Assignment 3

CS440

Abdulrahman Abdulrahman (aa1684), Brian Moran(Btm80), Fares Easa(Fbe6), Jerry Wu (jw1104)

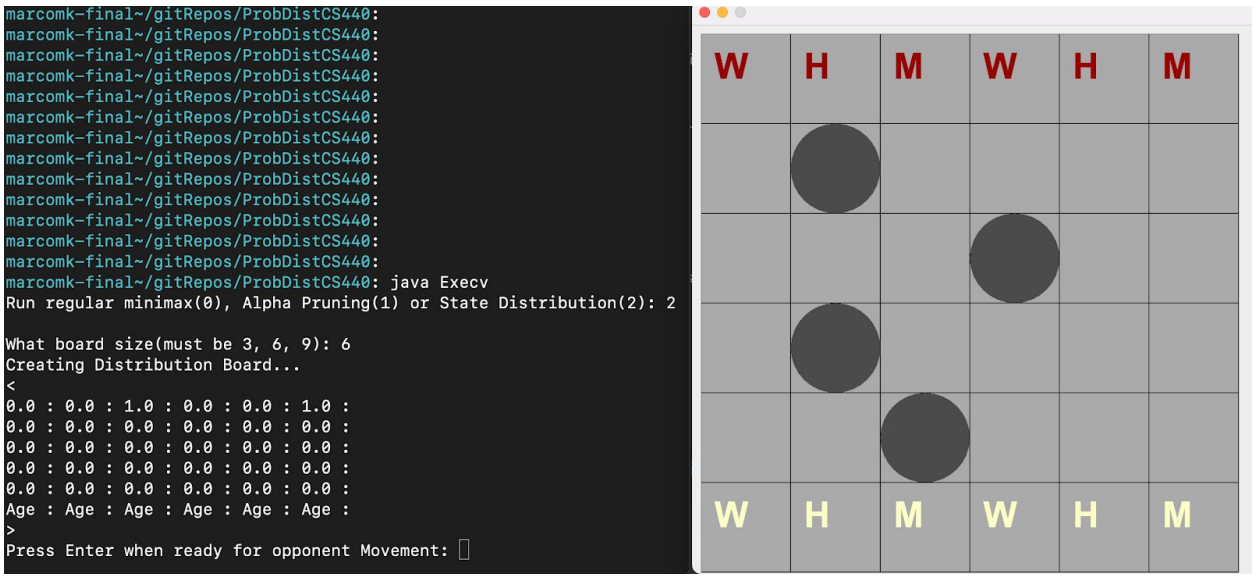
Readme:

Please Email aa1684@scarletmail.rutgers.edu if you have any questions!

