CAP 4601 Project 2: Wumpus World

Alexander Bachmann and Joshua LaRocca

Background:

Wumpus World is the representation of a simple world where an explorer searches a dark, dangerous cave in search for a bounty of gold. In this cave, there are two threats to the explorer's life: falling in bottomless pits and being slain by the Wumpus. The explorer's goal is to find the gold then exit safely by backtracking through the cave. Typically, the cave is represented by a 4×4 grid or 16 rooms. Each room will have indicators that can be detected by the explorer. If the explorer smells a stench that means that Wumpus may be in an adjacent tile. If the explorer feels a breeze that means that a bottomless pit may be in an adjacent tile. If the agent sees glitter that means that there is gold in the current room and upon retrieval of the gold, the explorer is allowed to leave the cave. While exploring the cave, the explorer gathers knowledge and acts according to the gathered knowledge; this type of behavior makes the explorer a knowledge-based agent.

The following figures will be used to explain the agent's ability to gather knowledge from its sensors, store that knowledge, then make predictions about the locations of threats throughout the cave.

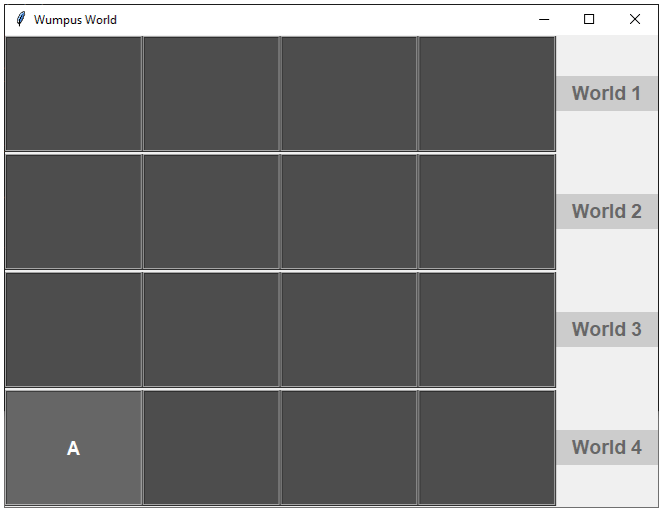

Figure 1: initial starting position of agent

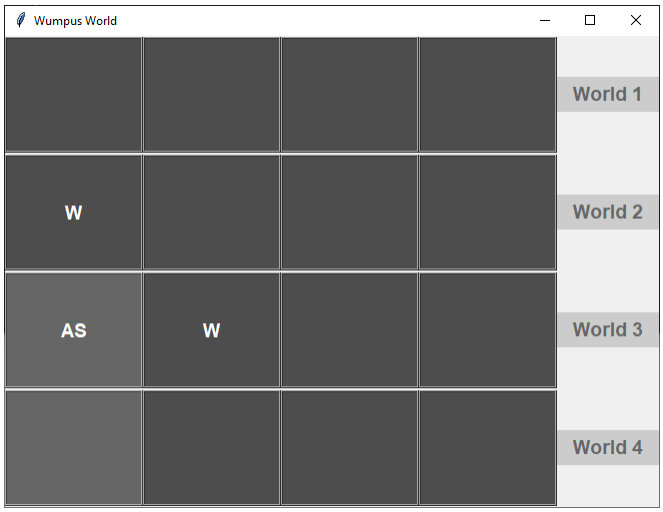

Figure 2: agent moves up

In Fig. 1, we see that the agent, represented by 'A', enters the cave from the bottom left [3, 0]. Upon entering the cave, the agent does not perceive any threat indicators. In Fig. 2, we see that the agent smelled a stench [2, 0], represented by 'S', indicating that the Wumpus must be in an adjacent tile [1, 0] & [2, 1]. From this indicator, the agent predicts all potential locations of the Wumpus and stores that information in its knowledge base.

