, Location)) -> impostor(x)).

    all x (crewmate(x) -> didTask(x, Location)).
    all x (imposter(x) -> didTask(x, Location) | -didTask(x, Location)).
end_of_list.

formulas(game).
    {game}
end_of_list.

formulas(messages).
    {messages}
end_of_list.
```

Listing A.30: template4.in File

```
{
    "max_models": -1,
    "dead_player": false,
    "vents": true,
    "players": ["red", "black"],
    "tasks": [
        "EmptyGarbage",
        "FixWiring"
    ],
    "locations": [
        {
            "name": "Cafeteria",
            "tasks": [
                "EmptyGarbage",
                "FixWiring"
            ]
        },
        {
            "name": "Security",
            "tasks": [
                "FixWiring"
```

```
            ]
        }
    ],
    "template": "template4.in",
    "messages": [
        "didTask({player}, {task}) & task({location}, {task})",
        "vent({player}, {location})",
        "-vent({player}, {location})"
    ]
}
```

Listing A.31: data4.json File

```
assign(max_models, {max_models}).
assign(domain_size, {domain_size}).

formulas(scenario).
    all x (player(x) -> crewmate(x) | imposter(x)).
    all x ((crewmate(x) -> -imposter(x)) & (imposter(x) -> -crewmate(x))).
    all x (location(x) -> task(x, Location)).

    imposter(x) & imposter(y) -> x = y.
    (exists x crewmate(x)) & (exists x imposter(x)).

    crewmate(x) -> message(x).
    imposter(x) -> -message(x) | message(x).

    all x (player(x) -> vent(x, Location) | -vent(x, Location)).
    all x ((vent(x, Location) & seenVenting(x, Location)) -> impostor(x)).

    all x (player(x) -> seenAt(x, Location) | -seenAt(x, Location)).
    all x exists y ((deadAt(y, Location) & seenAt(x, Location)) -> impostor(
   x)).

    all x (crewmate(x) -> didTask(x, Location)).
    all x (imposter(x) -> didTask(x, Location) | -didTask(x, Location)).
end_of_list.

formulas(game).
    {game}
end_of_list.

formulas(messages).
    {messages}
end_of_list.
```

Listing A.32: template5.in File

```
{
    "max_models": 100000,
    "dead_player": true,
    "vents": true,
    "players": ["red", "black", "green"],
    "tasks": [
        "EmptyGarbage",
        "FixWiring",
        "AcceptDivertedPower"
```

```
    ],
    "locations": [
        {
            "name": "Cafeteria",
            "tasks": [
                "EmptyGarbage",
                "FixWiring",
                "DownloadData"
            ]
        },
        {
            "name": "Security",
            "tasks": [
                "FixWiring",
                "AcceptDivertedPower"
            ]
        },
        {
            "name": "Weapons",
            "tasks": [
                "AcceptDivertedPower"
            ]
        }
    ],
    "template": "template4.in",
    "messages": [
        "didTask({player}, {task}) & task({location}, {task})",
        "vent({player}, {location})",
        "-vent({player}, {location})",
        "deadAt({player}, {location})",
        "-deadAt({player}, {location})",
        "crewmate({player})",
        "imposter({player})",
        "-crewmate({player})",
        "-imposter({player})"
    ]
}
```

Listing A.33: data5.json File