Final Project Report (Wumpus World)

Kyle Luu

May 2021

1 Abstract

The Wumpus World is a game where the player is on a 4x4 grid. They have to explore the grid, gaining information about different sources of danger as they do so. The goal is to find treasure and return to the starting position with it. Two different AIs were implemented, one modeling the Breadth-First Search and another modeling the Depth-First Search. It seems like the Depth-First Search was better at playing the game in every measured category.

2 Introduction

2.1 Premise

Imagine an adventurer who enters a temple after hearing tales of a treasure hidden inside. Anyone able to enter the temple, grab the gold, and exit will become instantly rich. However, there exists several complications that prevent just *anyone* from stealing the treasure. The entire inside of the temple is pitch-black, meaning that anyone inside will not be able to see the treasure, instead relying on their other senses. Littered throughout the temple's floor are bottomless pits that would ensure death to any that fall into them. And last but certainly not least, a monster roams the temple. Known as the Wumpus, this terrifying creature stands almost as tall as the temple ceiling and will easily gobble up anyone and anything that comes in its way. After all, it must survive to keep guarding the temple's treasures. The Wumpus is the reason why anyone trespassing through the temple must not bring any sources of light. The Wumpus will be able to see the trespassers' whereabouts, leading to a quick death.

There is a ray of hope for the adventurer, however. It is said that if one were to stand near a pit and concentrate, a faint breeze can be felt. Likely, if one were to stand near the Wumpus, its horrible stench would alert the adventurer to its presence. Lastly, if one were standing right next to the treasure, a subtle glittering sound can be heard. After hearing this, the adventurer feels ready and able to go in and grab the treasure. Perhaps due to hubris, perhaps poverty, the adventurer only brings one arrow. One arrow is good enough to slay the Wumpus anyway.

The adventurer waits outside the temple until the roars of the Wumpus inside dies down, a sign that it has gone to sleep. It is then that the adventurer heads inside the temple in an attempt to get rich.

2.2 Transforming it Into a Game

This is the premise for a game known as The Wumpus World. The player is put in a grid, the layout of which they do not know at the start. Around the grid are pits, a treasure, and a Wumpus. The player can move around the grid at will, but when the player enters a grid adjacent to a source of danger, they are alerted to it. Specifically, when they are adjacent (not diagonal) to the Wumpus, they smell a stench. When they are adjacent to a pit, they feel a breeze. When they are in the same cell as the treasure, they hear glittering. If the player enters the same cell as either the Wumpus or a pit, they die, losing the game. The player is also given 1 arrow, which when shot, flies straight in the direction they are facing. If the arrow strikes the Wumpus, a scream can be heard, signifying the death of the Wumpus. Otherwise, nothing is heard and the Wumpus still lives. If the Wumpus dies, the player no longer dies by entering the same square as it. The goal of the player is to get the gold and return to the square they started in.[1] [3]

2.3 Concrete Details

For the purposes of simplicity, when I implemented the game, I kept many things consistent between the different grids. Each one is 4x4, small enough where decision trees do not get too large, big enough where there is still plenty of room for complexity. The player always starts in the south-west cell, facing north. There is only ever one Wumpus and one treasure in the grid. Pits can vary, however, and each square other than the starting square and those adjacent to it has a 0.2 chance of containing a pit. Using these constraints, it became easier to not only program the game, but also to program multiple AIs to play it. [1] [3]

2.4 Artificial Intelligence

Once the game was made and able to be interacted with by a human, I started creating two different AIs that play the game. Specifically, they functioned based on two different common searching algorithms, the Breadth-First Search, and the Depth-First Search. In summary, when faced with a decision tree, the Breadth-First Search will 'search' through every branch at once, going down the different branches only after it has searched through every other branch of the same level. The Depth-First Search will explore one branch only until it has been completely searched, from that point it switches to the nearest branch. A way to visualize these searches is shown in Figure 1.