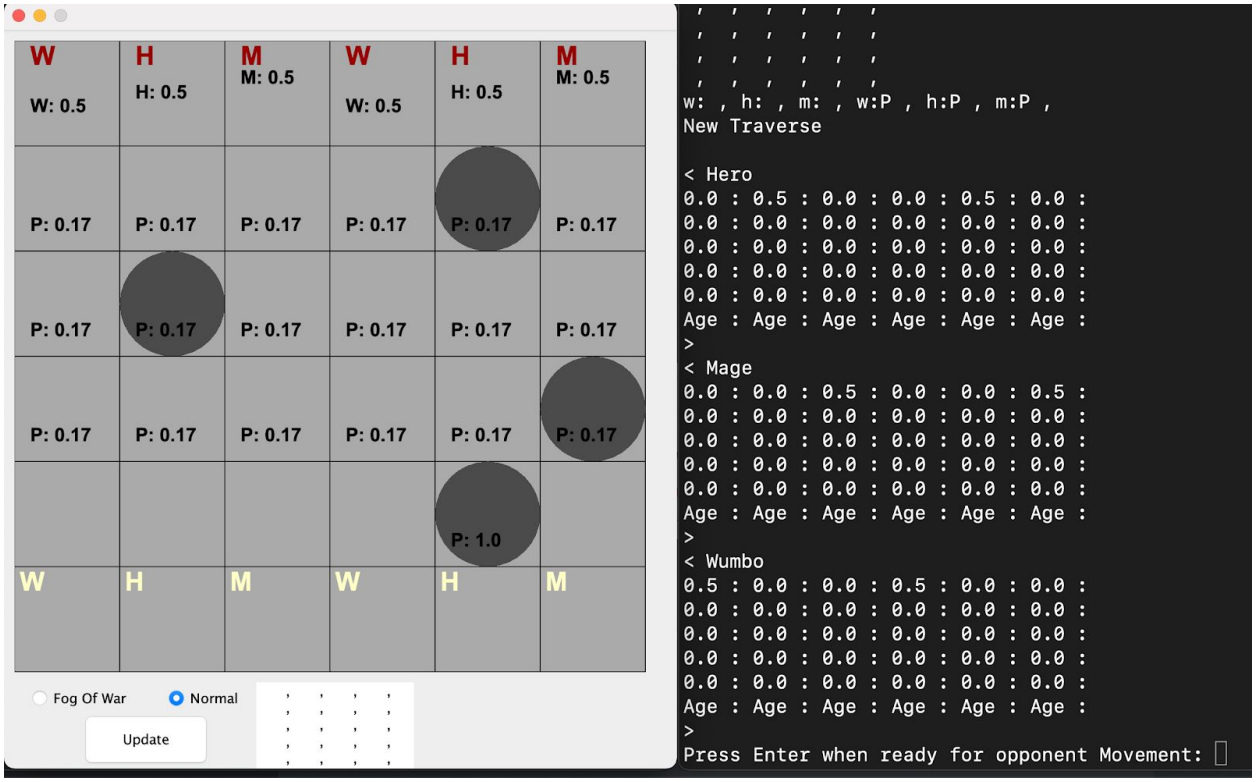
To run the Project, compile the Execv.java file, and run the resulting executable.

```
marcomk-final~/gitRepos/ProbDistCS440: java Execv
Run regular minimax(0), Alpha Pruning(1) or State Distribution(2):
```

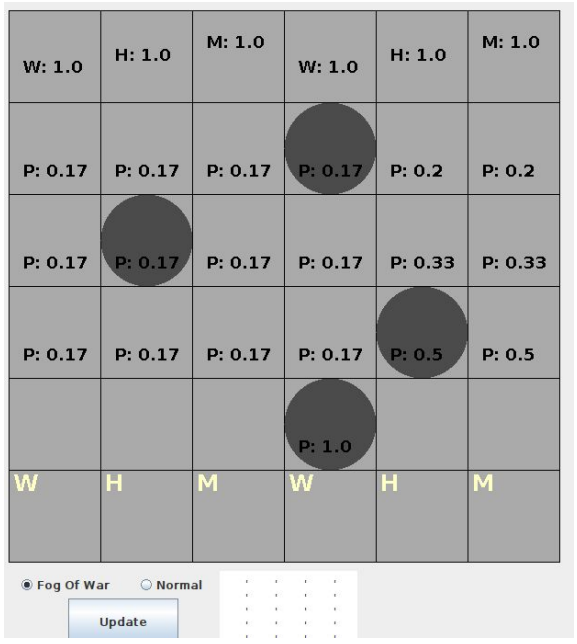
You will be prompted by many options for the Agent, and then a board will appear and will allow you to move.



You can move by clicking on the board and moving your white piece to another part of the board. The letters on the board represent the different pieces for each player, where red is the agent, and white could be the player or agent depending on which is chosen.

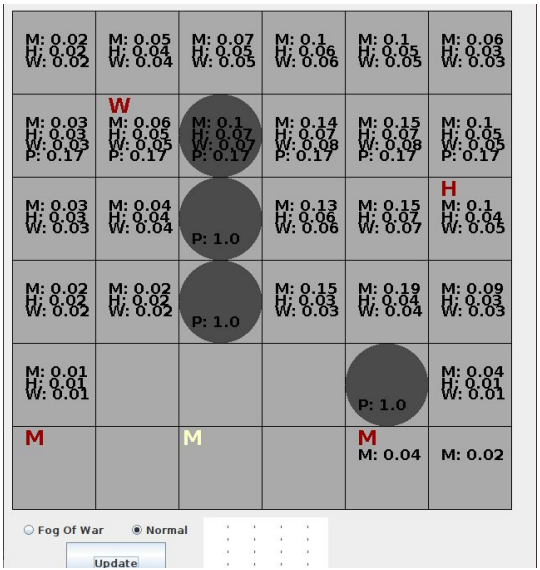


Fog of War functionality can be toggled with the supplied radio button. Click the corresponding button and “update” to get the required functionality. With fog of war selected, no character pieces are shown.



