**CSCI 446 - Artificial Intelligence: Project 2**

*FOL, Reasoning Agent, Reactive Agent, Wumpus World*

**Report**

< Version 1.0.0 >

Group 2

Emmanuel Armstrong

Matthew Rohrlach

Nicolas Moore

2016–10–24

**Surviving Wumpus World using First-Order Logic**

October 24, 2016

**Nicolas Moore** (nicolas.moore88@gmail.com)

**Matthew Rohrlach** (mattrohrlach@gmail.com)

**Emmanuel Armstrong** (emmanuelrarmstrong@gmail.com)

**1.** **Abstract:**

The Wumpus World problem is an artificial intelligence problem that consists of creating a cell-based world to use with a navigating agent. The goal is for the agent to traverse the world and find the gold while avoiding deadly obstacles; such as pits and the eponymous Wumpuses. In this project’s implementation of the Wumpus World problem, first-order logical operations are used to create a reasoning inference method for agents. The reasoning algorithm uses this logic to make decisions with regards to what the agent “knows” about the world, and as well as what the agent can sense within each cell. To act as a control, a reactionary agent is employed that only makes decisions with regards to what it believes exists in neighboring cells. The reasoning agent was able to solve almost every solvable world given to it, even at the largest world size; a grid of 25 by 25 cells. The reactionary agent could solve the smaller worlds fairly-well, but as the world size and probability of hazards increased, the solvability of the reactionary agent and the resulting number of “points” decreased drastically. The project team concludes the reasoning agent is more efficient than the reactionary agent.

**2.** **Problem Statement/Hypothesis:**

The purpose of this project was to create a knowledge-based, logically-reasoning agent with the ability to make decisions towards the successful traversal of varying “Wumpus worlds”. The agent must use a first-order logic reasoning system to decide the next appropriate action. First-order logic is symbolized reasoning in which each sentence, or statement, is broken down into a subject and a predicate. The predicate modifies or defines the properties of the subject. In first-order logic, a predicate can only refer to a single subject. A sentence in first-order logic is written in the form Px or P(x), where P is the predicate and x is the subject, represented as a variable. Complete sentences are logically combined and manipulated according to the same rules as those used in [Boolean](http://searchcio-midmarket.techtarget.com/definition/Boolean) algebra. The goal of the agent is to reach the cell containing the “gold” before succumbing to the hazards placed randomly throughout the world. The first task of our software was to generate the Wumpus worlds. Each Wumpus world consisted of a grid in sizes ranging from 5 by 5 to 25 by 25 cells.

The next task consisted of placing agents within worlds. Both agents traversed these worlds, resurrecting at each death; either determining they cannot reach the gold or reaching the gold. Of the two agents, the reasoning agent traverses the world by first maintaining a knowledge base of first-order logical facts in clause form; then by using that knowledge to make inferences about the unknown aspects of the Wumpus world; as well as actions that would be most-appropriate to take next. In comparison, the reactive agent traverses the Wumpus world by performing randomized moves to adjacent cells it believes to be safe, or by moving to an unsafe cell if there are no safe cells adjacent. To track this, the reactive agent is capable of remembering previously-visited cells as well as cells unmarked by the agent’s senses. The agent is not capable of remembering the path to these cells, but recognizes them if they re-appear.

Further details regarding the implementation of the Wumpus world generator, reasoning agents, and reactive agents are contained in Section 3 of this document. Our experimental designs, the structure of the tests we have performed, the parameters of such tests, and the comparisons that we have drawn between the reasoning and reactive agents may be found in Section 4 of this document.

It was hypothesized that the reasoning inference would be able to solve most solvable worlds, while carrying a higher overall percentage of Wumpuses killed. While still expected to solve a fair percentage of worlds, it was hypothesized that the reactive agent would do so less efficiently in comparison to the reasoning agent.

**3.** **Algorithms Implemented:**

In this project, software elements were written such that every class represents an object in a practical example of a Wumpus world. For example, the world is written as a self-contained class that accepts and carries out actions affected upon it by the agent class. The agent class shares no methods with the world class, however both classes maintain certain elements in redundancy; such as the current position of the agent. The agent class also contains instances of logically-reasoning or reactive “inference” classes that act as decision-makers. When the appropriate inference class has reached a decision, it passes this decision in the form of a string object to the agent; who then passes this decision back to the Wumpus world class to be parsed into an action. Stored within the reasoning inference class exists both a knowledge bank and a set of first-order logical rules. By updating this knowledge bank at every movement, a list of visited, safe-yet-unvisited (e.g. no suspicions), and suspicious cells can be maintained by the agent’s sense of reasoning. To follow the project rules, these fragments of knowledge are then applied in first-order logic rules in clause form to determine an appropriate action. This object-based design, and the familiarities of the project authors, left the Java language as the appropriate choice of programming language.