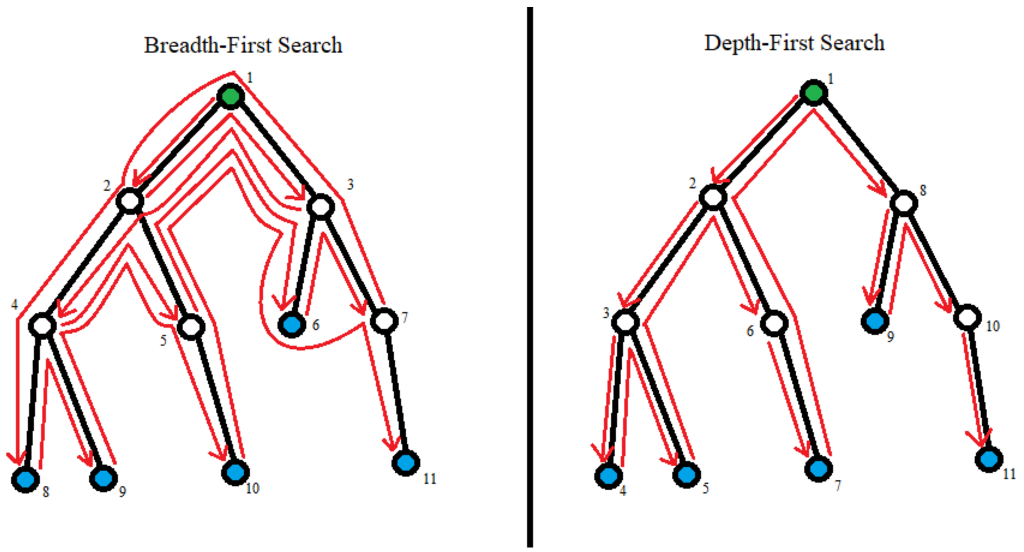


Figure 1: Breadth-First and Depth-First searches. Follow the arrows to see the order in which they find nodes. The order is also denoted by numbers next to the nodes.

3 Background

3.1 Wumpus World Details

Other than details mentioned before, there are some more details about the implementation. The player can only move forward, but can turn 90°to the left or right. If the player tries to walk into one of the walls, they don't move. The agent is not required to shoot the wumpus, but doing so may help by making that square safe. It may also open up a path that wasn't available before. [1] [3] [7]

3.2 Uninformed Searches

The two main algorithms that will dictate how the agent plays are the Breadth-First Search and the Depth-First Search. The Breadth-First Search will prefer to explore squares closest to the starting point and iterating outward, while the Depth-First search will prefer to explore in a certain direction, taking other directions when hitting a wall or when avoiding danger. To describe the searches in detail, cells in the grid will be described using a coordinate system. The bottom-left corner is cell {1,1}, and the top-right corner is cell {4,4}. Moving up increments the y-coordinate while moving down decrements it. Moving to the right increments the x-coordinate while moving left decrements it. The agent cannot have coordinate values below 1 or above 4.

The Breadth-First Search will expand in a way that looks like it radiates outward. Starting at cell {1,1}, it will try to explore cells {2,1}, then {2,2}, {1,2}, {3,1}, {3,2}, {3,3}, {2,3}, {1,3}, {4,1}, {4,2}, {4,3}, {4,4}, {3,4}, {2,4}, and finally {1,4}. The Depth-First Search will expand row-by-row from bottom to top, going left to right. These are shown on

Figure 2. [6]