In the figure shown on the left, we can see that fog of war radio button has been selected and that the opposing pieces have been removed from view. All that remains is the probability distribution and player/agent pieces.

Hitting enter will cycle through the game, playing the agents against one another until there are no moves left or a player runs out of pieces.



As we can see on the right, as the game progresses, the probability distribution of what is located in that space continues to progress and continuously updates after every move.

1. Deliverables

1. Formulate the new wumpus game in terms of Boolean variables and define the relationship between these variables. Hint: For each cell you should have a variable for each observation and a variable for each item that the cell can contain (i.e. units, pits). (10 points)
    a. We represent the probability of a location having a certain piece using a Pnode object. The Pnode object holds the 4 probabilities: Mage, Hero, Wumpus and pits, as well as a boolean identifying whether or not the piece is our agent's piece or not. The ProbDist 2D array is a two dimensional matrix that contains the PNode object that contains the corresponding probability values of the pieces according to our board grid value. We update the probability values in the GUI when the program does a complete update of the board through movements, the user can hide the probability values through the Execv.java file in the main method. The board uses the same pieces for the values and the probabilities are listed in the order of Mage, then hero, then Wumpus, then the probability of there being a pit. On the console we printed just the probability of the hero in a grid format for reference and clarification.
2. Update your game interface to display the observations and to include the fog of war option that can be toggled on and off to display and hide the agent's pieces. (10 points)
    a. In our new GUI there are two new components to clarify the values for the probability, to update the fog of war option, and two show the observations at each position. The text box in the bottom middle prints out the string of observations that the agent's pieces have and separates them by commas. The Format is printed out as the letter of the piece followed by the type of observation, where N is noise from the hero, P is breeze from the pit, E is heat from the mage, and S is the stench from the wumpus. The toggle on the bottom left gives the user two options. Fog of war is the option to hide or display the opposing agents pieces, while Normal provides the normal view of all the pieces. To switch between the two settings, switch the toggle and press the update button to make the adequate switches.
3. Implement the method for computing probability that you described in section 3. Modify your interface to display the probabilities $P(P_{x,y})$, $P(P_{x,y})$, $P(H_{x,y})$ and $P(M_{x,y})$ for each cell. (20 points)
    a. The probability was described in deliverable one, and only appears once the user inputs the type of probability distribution that they intend to use. Once inputted, the probabilities will be displayed on each cell of the grid that does not contain the home agents piece. The FOW.java class has the main functions that create the correct probability distribution for the different pieces. The obs 2D array is then measured against the probability of those movements that an opponent can make, and the rest of the probabilities are normalized. The FOW constructor

without the estimator integer parameter is the normal random probability calculator using the 1/c value to decide which piece will move in each location.

4. Propose a policy for selecting a good move given a probability distribution. Implement an agent that uses this policy. (30 points)
    a. In our Execv function, a turntable is made where the order of our game is: The opponent makes a move (creating our probability distribution), the home agent senses the state of the board and updates the probability distribution accordingly, the home agent then decides which move would be optimal for the agent to make a move given this probability distribution, then updates the probability distribution based on the observations made after their movement, and then we loop back to the opponent making the movement. The policy we decided the agent would use to make that movement was to calculate a value based on the probability of the pieces around it. The function FOW.movement() loops throughout the surrounding locations on the grid with a depth of two, and calculates the value based on the probability. The general format for that calculation is the probability of there being a piece that they can take subtracted by the probability of all the bad moves that the piece can have. For example for the Hero, the p(W) - p(M) - p(H) - p(P). We scale the values of the worse probabilities to make the most intelligent movements. Because the hero can get killed by a mage but can take down another hero, they are more likely to choose the taking down of a hero because we scale the probability of the p(M) to a value times two and we divide the probability of a hero by two. When the Agent calculates all the possible movements around them in the board through our pseudo-minimax algorithm the piece then chooses the move with the highest heuristic value at that location. This keeps our agent aware of its surroundings and lets the agent move forward in that best direction. The movement function returns a grid that we set to the main grid and then scales the distribution accordingly.
5. In Section 2 we assume that the opponent will randomly select a piece to move and then move that piece in a random direction. This is not a realistic approximation of the moves the opponent will select. Propose a better alternative for approximating the opponents moves and discuss how you would compute the effect of the opponent's move on the probability distribution given this approximation method. (10 points)
    a. Instead of randomly selecting a piece to move and then moving that piece in a randomly assigned direction, we can more intelligently assign which pieces based on the probability distribution of the surrounding space. Regarding the direction of movement, we can more intelligently predict the direction of movement of the opponent based on their spatial relation to the agent's pieces. In the normal random distribution the probability of a piece staying compared to a probability of a piece moving was the same scale for all of the pieces where moving was 1/c (c being the total number of opponents pieces). This is a random assumption, so it is not a good idea to base the probability on this movement. To counter this we decided to change that distribution based on the location of an agent around it. The logic is, pieces that are next to an opponent are more likely to move compared to ones that are at random locations around the map.