**3.1** **Wumpus World Generator**

As mentioned in Section 1, the Wumpus world generator builds Wumpus worlds of sizes ranging from 5 by 5 to 25 by 25 cells. Each increment in grid size will occur by 5 cells (i.e. 5 by 5 becomes 10 by 10, 15 by 15 becomes 20 by 20, etc.). A cell consists of an Integer key, five boolean values that respectively determine if the cell is empty, contains a pit, contains an obstacle, contains a Wumpus, or contains gold. A cell also includes a list of cell keys that are connected to itself. As input, the Wumpus world generator’s constructor requires an agent, either reasoning or reactive, an integer denoting the side dimension of the Wumpus world, measured in cells, an integer to denote the probability of generating a pit in a cell, an integer to denote the probability of generating an obstacle in a cell, and an integer to denote the probability of generating a Wumpus inside a cell.

The Wumpus world constructor initializes a cell array with size equal to the side dimension squared. A helper function called by the constructor then performs the work of actually building the Wumpus world. Given the integer probability values, the helper function then generates cells incrementally until the constraints of the world are met. During the generation of a cell, a key value assigned to the cell and, based on the aforementioned probability values, the helper function determines whether or not to place a pit, an obstacle, or a Wumpus in the given cell. The boolean values for the cell be set accordingly, and the cell is added to the cell array stored by the world. The southwest cell will always be made empty, so that it may be used as a safe starting point for the agent. Once all the cells have been created, each cell has its own list of connected cell keys populated, and a list of empty cells, not containing the first cell, is also created. One cell from the list of empty cells is then chosen at random and the gold is placed in that cell. After the gold is placed, the agent is placed in the southwestern corner cell.

The Wumpus World is the class responsible for keeping track of the agent's current location. The Wumpus World is also responsible for accepting and executing the actions of the agent, including arrows fired.

**3.2** **Reasoning Inference**

When the current position is passed to the reasoning inference class contained within the agent class, the object automatically updates the knowledge base of that agent with the implication of the current position being safe. The knowledge base consists of “knowledge fragment” objects, each representing a cell that the agent is aware of. Each of these fragments maintains its own list of connected fragments, drawn from the physical cell’s list of connected cells. Each fragment also contains lists that represent the suspicions said fragment has about its neighbors’ Wumpus and pit states, respectively. The smell or wind of the current position affects both this current fragment’s suspiciousness in neighboring lists, as well as the lists maintained by this fragment itself.

At every movement, the knowledge bank of the inference class is updated in a linear process. After the current position has been matched to the fragment that stores information known about that position, the fragment then removes itself from the suspicions of every neighboring cell. If a previous cell had a smell or breeze, such a report would imply that a neighboring cell had a hazard, and all non-safe neighbors would be suspected of harboring that hazard. With suspicion removed from the current position, if there is only a single remaining suspicious neighbor in a list, a conclusion can be drawn about a hazard within that neighbor. A conclusion drawn about a pit would be permanent, but, in the case of a Wumpus, a conclusion would be tentative. A Wumpus may be killed by an arrow fired from a distance, so this conclusion would be subject to change in the face of an updated smell report at a neighbor. A conclusion about an obstacle could only be drawn by an attempt to move into a cell, so this conclusion would not be drawn through suspicions. When the knowledge bank is properly maintained, it provides a “frontier” of safe, non-visited cells and safe, visited cells. The distinction between the two is such that a visited cell is a guaranteed traversable path, but an obstacle might still exist in a non-visited cell. All cells blocked with obstacles are permanently marked as non-traversable (e.g. not safe).

The decision-making process is now structured through the aforementioned first-order logic rules. These rules are split into two major sections. The actual rules for decision making, and the rules for forming the knowledge. These rules are listed and described below:

DECISION-MAKING RULES:

|  |
| --- |
| (actionToTake):  gold(currentCell) ⇒ action(grab)  -OR-  currentlyPathingToSpecificCell(inferenceClass) ⇒ action(nextPathAction)  -OR-  (destinationsAwaitingPathing ∧ destinationPossible(nextDestination) ⇒ (createPath(nextDestination) ∧ action(nextPathAction))  -OR-  (safeUnvisitedCellExists(knowledge) ∧ destinationPossible(nextSafeUnvisitedCell(knowledge))) ⇒ (createPath(nextSafeUnvisitedCell(knowledge)) ∧ action(nextPathAction))  -OR-  ((unvisitedFrontierCellExists(knowledge) ∧ destinationPossible(nextFrontierUnvisitedCell(knowledge))) ⇒ (createPath(nextFrontierUnvisitedCell(knowledge)) ∧ action(nextPathAction)) ⇒ nextCellHasWumpusSuspicion(knowledge) ⇒ fireIntoNextCell(nextFrontierUnvisitedCell))  -OR-  action(halt) |

The process of decision making is outlined by the rules above. The reasoning inference class will run one of these rules at a time, and only choose the next rule after the previous is no longer true. The class starts by choosing the first true action to be taken. These potential actions include:

- Grabbing the gold, which is set at the highest priority, ensures that the gold is always grabbed if possible.