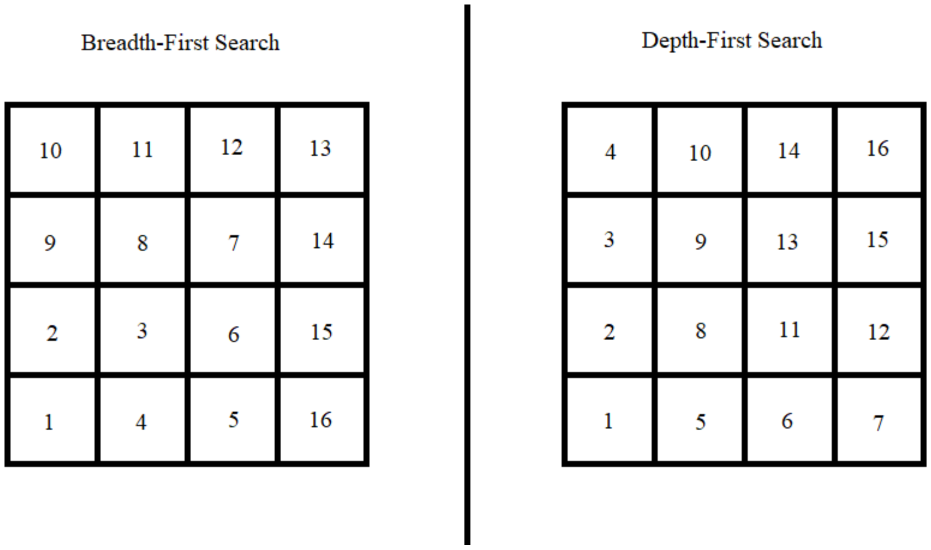


Figure 2: The orders in which the two algorithms will explore the tiles on the grid. The numbers show the order.

3.3 Informed Searches

Usually, when the idea of searches comes up in a programming situation, some informed searches get brought up. The two most popular ones are Dijkstra's Algorithm and the A* algorithm. Dijkstra's Algorithm requires maintaining different paths to the goal and only choosing to expand the ones that will cost the least to reach the goal. While it is good at finding the most optimal solution, the only way of expanding different paths at different times in a Wumpus World situation would be to physically move the character between the different paths. Since movement lowers the score, I suspect that it won't do well. Regardless, I won't be implementing it due to time constraints.[8] [2]

The A* Algorithm is just unfeasible in a Wumpus World situation. This is because it has to take into account some function that estimates the cost to travel to the gold. However, since the gold's location is unknown, it is impossible to devise a heuristic that could estimate the cost to retrieve the gold from the agent's current location (unless the gold has already been revealed, but once that happens, the agent is already in the same cell as it). Thus, the A* algorithm won't be implemented as a way to solve the problem. [8] [5]

3.4 Machine Learning

In addition to informed searches, Machine Learning also gets brought up often. While I am not going to be implementing such a solution, pondering it is worthwhile. Luckily, someone has implemented a genetic algorithm to play the game. The algorithm was given a cost per action and a priority on which to act given a specific circumstance. Important in genetic algorithms is some measure of ”fitness”, or how well each iteration or generation performs. The Wumpus World has its own scoring system, which the algorithm uses to determine its fitness. With all these constraints, the genetic algorithm was able to develop a strategy to always win the game. [4]

4 Approach

First, the game had to be implemented. This was done in Python (the language I'm most familiar with). The first thing the game does is generate a board. It makes a 4x4 array of strings, each equal to the string ” ”. Then, the user is placed on the bottom-left, signified by the string ”U”. Then, the pits are placed. For every cell that isn't the bottom left or those adjacent to it, it generates a pseudo-random number between 0 and 1 and if it is below 0.2, it puts a pit in that cell. Next, It places a Wumpus in one of the remaining empty spaces. Finally, it puts the gold in one of the remaining empty spaces.

4.1 Making a Playable Grid

Using this approach to creating the playing grid, it is possible to generate a grid that is impossible to solve (i.e. the gold is surrounded on all sides by pits). Due to this possibility, the code had to know if the generated grid is solvable, and if it is not, to generate a new one. Determining if a grid is solvable took some effort. In summary, it tries to find a path from the starting position to the gold. It does not need to find a way back because the player can always just backtrack to get back to the start. Finding the path is what took some effort. Basically, the code has to test every combination of movements and if any reach the gold without going through any pits, then the grid is solvable. Even this method needed some refinement. On its own, it would reach the maximum recursion length (recursion was used). However, once the code was changed so that if it is pondering a path that it has been on before (if it is retreading already-explored territory), it no longer ponders it. If one of the paths reaches a pit, then the path is cut off, no longer to be pondered. With these refinements, the code is now able to determine whether a board is solvable by using a reasonable amount of recursion. The code does not need to worry about the Wumpus because there is only every one and if it blocks a path to the gold, the player can just shoot it.

