Artificial Intelligence: Wumpus World

Carsen Ball
Brett Layman
Jordan Pottruff

Novemeber 3, 2018

1 Software Architecture

Our software divides the task of resolving a Wumpus World game between both prolog and python. The `wumpusWorld.pl` prolog file is used as our knowledge base (KB) and provides predicates for asserting discovered facts. In addition, it provides predicates that allow the player to make inferences given the relations stored in the KB. The program itself is ran as a python file, `wumpusWorld.py`, and communicates with our prolog file using the pyswip library.

The program starts by generating a wumpus world map using `worldBuilder.py`, either randomly or from a specified file. The world builder will assert the locations of the wumpus, pit, and gold into prolog but will not be visible to player as part of the KB. As the player visits a cell, these assertions are queried to determine the percepts to add to the KB.

The game begins by calling startGame on the relevant prolog instance. Starting at position (1,1) facing right, the player's actions from that position are determined in each iteration of a while loop. The decision making process can best be summarized as the following:

1. Does our current position cause our death? If so, return the proper death result.
2. Are we receiving a glitter percept at our position? If so, pick up the gold and initiate a traversal back to the start.
3. Can we deduce any safe cells that we have not yet visited?
   (a) If yes, execute a traversal to the closest unvisited safe cell.
   (b) If no, is the wumpus still alive and can we infer where he is?
      i. If yes, traverse and kill the wumpus to free up more safe cells.
      ii. If no, then no cautious move exists and we must make the smartest move we can based off our KB.

One important characteristic of our program design is that the decision we make at any position can involve moving to a new position that is many cells away. In order to accomplish this, we use the function `shortestPath` which returns the shortest list of visited cells that we can traverse through to get to our destination. From there, `traversePath` is used to simulate the movement along that path, taking into account the possibility of bumping into a wall at the end.

In addition to using `shortestPath`/`traversePath` to line up with the wumpus, killing the wumpus also requires `directionToFaceNext` to ensure that the player faces the right direction when killing the wumpus. The wumpus is only killed as a last resort, when we cannot cautiously explore the world any further due to our KB inferring that potential hazards are blocking us.

Once the player has left the cave or perished inside, the program outputs the results of the player's traversal.

To run the program, execute the following command from the terminal:

`python3 wumpusWorld.py` *`size filename [num]`*

Where *size* is an integer representing the world's width and height, *filename* is the name of the file (including path) to a prewritten world or the string 'random' to generate random worlds, and *num* is optional and used to define the number of random worlds to generate.

2 Logic

The Wumpus World problem is particularly well suited to logical agents. The state of the world is only partially observable. In order to compensate for this, the agent must infer the state of unobserved spaces based on the rules of the world. These rules are deterministic in nature, meaning that the agent can reliably use logical inference. First order logic is especially useful here, because it permits general functions and relations with the "for all" quantifier. This allows us to make statements like: "for all spaces, if a space is 'stenchy', then it's adjacent to the wumpus".

As the agent explores the environment, it adds facts to a knowledge base. These facts are combined with predicates about the world to infer conclusions, thus allowing the agent to behave more rationally than a simple reflex agent. For example, when the agent moves to a "stenchy" space, a simple reflex agent might be programmed to return to the spot it came from to avoid danger. A logical agent, on the other hand, might be able to deduce where the wumpus is, and explore other safe spaces accordingly.

Our program includes a number of logical predicates to define the rules of this world for our agent. One of the simpler predicates involves the act of perception. As the agent receives percepts, it draws conclusions about the nature of the space it's on. For example, if an agent senses a breeze, it can conclude that the spot it's on has a breeze:

$$\forall\, s\ Visit(s) \wedge HasBreeze(s) \Rightarrow FoundBreeze(s)$$

Another important predicate involves identifying if two spots are neighbors (adjacent to each other):

$$\forall\, s1, s2\ Above(s1, s2) \vee Below(s1, s2) \vee Right(s1, s2) \vee Left(s1, s2) \Rightarrow Neighbors(s1, s2)$$

The above expression is used in a predicate for confirming that the wumpus is not at a particular square:

$$\forall\, s1, s2\ Neighbors(s1, s2) \wedge Visited(s2) \wedge InBounds(s2) \wedge \neg Stench(s2) \Rightarrow NoWumpus(s1)$$

Many of our predicates, including the above examples, were designed with the goal of verifying that spaces are safe, so that our agent can continue exploring the environment. As you can see, these predicates can build on eachother to form powerful inferential statements. Our program uses the Prolog programming language to perform logical inference. Prolog simplifies the inference process by making some assumptions. One assumption is the closed world assumption, which says that facts that are not known are assumed to be false. Using this assumption, we designed our agent to perform inference based on what we knew for certain, rather that looking at all possible worlds. Prolog operates by substituting valid atoms into predicates via a backtracking depth first search. This means that Prolog is able to use the established facts in our knowledge base as a starting point for inference.