- Take in a list of actions that will take the agent to a set destination cell from the knowledge base.

- Build a path to the next destination that is told to the world by the explorer.

- Build path to next safe unvisited cell that is known by the knowledge base.

- Check the distances between unvisited frontier cells, and chose the path to the best cell. If a cell has a sense report that is risky then that cell will be fired into beforehand.

- If there are no more actions to be taken then the world is unsolvable, either through certain-death (no remaining arrows) or blocked gold. The only action to do is to halt and output that the world is unsolvable.

KNOWLEDGE-FORMING RULES:

|  |
| --- |
| ∀ currentCell, ¬hazard(currentCell) ⇒ isVisited(currentCell) ∧ isSafe(currentCell)  ∀ currentCell, isSameCellUnexpectedly(currentCell, previousCell) ⇒ isObstacle(formerDestinationCell)  ∀ cell, isSafe(cell) ⇔ ¬hasWumpus(currentCell) ∧ ¬hasPit(currentCell)  ∀ cell, isVisited(cell) ⇒ isSafe(currentCell) ∧ ¬isObstacle(currentCell)  ∀ neighboringCell, hasWumpusSuspicion(neighboringCell) ⇔ hasSmell(currentCell) ∧ ¬isSafe(neighboringCell)  ∀ neighboringCell, hasPitSuspicion(neighboringCell) ⇔ hasWind(currentCell) ∧ ¬isSafe(neighboringCell)  ∀ currentCell, ¬hasSmell(currentCell) ∧ ¬hasWind(currentCell) ⇒ ∀ neighboringCell, isSafe(neighboringCell)  ∀ cell, onlyOneWumpusSuspect(cell) ∧ ¬isSafe(suspectCell) ⇒ hasWumpus(suspectCell)  ∀ cell, onlyOnePitSuspect(cell) ∧ ¬isSafe(suspectCell) ⇒ hasPit(suspectCell)  ∀ cell, hasWumpus(cell) ⇔ ∀ neighborCell, hasSmell(neighborCell)  ∀ cell, hasPit(cell) ⇔ ∀ neighborCell, hasWind(neighborCell) |

All knowledge is represented by knowledge fragments that themselves represent cells. Each visited cells fragment maintains a list of suspicions about its neighbors. This list is determined through the knowledge-forming rules stated above. Below is a description of each of the rules:

- If no hazard is detected it confirms that the cell as visited and safe.

- If the agent is being returned to the same cell unexpectedly it indicates that the cell that was attempted to be moved into contains an obstacle.

- All cells are considered safe for an agent if they have no Wumpus or pit.

- If a cell is marked as visited it implies that it is safe and also has no obstacle

- If a cells neighbor has not been marked safe, and has a smell then it is suspected of containing a Wumpus.

- If a cells neighbor has not been marked safe, and has a breeze then it is suspected of containing a pit.

- If a cell contains no smell and no wind this implies that all neighboring cells are also safe

- If any cell has only one neighbor that is suspected of containing a Wumpus, this implies that that suspected cell does contain a Wumpus.

- If any cell has only one neighbor that is suspected of containing a pit, this implies that that suspected cell does contain a pit.

- Any cell may contain a Wumpus if and only if all of its neighbors have a smell.

- Any cell may contain a pit if and only if all of its neighbors have wind.

The world uses these rules to send the agent into safe, non-visited cells until the gold is found or there are no more safe non-visited cells. The visited cells of the world function as a road between such safe, non-visited cells that may exist at opposite ends of the world. When there are no more safe, non-visited cells to explore, the agent uses the first-order logic rules to choose a Wumpus-containing cell with an unvisited, unmarked neighbor. The agent will then fire-into and traverse this cell, ideally opening a new avenue of movement. If no such cells exist, the agent uses the aforementioned logic to find any unvisited cell that may be visited without guaranteed failure. Finally if no such cells still exist, the agent reports their failure to the world. When the gold has been reached, regardless of the path taken, the world recognizes this fact and automatically reports success to the agent.

**3.3** **Reactionary Inference**

To act as a control in experiments concerning the reasoning agent, a reactive agent has been introduced as an alternative choice in decision-making. The reactive agent contains a reactionary inference class, designed not to store knowledge, but to act upon the best conclusion that can be drawn about the agent’s immediate surroundings. For example, if the agent reaches a cell with both a smell and wind, and the current position has three neighbors, the best course of action would be to return to the previous cell. Such an action would be governed purely by the agent’s reaction to an immediate condition; not by conclusions drawn in the context of broader knowledge.

To accomplish this task, the agent consults a set of logical rules that, in contrast to the reasoning agent, will be run on current conditions only. The agent does keep a knowledge base that consists of the cells they have already visited. However it does not use this knowledge to infer safety in the cells that are not in its immediate surroundings.