Once the grid itself was made, several supplemental functions were implemented to help future code interact with the grid more easily. The first one can be called to ”sense” the cell the user is. It returns an array containing all the different sense the player would sense. Specifically, it checks for a pit or a Wumpus and adds to the array ”Breeze” or ”Stench,” respectively. It also checks if the player is in the same cell as the treasure and adds ”Glitter” to the array. There is also a function to shoot the arrow, modifying the board to remove the Wumpus if the arrow hits it, one to turn the player left or right, changing what direction they are facing, one to move the player one cell in the direction they are facing (provided they are not dead and are not facing a wall), one to pick up gold if the player shares the cell with the treasure, a function to see if the player has won (the player is holding the gold and is in the south-west corner, and one to see if they had lost (if they are in the same cell as either the Wumpus or a pit). The actual game function uses all of these functions to help the game work.

4.2 The Scoring System

As explained previously, the game has a scoring system. Every time the player moves successfully, the score decreases by 1. Every time the player shoots an arrow, the score decreases by 10. Finally, if the player wins they are awarded 1,000 points. This scoring system leads to a maximum of 995 points: 1,000 for winning, -1 for moving forward, -1 for picking up the gold, -2 for turning around twice, and -1 for moving back to the original square. The player gets punished for firing the arrow, which means the game treats killing the Wumpus as something only needed to be done as a last resort. The AIs I implemented did not hold back when it came to killing the Wumpus, however.

4.3 The Game

The game itself keeps track of many different variables. It has a score, if the Wumpus has been shot, if the arrow has been shot, whether the player has decided to move last turn and whether the movement was successful, the direction they are facing, whether they are holding the gold, and the amount of turns that have been taken. At this point, the game starts a loop that only ends under three conditions: the player has won the game, the player has lost the game, or the loop has iterated 1,000 times. The last break condition exists only to terminate any AIs that run into infinite loops (which does happen). The game is also responsible for appending the "Scream" to the array of senses when the Wumpus dies. The loop begins by getting the player's senses and adding a "Scream" to it if the Wumpus died in the previous iteration. It then calls the AI function (which was passed in as a parameter) with the senses as a parameter. The AI is responsible for keeping track of where they are in the board, where they are facing, if they shot their arrow, etc. The only thing that the AI is given is the senses for the cell they are in. Fortunately, this is enough to keep track of the game. The AI function returns what action they wish to take given the state of the game they think they're in. The game function takes the output of the AI function and applies the specific action to the game (either moving, turning, shooting, or picking up the gold).

4.4 The AIs' Assessments of Danger

The core of both AIs function the same. The only difference between the two are the order in which they prioritize exploring the grid. This is manually entered into both AIs. The order they try to explore the grid is shown in Figure 2.

The AIs start when the main function is called. Upon being called, they update their internal board with the senses they are given from the game. Using the internal game board, they try to pinpoint where the Wumpus and pits are. Specifically, the AIs keep track of 2 different grids of the same size as the game board. Each cell correlates to the corresponding one on the internal game board. The cells contain a float indicating the likelihood that the corresponding source of danger is in that cell. The algorithm calculates how many unexplored cells are neighboring the one they are scanning, and adds the reciprocal of the number of neighboring, unexplored cells (1/n) to their probabilities. A way of visualizing this is shown in Figure 3. For the north-west cell, there is a breeze meaning the only unexplored cell next to it has a 100% chance of containing the pit. The number in that cell, however, is 1.5. This is because the cell below it is also a Breeze, but its surrounded by two unexplored cells. 1/2 = 0.5, so it adds 0.5 to each of its neighbors. The calculations are done for every cell.