3 Results

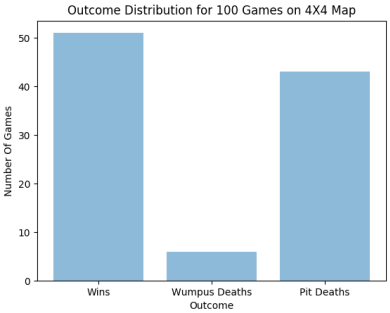
We computed 8 statistic sets for each of the map sizes. For 100 runs on each map, the computations are as follows: the counts of the outcomes(Win, Death by Wumpus, Death by Pit), a histogram for entered spaces across all outcomes, a histogram for scores and entered spaces calculated for each outcome. Due to the wide range of scores, the histogram of the scores across all of the outcomes was non informative, thus it was been omitted. The output.txt file in the zip contains the output for 10 runs for each map.

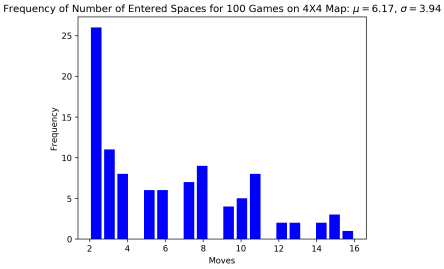
3.1 Analysis of Results

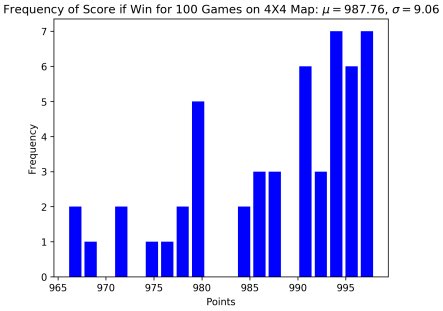
For every map size, the number of moves when there was a death due to a Wumpus was two. Two moves means that the wumpus was in either of the two locations adjacent to the starting point. The only way to win these maps is by luck. This tells us that the program excels at killing the Wumpus when

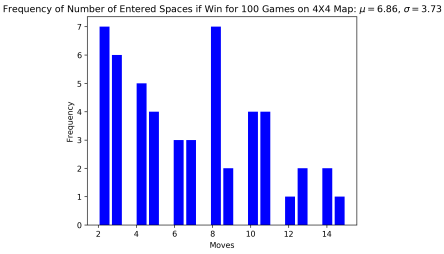
necessary, because there were no other deaths by Wumpus besides these unlucky maps. In regards to spaces entered across all outcomes, it tends towards lower number of moves. This is due to the high likelihood of a pit death, and that pit deaths also tend to result in less moves. Much like the Wumpus, when pits are closer to the start, the agent isn't able to gather very much information before having to take a risk. Typically in these situations, due to an unlucky guess of where a pit might be, the agent dies. The number of spaces entered is more evenly distributed when the game ends with a win. On average, for the agent to win, it has to explore more spaces. In other words, the games in which the agent wins tend to be longer, because premature death due to pits and the Wumpus are avoided. The score distributions are tightly correlated with the spaces entered distributions, which is consistent with how the score is kept: the more moves, the less points. Thus most games where a death occurred had a value relatively close to -1000, because most games resulting in a death had fewer numbers of moves. For games where the agent won, the point distribution is relatively spread out, similar to the spaces entered distribution. The one difference being that there are more games with lower point values, due to the fact that killing the Wumpus removes 10 points from the total. Overall our program behaved as expected. Due to the nature of generating random maps, there are some impossible maps, where even if the agent knew all of the percepts from the start, it still could not be solved. Additionally, there is a high probability that the agent will have to guess its next best move. The probability of guessing correctly is low.