**4.** **Experimental approach:**

In order to compare the reasoning and reactive agents, a set of one hundred Wumpus worlds will be generated for each size: 5 by 5, 10 by 10, 15 by 15, 20 by 20, and 25 by 25 cells. Each agent will traverse the same set of Wumpus worlds to ensure that no agent has an easier set of Wumpus worlds relative to the other agents. During an agent’s traversal of the Wumpus world, many metrics, described below, will be collected in order to form a means for comparing the agents.

**4.1** **Planned Parameter Tuning**

For this version of the Wumpus World, there will be specific parameters that must to be tuned to achieve accurate results. The three primary parameters that will be used in this example will be: the max number of actions that the agent is allowed in each world, the number of worlds to send the agent through in each world size, and the probability that a hazard will be placed in a given cell. These variables will be very important in producing accurate results from the agent that can be compared to the results of other agents ran through the world.

The “max number of actions” given to each agent will provide an adequate sample of decisions made in order to illustrate how each agent deals with more complex worlds. This parameter will also provide a good level for comparing the reactionary inference to the reasoning inference. The reasoning inference should solve harder worlds much more consistently, and this parameter will allow better illustrations of its efficiency.

The “number of worlds” parameter is fairly straightforward. The more worlds of a given size that an agent is sent through, the more data that will be collected to compare and interpret. The experimentation done in this project implementation will run one hundred different worlds of each world size. The data gathered from each set of worlds will be averaged and normalized into a single metric to be compared between agents.

Finally, the probability that a hazard will be placed within a cell will be used to fine-tune the difficulty of the worlds to be explored. This will help to illustrate the efficacy of each agent in regards to world-solving. If the program reaches a the point where no agents are completing worlds, it will be clear that the difficulty is set too high. If this situation occurs, this parameter can be tuned once more to find the optimal difficulty for accurate results and data gathering. Determining the parameter values that produce the strongest results will be a major focus of this experimentation.

**4.2** **Variables to Track for Comparison**

The main metric that will be tracked for comparison will be the average score obtained by each agent when a set of worlds has been completed, under a given hazard probability setting. This average will be saved into the points variable. A higher average score in points will obviously indicate a more successful agent, while a lower score will show a less successful solution. Because of this, the agents are able to be compared, and a conclusion about efficiency made with reasonable confidence.

The size of the world that the agents are running on will also be tracked. This variable is very important for accurate data collection. Each agent must be positive that it is being run on a world of the same size, or else the results would be very different for each.

The final variable to be tracked is the probability of hazard spawns, in other words, pitProbability and wumpusProbability respectively. These variables will be tracked for comparison once the agents have completed their exploration of every world. The agents must have worlds that are generated with similar levels of difficulty for proper testing. If one agent has a world with twice as many Wumpuses or pits as a different one then it will provide very skewed results for comparison.

**4.3 How Comparisons are Created**

As the agent moves through the world its score will be constantly monitored. There will exist a set starting number of points and each action made will have an effect on the score. At the end of each iteration, the final value stored in the points variable will be saved and the next world will be initialized. At the point where all of the worlds of a particular size have been completed, the stored points will be averaged together for each survivor. The agent will know when points are edited due to its senses in the world, as well as the default actions that are always documented. For example, when a scream is detected, the agent will be told to add 10 points to its points variable. This is performed for every action and reaction that an agent makes.

For this project the probability for hazard spawns was set to have a 33% chance of a hazard in each cell. This is to guarantee the worlds are all of a similar difficulty, thus providing accurate result collection. This probability will stay the same for every world until the program ends, and then can be adjusted accordingly for future tests.

The program will also keep track of the size of the world that is being tested for this set of iterations. The world size will be saved as a variable; and this variable will remain the same for the set number of iterations before increasing to match the following world size. This process will repeat until the max size of 25x25 is reached. When this has occurred, the agents will be run for the set number of actions, and the final averages calculated.

The strongest agents, those with the highest average scores for all worlds, will be deemed the best. The size of the worlds will also affect the difficulty set on the agents. If the worlds are bigger there are more possible cells for hazards to appear in, however, there are more possible routes through the world that could be utilized by an agent. Accordingly, the project will take this into account when setting the hazard probabilities in different-size worlds.