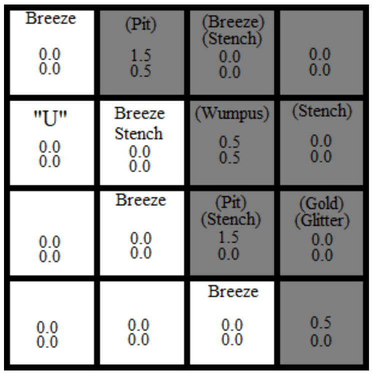


Figure 3: An example of a game state and the corresponding probabilities of danger that the program calculates. Cells with a gray background represent unexplored cells. ”U” is where the player is. Words in parentheses are present in the actual game board, which the AI does not know (as those squares are unexplored). The top number in each cell is the probability of a pit being there. The bottom number is the probability of a Wumpus being there. Both probabilities are influenced by the neighboring cells.

Another factor influencing the potential danger of an unexplored cell is whether or not an arrow has been shot through it. If it has, there is 0 danger from the Wumpus in that cell. If the Wumpus has died, there is 0 danger everywhere from the Wumpus.

4.5 Challenges of the AI Implementations

These probabilities influence the decision of the AI. If the AI wants to move to a specific cell (the order of which is determined by the type of AI, either Breadth-First or Depth-First) and it predicts a pit in that cell, it actively avoids it by going around the pit. If the AI predicts the Wumpus in the cell it wishes to move into and it has an arrow, it shoots the arrow at the cell. If it is out of arrows, however, it treats that cell as if it were a pit (avoid at all costs). If, for some reason, there are 2 cells that the Wumpus could potentially be in, the AI will shoot the arrow at one of them. If the arrow hits nothing, then the AI knows that the Wumpus is in the other one.

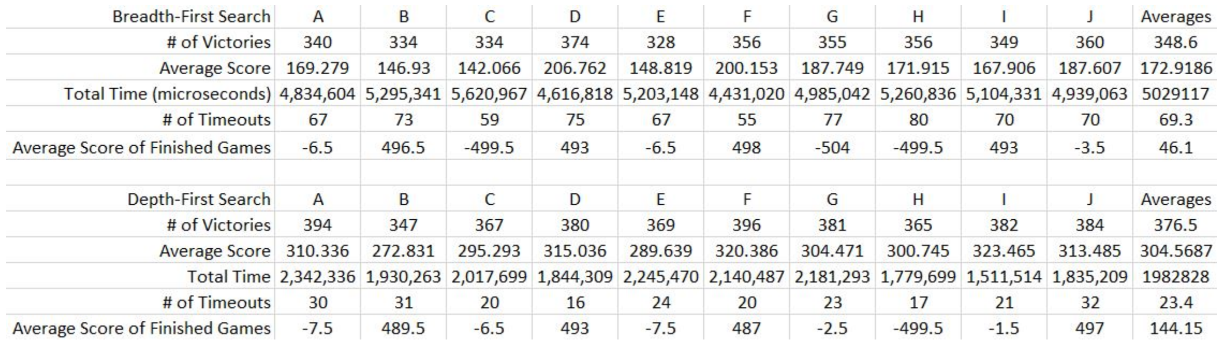
The toughest part is figuring out a path for the AI to get to a specific cell. Not only must it figure out the right order of turns and movements to get to the target cell, but it also must actively avoid sources of danger. I was able to implement the pathing well (tested on a grid with no sources of danger). However, when both the Wumpus and pits are in the grid, there are many situations in which the AI will actively try to avoid a pit by moving around it, but when it is on another side of the pit, the pathing takes it back to where it used to be. This would lead to infinite loops. While I am sure this could be fixed with more refinement, I was not able to do so because of time constraints. Instead of fixing this, I made the game terminate after 1000 turns no matter what and entering a fail-state if it has done so.

Other than that, the different AIs explore the grid in the order given by what type of search they emulate. They have a list of coordinates that determine the order in which they explore the grid. Every time they reach a goal (or are unable because of a source of danger), they set their goal to the next coordinate in the list. If at any time they find the gold, they pick it up, discard all notions of goals, and instantly head back to the south-west corner of the board.