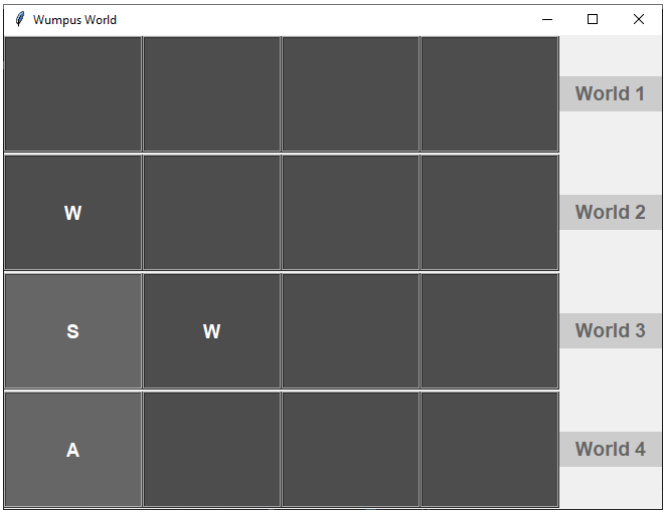

Figure 3: agent moves down

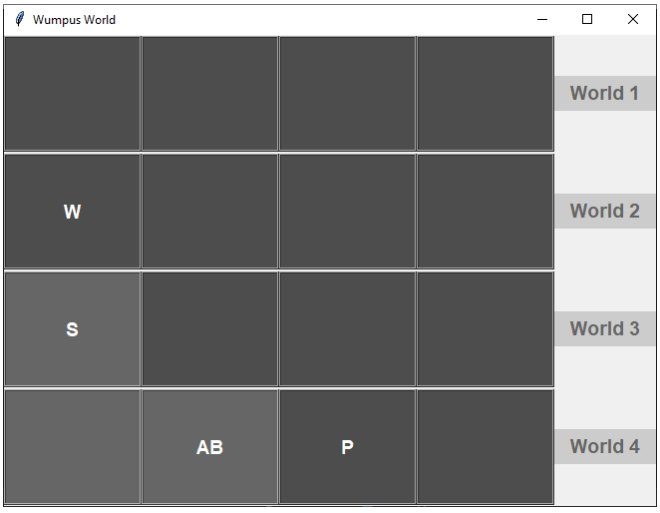

Figure 4: agent moves right

In Fig. 3, we see that the agent was forced to backtrack [3, 0] because there was no perceptibly safe move given its current knowledge base. In Fig. 4, we see that the agent traveled one tile to the right [3, 1]. In this tile, the agent sensed a breeze indicating that a pit could *only* be in [3, 2] because a breeze was not sensed when the agent visited [2, 0] in Fig. 2. In Fig. 4, we also see that the agent further updated its knowledge of the cave by removing its prediction of there being a Wumpus in [2, 1]; the agent was able to remove this prediction because it did not smell a stench in its newly visited tile (all indicators must be adjacent to the possible threat).

Methods and Algorithms:

Our program begins by reading and parsing a text file. The text file is a representation of the world for the agent to solve. The file begins by specifying the dimensions of the cave followed by the starting coordinates of the agent which must be followed by the location of the Wumpus, gold, and all of the pits. An example of this text file can be seen in Fig. 5.

Upon initialization of a world, a three-dimensional Python list is populated using the coordinates provided from the world generation text file. After the agent, gold, Wumpus, and pits are populated, their respective indicators are generated. We chose to generate breezes and stenches for our users because otherwise creating new world text files would feel very cumbersome; the indicators would almost quadruple the number of lines in the text file because every threat can have up to four adjacent indicators.

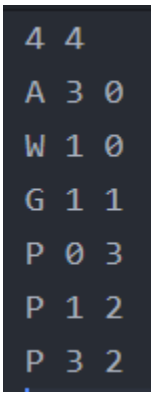

Figure 5: text file used to generate the world

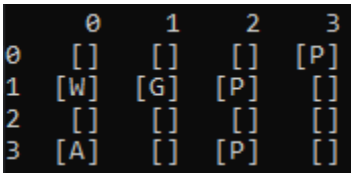

Figure 6: World representation before generating indicators

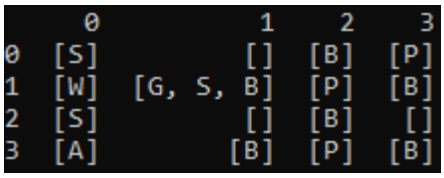

Figure 7: World representation after generating indicators

At the beginning of the agent's exploration, it knows nothing of the cave other than the dimensions of the cave. As the agent explores the cave, it senses and updates its knowledge and threat predictions based on that knowledge. Wumpus must have a stench in all adjacent rooms and a pit must have a breeze in all adjacent rooms. If either of these requirements are broken, then the agent can confirm its prediction was mistaken and update its knowledge accordingly. Below, the figures represent how the agent stores its understanding of the world in its knowledge base, a 3D Python list, and the legend that the agent uses to update its knowledge base.