**5.** **Results:**

Table 1- Gold Found 5% Probability Table 2- Gold Found 10% Probability

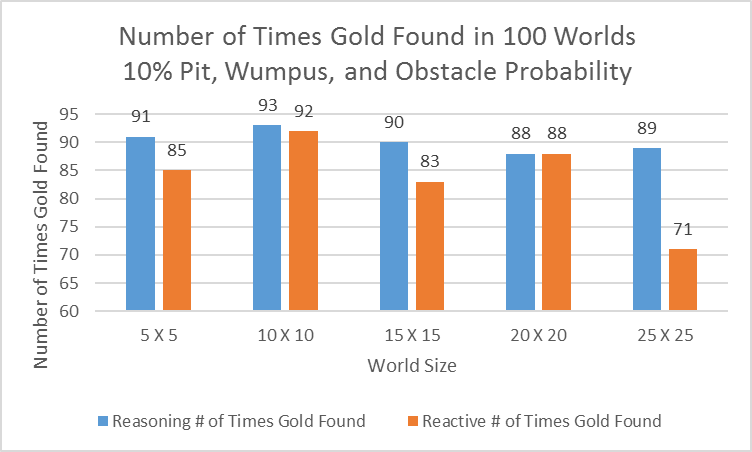
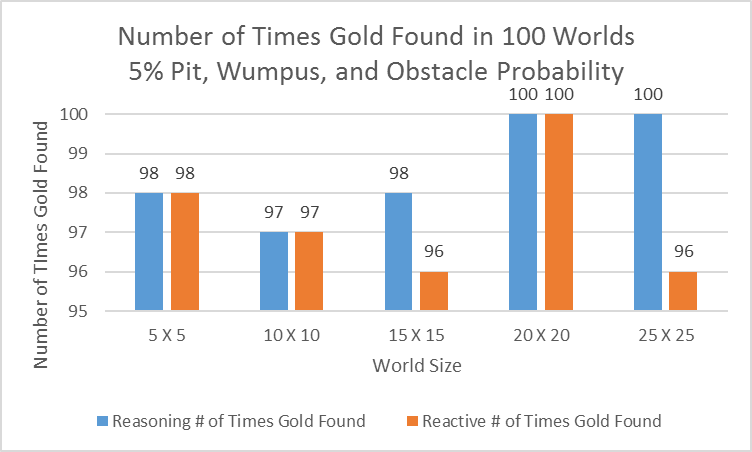


Table 3- Gold Found 15% Probability Table 4- Average Points 5% Probability

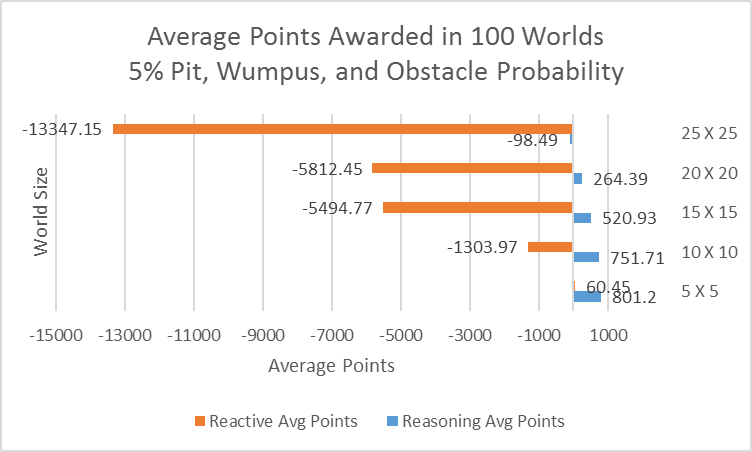
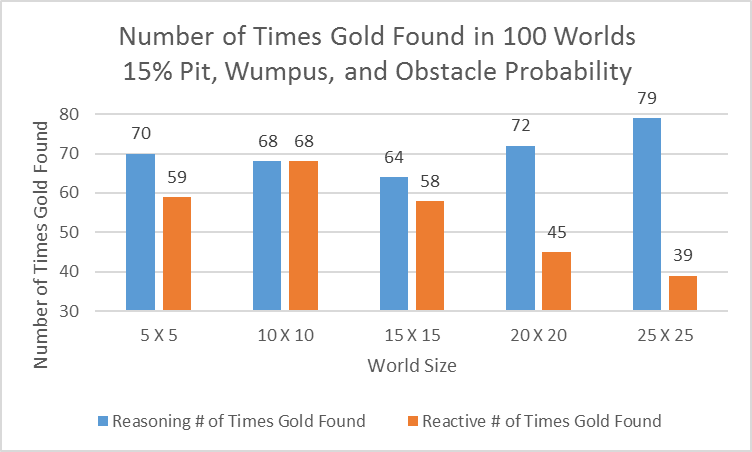