5 Experimentation

5.1 The Data

Finally, one last algorithm was implemented to run the different AIs and collect relevant information from them. It generates a board, and copies it. It runs the game using the Breadth-First AI using one copy, and then it runs the game using the Depth-First AI using the other copy. 10 sets of 1,000 games were run per AI, resulting in a total of 10,000 games run per AI. The program collected information about the amount of victories, the average score of the 1,000 games, the total time it took each game to run, the number of timeouts (the AI took over 1,000 turns), and the average score of the games that did not result in a timeout. The results are in Figure 4.



Breadth-First Search	A	B	C	D	E	F	G	H	I	J	Averages
# of Victories	340	334	334	374	328	356	355	356	349	360	348.6
Average Score	169.279	146.93	142.066	206.762	148.819	200.153	187.749	171.915	167.906	187.607	172.9186
Total Time (microseconds)	4,834,604	5,295,341	5,620,967	4,616,818	5,203,148	4,431,020	4,985,042	5,260,836	5,104,331	4,939,063	5029117
# of Timeouts	67	73	59	75	67	55	77	80	70	70	69.3
Average Score of Finished Games	-6.5	496.5	-499.5	493	-6.5	498	-504	-499.5	493	-3.5	46.1

Depth-First Search	A	B	C	D	E	F	G	H	I	J	Averages
# of Victories	394	347	367	380	369	396	381	365	382	384	376.5
Average Score	310.336	272.831	295.293	315.036	289.639	320.386	304.471	300.745	323.465	313.485	304.5687
Total Time	2,342,336	1,930,263	2,017,699	1,844,309	2,245,470	2,140,487	2,181,293	1,779,699	1,511,514	1,835,209	1982828
# of Timeouts	30	31	20	16	24	20	23	17	21	32	23.4
Average Score of Finished Games	-7.5	489.5	-6.5	493	-7.5	487	-2.5	-499.5	-1.5	497	144.15

Figure 4: The results of the data. Each set is denoted by a letter of the alphabet and each set represents 1,000 games run per AI. Each of the 1,000 games are run with different boards, but the same board was used for both AIs for each iteration.

5.2 Analysis of Data

Broadly, it seems like the Depth-First Search was the better algorithm in every measured way. It won more often, finished with higher scores, finished faster, and timed out less. Even if all the timeouts are excluded, the Depth-First Search scored higher on average. This was surprising to me, since the only difference between the two AIs is the order in which they prioritize exploring the grid. As such, the differences between the two cannot be attributed to differences in other attributes such as obstacle-avoidance, pathing back to the start, etc, as no such differences exist.

5.3 Surprises

All around, the corresponding data points between each of the data sets are fairly consistent except for one category, the average score of the games that were finished. surprisingly, this category is the only one where the Breadth-First Search was able to surpass the Depth-First Search occasionally. Another surprising face was that some games that finished ended up with an average score of around -500. That means that the AI moved around for 500 turns before entering a pit or the Wumpus. Scores of around +500 can be explained because they are the average of 1,000 games. An average of +500 shows that on average, half of the games were lost quickly (scores closer to 0) and the other half were won (scores closer to 1,000). The Breadth-First Search had an average score for finished games of around -500 for three sets, while the Depth-First Search only had that for one set.

5.4 Explanations

It seems that the Depth-First Search does better because of the how if functions: it turns less often and backtracks more. I think this contributes for its victories because turning appears to confuse the AIs more. The algorithms I implemented often check for sources of danger when they are about to move into an unexplored cell. Turning does not seem to interact kindly with this, as it changes the trajectory of the path without changing the coordinates of the player. Since the Depth-First Search turns less often and backtracks to familiar and safe cells less often, it is able to "get its bearings" better than the Breadth-First Search. Whereas the Breadth-First Search is constantly exploring new cells, the Depth-First Search is more able to methodically explore new cells, pretty much only entering them from the south or west side. The Breadth-First Search enters new cells from all of the four sides. This discrepancy helps to explain the differences in the data. In fact, it seems like for many of the categories, the Depth-First Search scores around twice as better than the Breadth-First Search. For example, the average number of timeouts for the Breath-First Search was 69.3 and 23.4 for the Depth-First Search. The Breadth-First Search's average time to play 1,000 games was around 5 seconds while the Depth-First took around 2 seconds to play the same amount of games.