(Fig. 9-12 are direct representations of the agent's knowledge base from Fig. 1-4)

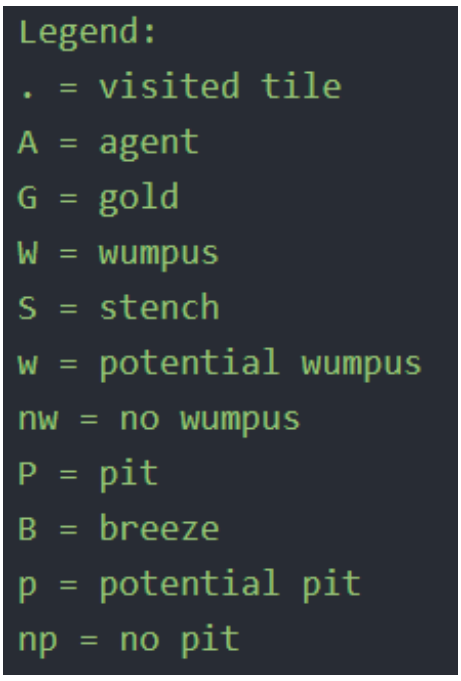

Figure 8: knowledge base legend

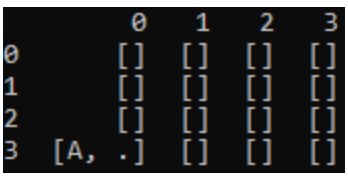

Figure 9: agent initial starting location

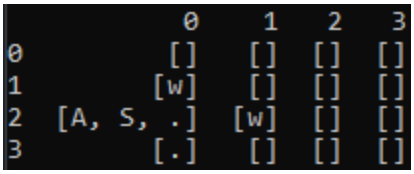

Figure 10: agent moves up

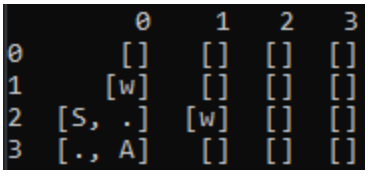

Figure 11: agent moves down (backtracked)

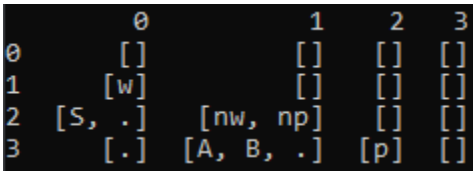

Figure 12: agent moves right

```
def move(direction):
    successful_move = False

    # move up
    if direction == 'u':
        if is_safe_move('u'):
            move_up()
            successful_move = True

    # move right
    if direction == 'r':
        if is_safe_move('r'):
            move_right()
            successful_move = True

    # move down
    if direction == 'd':
        if is_safe_move('d'):
            move_down()
            successful_move = True

    # move left
    if direction == 'l':
        if is_safe_move('l'):
            move_left()
            successful_move = True


    if successful_move == True:
        add_indicators_to_knowledge()
        mark_tile_visited()
        predict_wumpus()
        predict_pits()
        clean_predictions()
        confirm_wumpus_knowledge()
```

Figure 13: simplified pseudocode for the agent's move function

The agent's tile choice is not particularly sophisticated: it moves clockwise with each time-step. The agent will attempt to move up; if that move fails, the agent will attempt to move right. The agent will continue this clockwise movement until a successful move is found. If the agent cannot find a successful move, it will backtrack.

In Fig. 13, to the left, we see that the movement to a new tile is governed by whether or not that direction is a safe move for the agent to make. A safe move is determined by whether or not there is a threat in the attempted tile. The agent must be certain that a tile is completely safe for it to move into that tile. Even if the threat is just a prediction, the agent will not risk failure and will instead seek out an alternative path. Upon successful move, we see that six functions are called.

These six functions are all used to update the agent’s knowledge base and are called every time-step that an agent moves successfully.

add_indicators_to_knowledge(): by referencing the 3D list that represents the world if Fig. 7, the agent updates its knowledge of its world. If an indicator (breeze or stench) exists in the world, the agent adds the indicator to its knowledge.

mark_tile_visited(): if the tile the agent is current in is new to the agent, then leave a bread crumb, represented by a period.

predict_wumpus(): if there is a stench in the tile occupied by the agent, add a 'w' in the agent's knowledge base to every adjacent tile that has not already been visited by the agent.

predict_pits(): if there is a breeze in the tile occupied by the agent, add a 'p' in the agent's knowledge base to every adjacent tile that has not already been visited by the agent.

clean_predictions(): Every threat must have an indicator adjacent to it. These threats can be ruled out if an indicator is missing from an adjacent tile. For every tile in the agent's knowledge base: if a tile has a predicted threat and if the threat does not have all necessary indicators, then mark it as a 'nw' (no Wumpus) or 'np' (no pit).

confirm_wumpus_knowledge(): The primary difference between Wumpus and pits is that there can only be one Wumpus. This function is necessary because it further confirms the knowledge of the Wumpus. Since there can only be one Wumpus, that means that as more stenches are discovered, the possible locations of the Wumpus drastically decreases. The agent keeps track of the number of stenches it has encountered. For every tile in the agent's knowledge base: if a tile is predicted to have the Wumpus but that prediction has fewer stenches around it than the agent has experienced then it can be said that there is no Wumpus in that tile and is marked as 'nw' (no Wumpus).