Whether they have the intention of taking a piece or the intention of moving to safety, an intelligent agent would decide to make that move. To account for this in our program, instead of the probabilities of the pieces staying in their normal positions being 1-(1/c), and the probability of a piece staying being 1/c we decided to make them 1-(1/(surroundings + c)) and (1 + around)/(surroundings + c). Surroundings are the number of pieces of our agent that are around the nodes of opponents with probabilities. Around is the number of home/agent pieces that are around that specific square. Those who are surrounded by pieces have a higher probability of being moved, compared to those who are not surrounded by pieces. So their numerators of movement would increase with more surroundings.

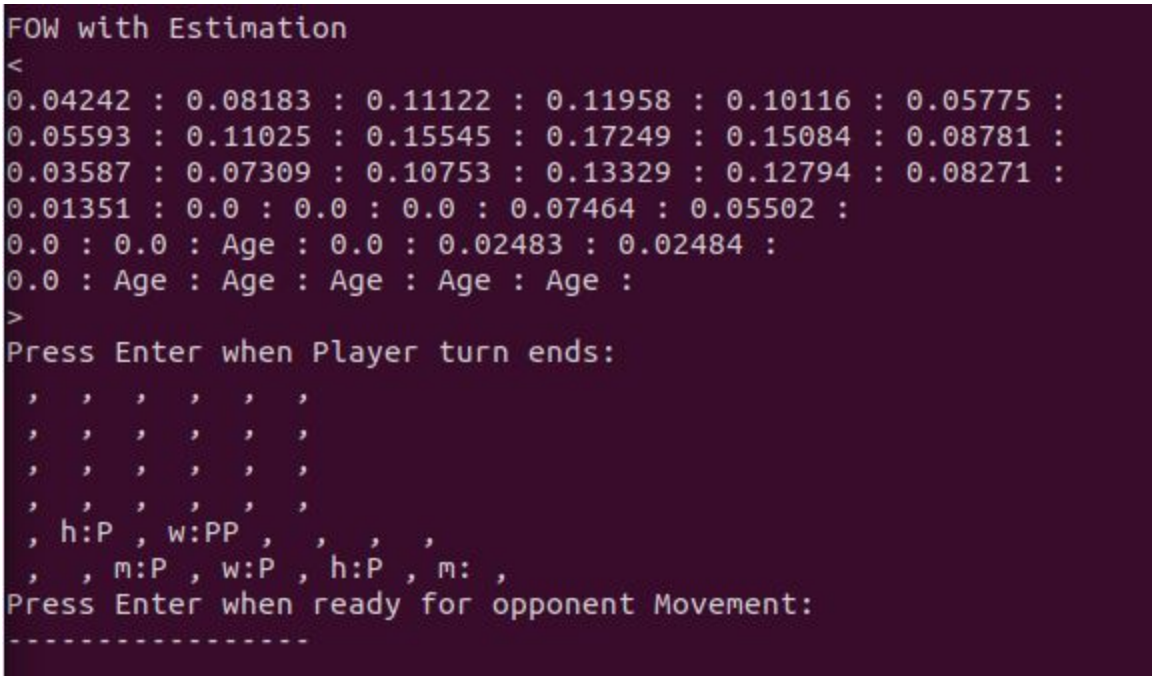
b. We did benchmarks to find out how good the new algorithm runs compared to the random implementation. The following stats show how many games are played and who the winners are.
    i. Random Implementation
        1. Games played: 20
            a. Home Won: 7
            b. Away Won: 13
            c. Draws: 0
        2. Rate: 35%
    ii. Weighted Implementation
        1. Games played: 20
            a. Home Won:10
            b. Away Won: 8
            c. Draws: 2
        2. Rate: 50%

c. Based on the data provided above, our implementation of the Weighted choice opponent distribution, has improved the win rate of the agent by 15%. We also saw more draws in situations where it would be likely for the opponent to win. There is a small sample size of only 20 games, so this does limit the rate aspect of the distribution, however this supports our algorithm's implementation of an optimization of our opponents probability distribution.