Something that could be noted is that both algorithms never won over 500 times in any set. The most wins was for Set F in the Depth-First Search, which won 396 times.

Since these numbers are below 500, that means that both these algorithms are losing more than they are winning. I do not attribute these losses to a fault of the searches the AI are modeled after, but rather to my implementation of the searches, which was imperfect. If both searches played perfectly, however, I think that the Breadth-First Search will perform better than the Depth-First Search. This is because of it spends more time exploring and potentially uncovering the treasure than the depth-first search does. It would spend less time backtracking, leading to higher scores and faster times.

6 Conclusion

The Wumpus World provides a simple situation in which Artificial Intelligence can be implemented in many different ways to solve its primary problem. I will be implemented methods based off of the Breadth-First Search and Depth-First Search to see which one can get the highest score when beating the game. After running 10,000 games for each AI, the Depth-First Search was able to perform better, scoring higher, losing less, and timing out the game less.

6.1 Moving Forward

There are many ways to switch things up to test the AI even more. However, if I were to put more effort into the tests, I would start by refining the existing AIs even more. They ran into infinite loops and lost *way* more often than I wanted. I am sure that this is due to how it reacts to danger, since when the AIs were tested on grids with no pits or Wumpuses (Wumpi?), they were always able to get the treasure and return to the starting position. As soon as just 1 Wumpus was added, the AIs started running into infinite loops and losing.

Once that is done, the AI can be modified to react to different game scenarios. Since the game is fairly simple, different aspects can be changed to provide more variety. For example, the board size can be changed, the number of Wumpuses can be changed, the frequency of pits can be changed, and the original placement of the player can be randomized. In addition, Wumpuses can be given different AIs to provide even more complexity. Wumpuses could be set to move about randomly, target the player, guard the gold, stand still until the player is within a certain distance of it, or stand still until a certain amount of turns have passed.

Another way to move forward would be to implement other AIs, such as one that follows the Dijkstra's Algorithm (possibly only exploring cells that have the least danger), or a Machine-Learning algorithm to play the game. The Wumpus World already has a scoring system, which would help the Genetic Algorithm learn to play the game better.

One last way to vary the game would be to change the scoring system. It currently treats killing the Wumpus as something that should only be done as a last-resort. If killing the Wumpus awarded points, the player can be incentivized to do so.

6.2 Wrapping Up

The Wumpus World, for being a simple game with a simple premise, has resulted in a surprisingly complex experiment. Its correlation to existing search algorithms provided a way to implement different AIs. Specifically, the Depth-First Search has had the more success than the Breadth-First Search. If I were to implement a Dijkstra's Algorithm or use Machine-Learning to play the game, I'm sure they would do better than the Depth-First Search. Even still, the Depth-First Search was very dependable.

References

[1] Koen V. Hindriks and Wouter Pasman. The wumpus world, February 21, 2011.

[2] Sandeep Jain. Dijkstra's shortest path algorithm.

[3] Sonoo Jaiswal. The wumpus world in artificial intelligence.

[4] Gul Muhammad Khan. *Evolution of Artificial Neural Development In Search of Learning Genes*, volume 725. Springer International Pubishing, 2018.

[5] Amit Patel. Introduction to a*: From amit's thoughts on pathfinding.

[6] S Pooja, S Chethan, and C V Arjun. Analyzing uninformed search strategy algorithms in state space search. *International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)*, 2016.

[7] Sabastian Sardina and Stavros Vassos. The wumpus world in indigolog: A preliminary report. *IJCAI-05 Workshop on Nonmonotonic Reasoning, Action, and Change (NRAC'05)*, pages 92–95, 2005.

[8] Dmitry S. Yershov and Steven M LaValle. Simplicial dijkstra and a* algorithms for optimal feedback planning. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011.