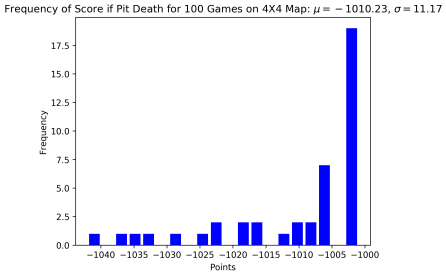
It's interesting to look at how different map sizes resulted in different patterns in the data. We found that with larger maps, the number of deaths by pits increased, while the number of deaths by Wumpus decreased. We think that the increase in pit deaths is due to the fact that, on average, the agent must navagate more pits to reach the gold on larger maps. In contrast, the number of Wumpuses remains constant, even with larger maps, so the agent isn't more likely to be killed by a wumpus on the way to the gold. But why is the agent actually less likely to die from the wumpus on these maps? It's possible that a larger environment allows the agent to gather more information before being forced to confront the wumpus, or perhaps it's simply easier to avoid the wumpus altogether on larger maps. In any case, it's interesting to see how changing the environment can influence the effectiveness of the agent, and the logical rules being employed by the agent.

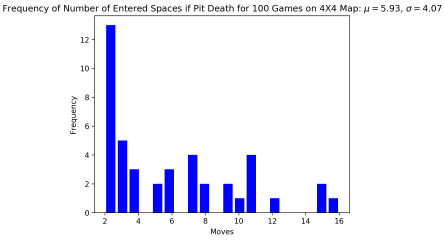

Outcome Distribution for 100 Games on 4X4 Map
Number Of Games
0
10
20
30
40
50
Wins
Wumpus Deaths
Pit Deaths
Outcome

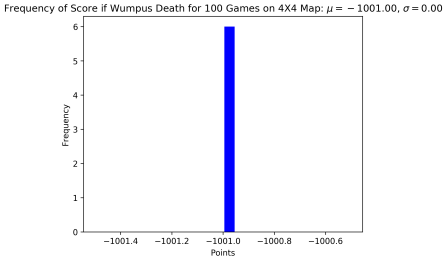

Frequency of Number of Entered Spaces for 100 Games on 4X4 Map: $\mu = 6.17$, $\sigma = 3.94$
Frequency
0
5
10
15
20
25
2
4
6
8
10
12
14
16
Moves

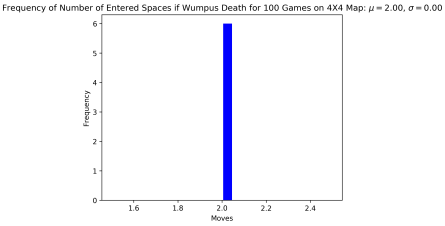

Frequency of Score if Win for 100 Games on 4X4 Map: $\mu = 987.76$, $\sigma = 9.06$
Frequency
0
1
2
3
4
5
6
7
965
970
975
980
985
990
995
Points

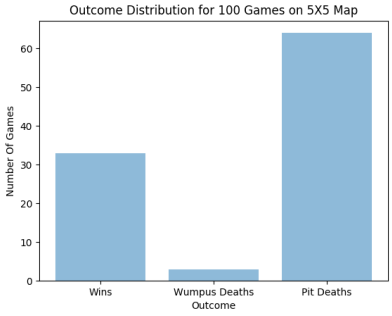

Frequency of Number of Entered Spaces if Win for 100 Games on 4X4 Map: $\mu = 6.86$, $\sigma = 3.73$
Frequency
0
1
2
3
4
5
6
7
2
4
6
8
10
12
14
Moves

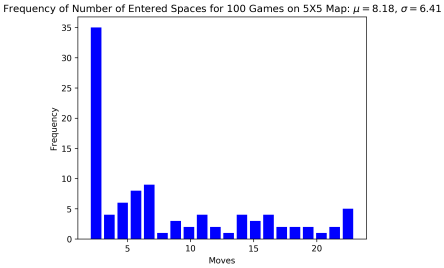

Frequency of Score if Pit Death for 100 Games on 4X4 Map: $\mu = -1010.23$, $\sigma = 11.17$
Frequency
Points