Table 5- Average Points 10% Probability Table 6- Average Points 15% Probability

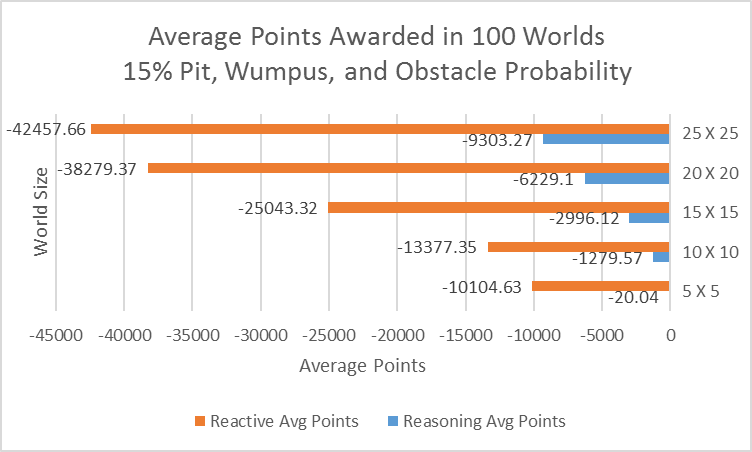
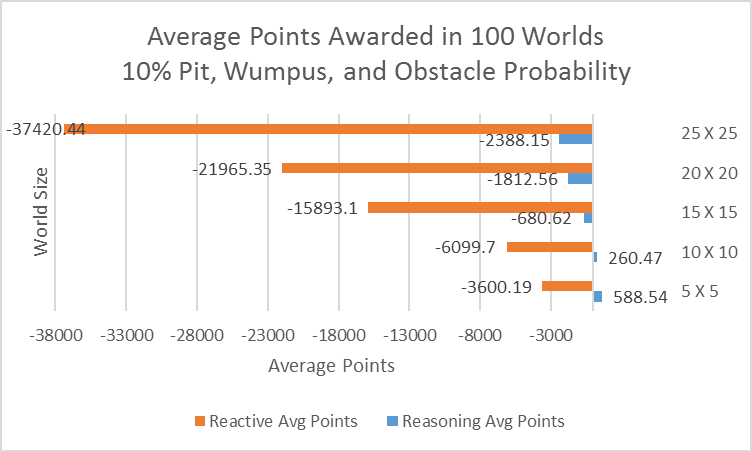


Table 7- Percent Cells Explored 5% Probability Table 8- Percent Cells Explored 10% Probability

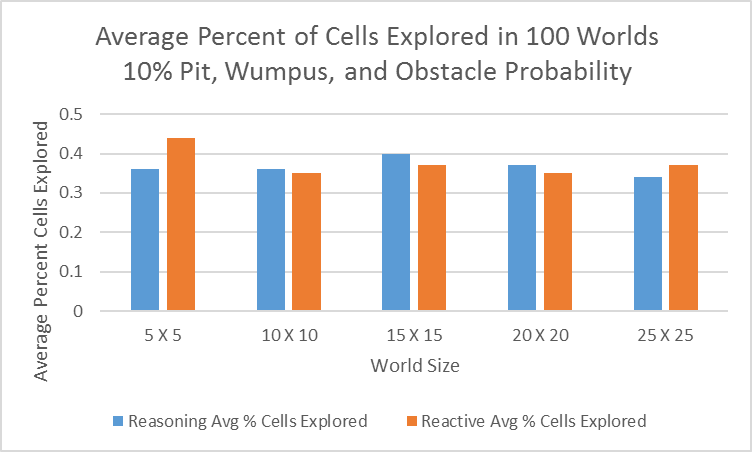
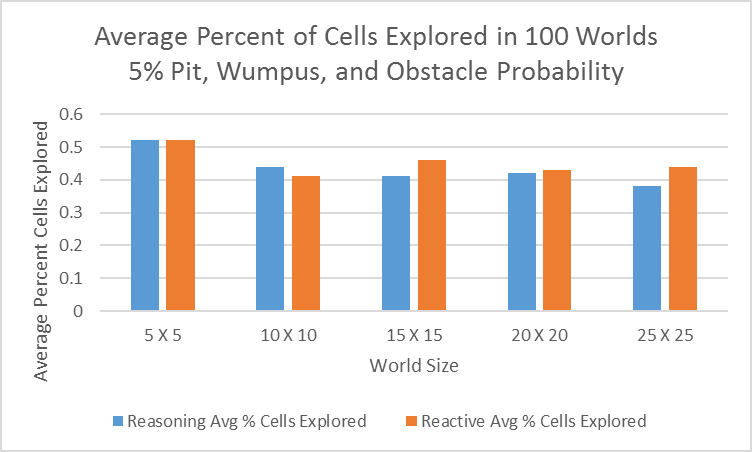