Agent Behavior:

Bread crumb exiting:

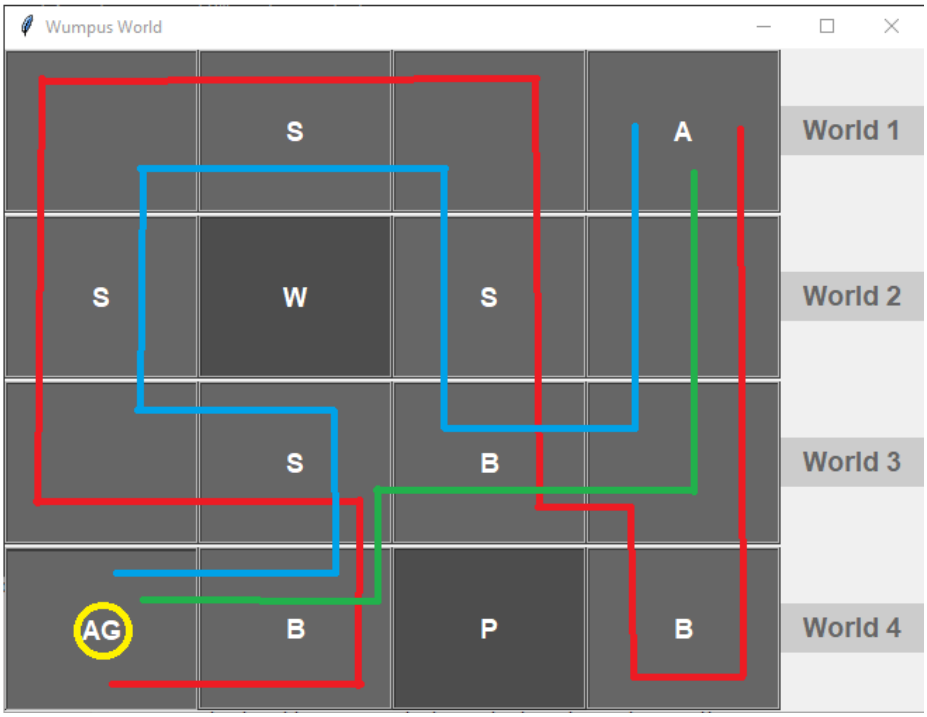

Figure 14: Representation of the agents movement through World 4

Above, Fig. 14 details the agents movement through World 4. The agent enters the cave from the top right of the matrix and explores the cave down to the bottom left of the matrix. The agents exploration for the gold is indicated by the red line. Upon retrieval of the gold, the agent backtracks through the cave, indicated by the blue line.

This backtracking is inefficient in terms of number of steps taken. We are aware of this and still decided to go with a breadcrumb approach for the agent to return safely to the exit. This decision is predicated on the reasoning that a real human would much more likely return out of a dark cave by using a breadcrumb approach. Even if a new path has been revealed as being safe, indicated by the green line, a human is far more likely to take a slightly inefficient route because the most recent route is more solidified in our memory.

Free Exploration:

Traditional Wumpus World simulations reward or penalize the agent depending on its performance. Typically the agent is rewarded or penalized 1,000 points depending on successful completion of the cave and penalized 1 point for each step and 10 points for using an arrow. We chose to create an agent that was not governed by the standard point system because our agent will always solve the same world the same way and our agent will always solve a solvable world. We encourage our agent to freely explore without fear of penalty.

Pacifistic:

We chose to make our agent pacifistic in nature which in the context of Wumpus World means that our agent is not violent; he refuses to use his bow and arrow to harm the Wumpus. Instead our agent explores the cave extensively in an attempt to not have to ever confront the Wumpus even if using the Wumpus room as a shortcut to the gold may have yielded a faster solve of the cave. We wanted our agent to be more representative of a free, pacifistic human explorer – like a Buddhist Indiana Jones.

Conclusion:

Creating a version of Wumpus World proved to be surprisingly difficult. We now see how knowledge based systems can rapidly grow to the point of requiring a colossal amount of maintenance and preparation. The agent in Wumpus World only has to be concerned with two threats and the intricacies of that simple behavior required quite a bit of code. It seems that knowledge based systems are quite hard to develop to an expert level of performance for solving difficult real world problems.

Citations:

The Wumpus world in Artificial Intelligence - Javatpoint. (n.d.). Retrieved from https://www.javatpoint.com/the-wumpus-world-in-artificial-intelligence