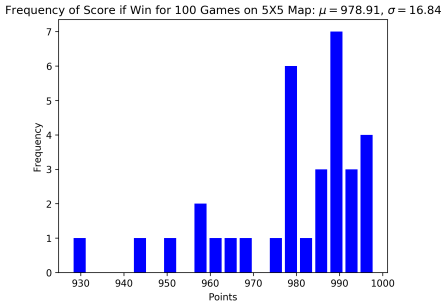

Frequency of Number of Entered Spaces if Pit Death for 100 Games on 4X4 Map: $\mu = 5.93$, $\sigma = 4.07$
Frequency
Moves

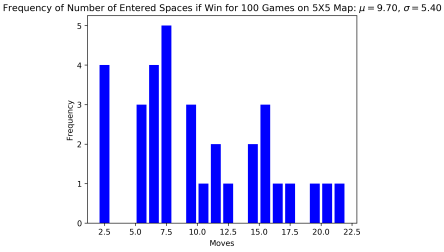

Frequency of Score if Wumpus Death for 100 Games on 4X4 Map: $\mu = -1001.00$, $\sigma = 0.00$
Frequency
Points

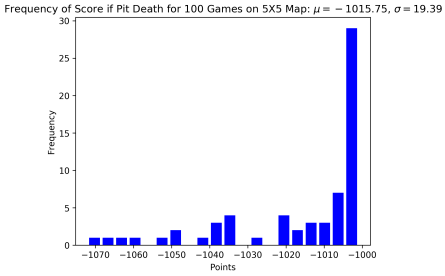

Frequency of Number of Entered Spaces if Wumpus Death for 100 Games on 4X4 Map: $\mu = 2.00$, $\sigma = 0.00$
Frequency
Moves

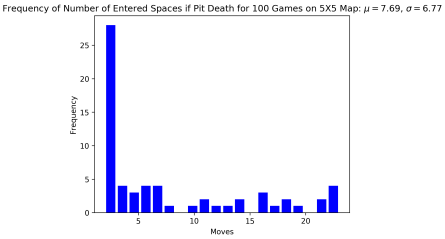

Outcome Distribution for 100 Games on 5X5 Map
Number Of Games
Wins
Wumpus Deaths
Pit Deaths
Outcome

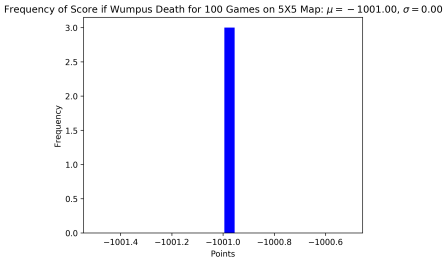

Frequency of Number of Entered Spaces for 100 Games on 5X5 Map: $\mu = 8.18$, $\sigma = 6.41$
Frequency
Moves

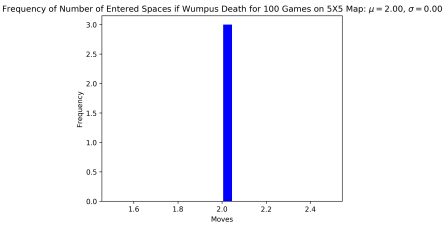

Frequency of Score if Win for 100 Games on 5X5 Map: $\mu = 978.91$, $\sigma = 16.84$
Frequency
Points

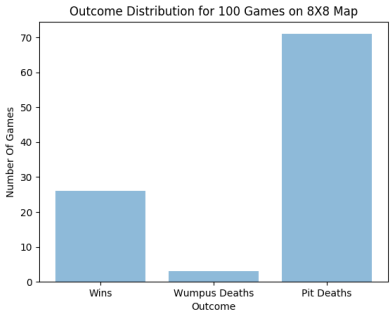

Frequency of Number of Entered Spaces if Win for 100 Games on 5X5 Map: $\mu = 9.70$, $\sigma = 5.40$
Frequency
Moves

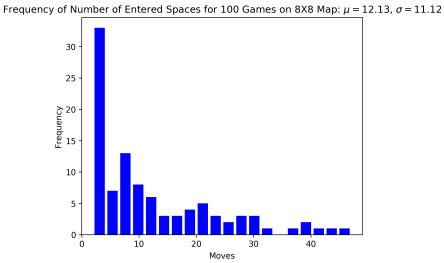

Frequency of Score if Pit Death for 100 Games on 5X5 Map: $\mu = -1015.75$, $\sigma = 19.39$
Frequency
Points