Table 9- Percent Cells Explored 15% Probability Table 10- Deaths 50% Probability

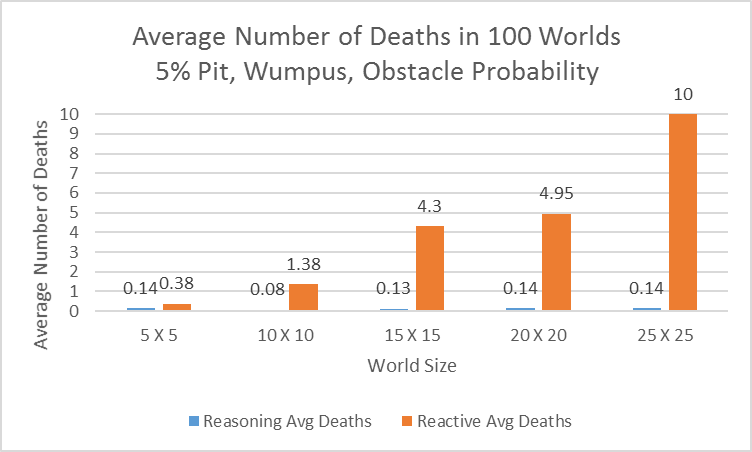
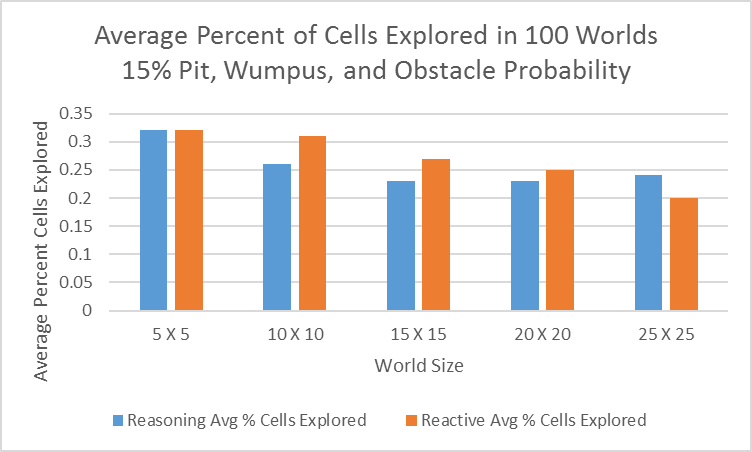


Table 11- Deaths 10% Probability Table 12- Deaths 15% Probability

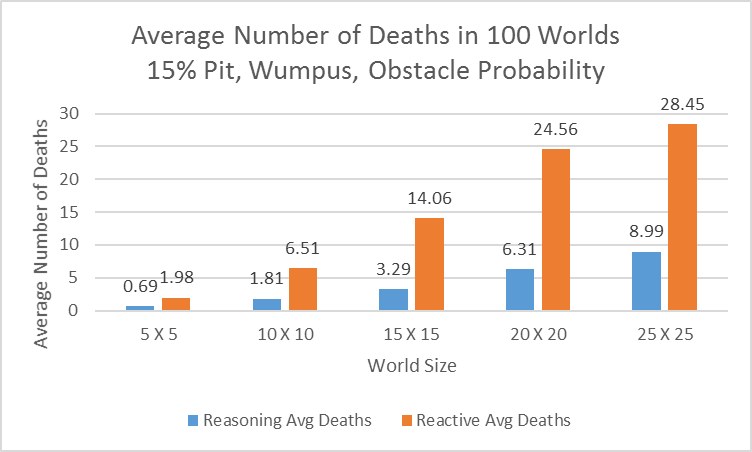
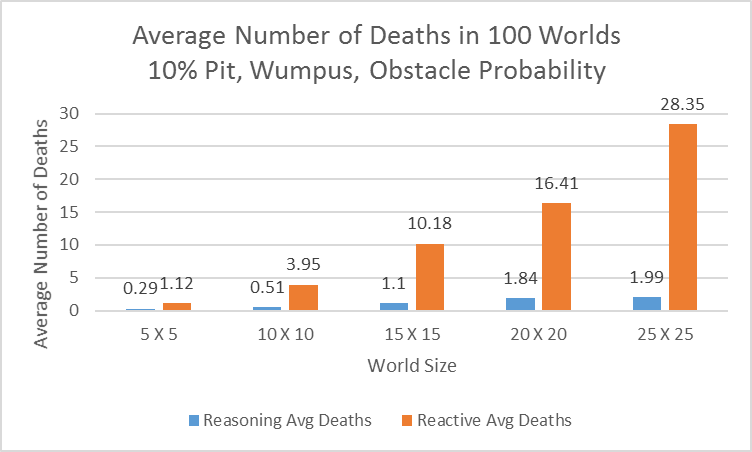