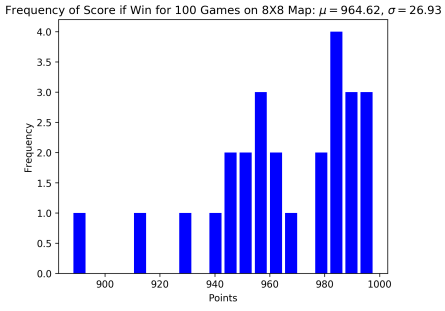

Frequency of Number of Entered Spaces if Pit Death for 100 Games on 5X5 Map: $\mu = 7.69$, $\sigma = 6.77$
Frequency
Moves

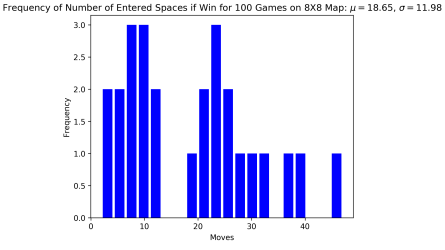

Frequency of Score if Wumpus Death for 100 Games on 5X5 Map: $\mu = -1001.00$, $\sigma = 0.00$
Frequency
Points


Frequency of Number of Entered Spaces if Wumpus Death for 100 Games on 5X5 Map: $\mu = 2.00$, $\sigma = 0.00$
Frequency
Moves

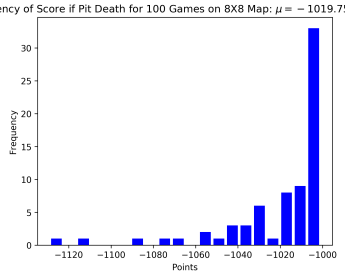

Outcome Distribution for 100 Games on 8X8 Map
Number Of Games
Wins
Wumpus Deaths
Pit Deaths
Outcome


Frequency of Number of Entered Spaces for 100 Games on 8X8 Map: $\mu = 12.13$, $\sigma = 11.12$
Frequency
Moves

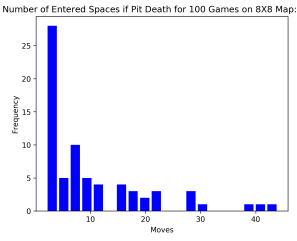

Frequency of Score if Win for 100 Games on 8X8 Map: $\mu = 964.62$, $\sigma = 26.93$
Frequency
Points


Frequency of Number of Entered Spaces if Win for 100 Games on 8X8 Map: $\mu = 18.65$, $\sigma = 11.98$
Frequency
Moves

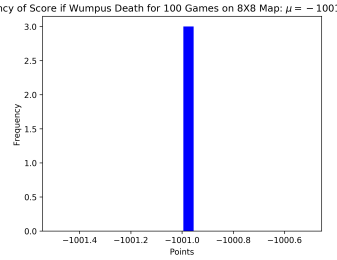
Frequency of Score if Pit Death for 100 Games on 8X8 Map: $\mu = -1019.75$, $\sigma = 26.03$



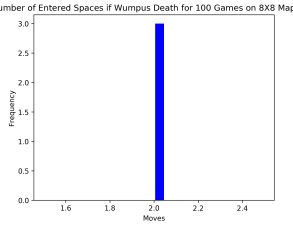
Frequency of Number of Entered Spaces if Pit Death for 100 Games on 8X8 Map: $\mu = 10.17$, $\sigma = 9.89$

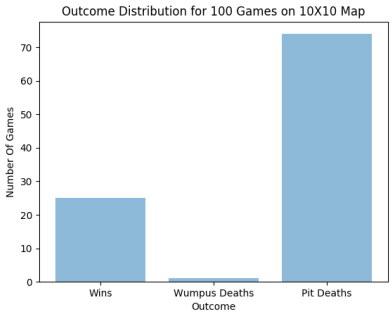


Frequency of Score if Wumpus Death for 100 Games on 8X8 Map: $\mu = -1001.00$, $\sigma = 0.00$

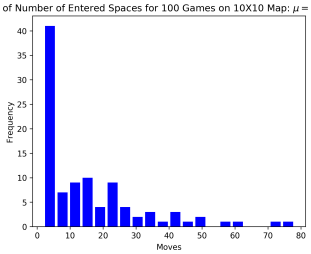


Frequency of Number of Entered Spaces if Wumpus Death for 100 Games on 8X8 Map: $\mu = 2.00$, $\sigma = 0.00$

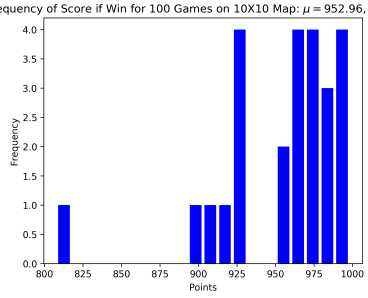




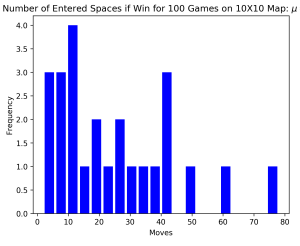
Frequency of Number of Entered Spaces for 100 Games on 10X10 Map: $\mu = 15.60$, $\sigma = 16.38$

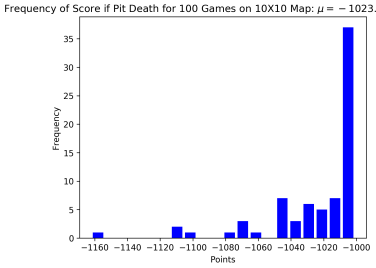


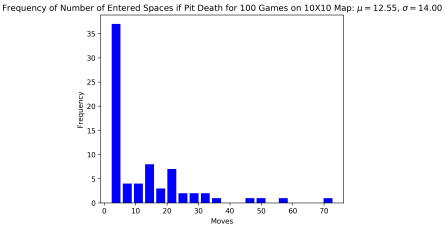
Frequency of Score if Win for 100 Games on 10X10 Map: $\mu = 952.96$, $\sigma = 41.38$

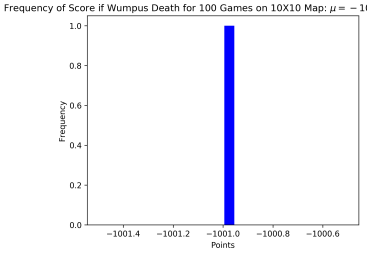


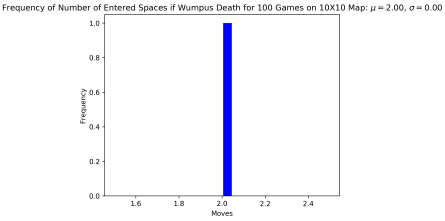
Frequency of Number of Entered Spaces if Win for 100 Games on 10X10 Map: $\mu = 25.16$, $\sigma = 19.15$




Frequency of Score if Pit Death for 100 Games on 10X10 Map: $\mu = -1023.41$, $\sigma = 31.32$
Frequency
Points
0
5
10
15
20
25
30
35
−1160
−1140
−1120
−1100
−1080
−1060
−1040
−1020
−1000


Frequency of Number of Entered Spaces if Pit Death for 100 Games on 10X10 Map: $\mu = 12.55$, $\sigma = 14.00$
Frequency
Moves
0
5
10
15
20
25
30
35
0
10
20
30
40
50
60
70


Frequency of Score if Wumpus Death for 100 Games on 10X10 Map: $\mu = -1001.00$, $\sigma = 0.00$
Frequency
Points
0.0
0.2
0.4
0.6
0.8
1.0
−1001.4
−1001.2
−1001.0
−1000.8
−1000.6


Frequency of Number of Entered Spaces if Wumpus Death for 100 Games on 10X10 Map: $\mu = 2.00$, $\sigma = 0.00$
Frequency
Moves
0.0
0.2
0.4
0.6
0.8
1.0
1.6
1.8
2.0
2.2
2.4

4 Contrbution

4.1 Brett

Tested the Python-Prolog interface. Developed preliminary Prolog predicates. Helped with brainstorming test cases and algorithms for guiding agent behavior. Wrote logic section of paper.

4.2 Carsen

Produced, formated and analyzed the results section. Contributed to the ideating stage of writing the program. Added some functionality to the prolog database and the wumpus world driver. Assisted in debugging.

4.2.1 Jordan

Wrote the software architecture section. Developed predicates for our prolog knowledgebase, helped write our problem generator, and wrote functions for moving around the map in python.