Table 13- Wumpus Kills 5% Probability Table 14- Wumpus Kills 10% Probability

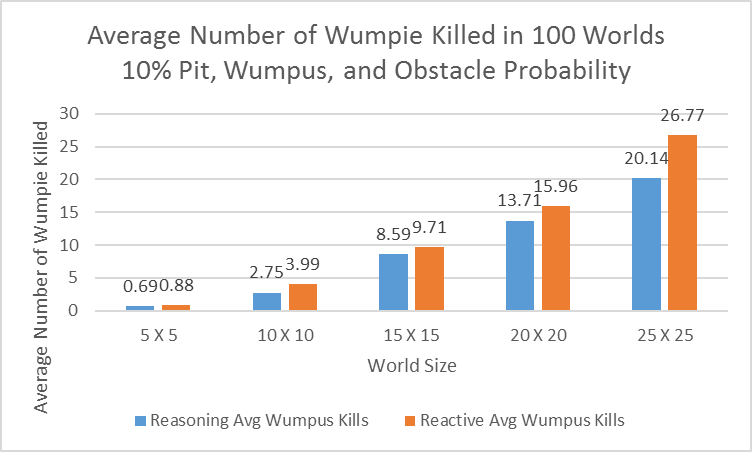
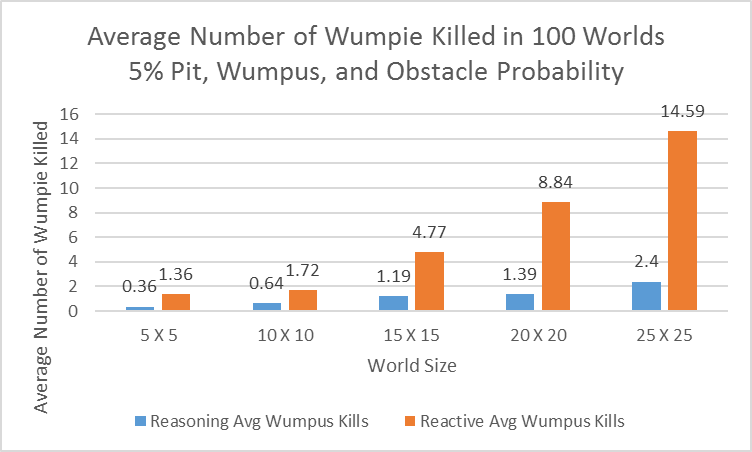
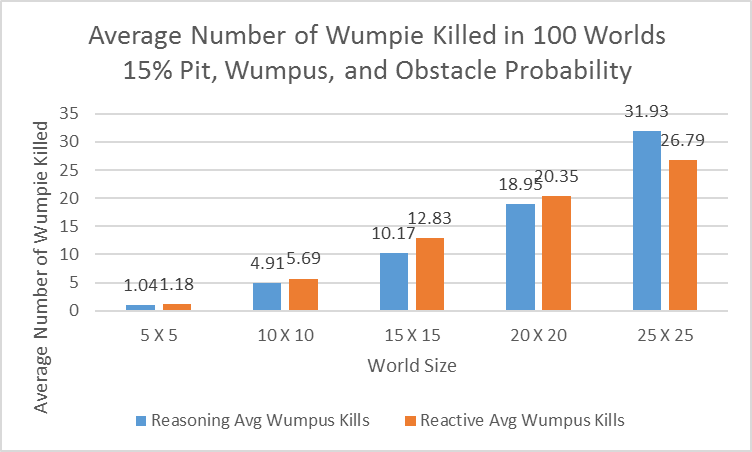


Table 15- Wumpus Kills 15% Probability



**6.** **Discussion of Algorithmic Behavior:**

As expected the reasoning agent was able to solve worlds much more efficiently than the reactionary. It was also able to solve more of the solvable worlds, due to the fact that the reactionary agent ran out of moves before it could solve some of the worlds. Once the program got to the larger worlds this was even more apparent.

The reasoning agent was able to discern where Wumpuses spawned and seemed to either dodge or kill them most of the time. They were able to kill a great number of Wumpuses on each map, as shown in Table 15. This agent also solved the worlds in a far more efficient manner with many less deaths than the reactionary agent did. This data comes from tests run on a set of 100 worlds of each size where not all worlds were solvable.

The reactionary agent had a much harder time solving worlds. Due to the fact that it made decisions based only off of what is known at that moment it took a lot more moves to get to the gold, and many times this would cause it to run out of moves before finding the gold. While it took many more tries and deaths to solve each world, it still did a fairly good job of killing the Wumpuses. This also comes from tests run on a set of 100 worlds of each size where not all worlds were solvable.

As far as points go the reasoning agent again outperformed the reactionary agent by a very large margin. While on the larger worlds both agents came up with negative average scores, the reasoning agents score was still far greater than that of the reasoning agent. The reasoning agent also was able to find and pick up the gold much more than the reactionary, with the exception of the 10x10 set of worlds, where they each were able to grab the same amount of gold.

**7.** **Summary:**

In this paper we have demonstrated using First-Order logical statements to efficiently solve the Wumpus World problem. This experiment turned out to be a very effective demonstration of logic based reasoning and problem solving. It relies on the understanding of this logic to be implemented correctly, both from the programmer, and the agent that is created. Because of this the project clearly shows when and how logic is used to solve problems at hand. Logic is a powerful tool and comparing one agent that uses logic against one that does not makes it very clear which agent performs better under these circumstances.

Tunable parameters and probabilities produced many accurate results that can be tested and compared. For example, limiting the number of tries an agent gets could produce a large gap in solvability between the two agents. The reactive should solve every graph the reasoning does, however, it just will do so much less efficiently. Taking all of this into consideration this project provided a very clear view into logic and how it affects the solving of problems.

**References:**

[1] Russell, Stuart J., Peter Norvig, and Ernest Davis. *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ: Prentice Hall, 2010. Print.

[2] Nelson, Nils J. *Logic and Artificial Intelligence*. Tech. Stanford: Computer Science Department, Stanford U, 1989. Print.

[3] Ayroinde, I. T., & Akinkunmi, B. O. (2013, September). Application of First-Order Logic in Knowledge Based Systems. Retrieved October 23, 2016, from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.392.7815&rep=rep1&type=pdf