6. 6. Implement the transition probability method that you described in question 5. (20 points)
    a. To run our improved probability functions for fog of war, selecting the weighted decision(1) when prompted after creating the probability distribution, runs the second constructor for fog of war, thus allowing the agent to make these predicted movements. You can see the probabilities are changed when you have two pieces sharing an observation where the probability of the two pieces being switched are higher compared to the random decisions.

2. Adding Fog of War to Game

As part of our deliverables package, we have successfully implemented a “fog of war” file aptly titled “FOW.java”. “Fog of war” is defined by the uncertainty in situational awareness experienced by participants in military operations. The term has become commonly used to define uncertainty mechanics in wargames. In our case, it refers to the inability of the agent's pieces to “see” pits or pieces unless the agent's pieces are in direct proximity to a pit or another piece.

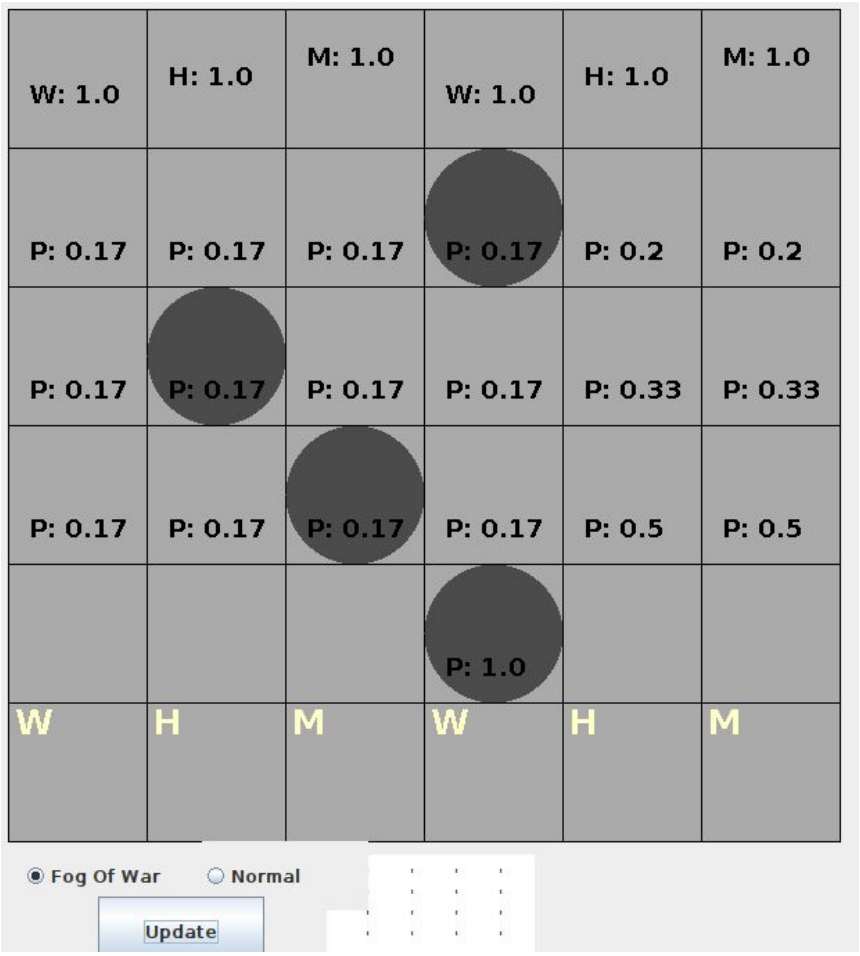


With respect to the scope of the assignment, we have updated our minimax search parameters accordingly: unless a character is in direct contact with another character or pit, it is not reflected in the agent's search mechanics.

3. Probability Distribution of State Variables

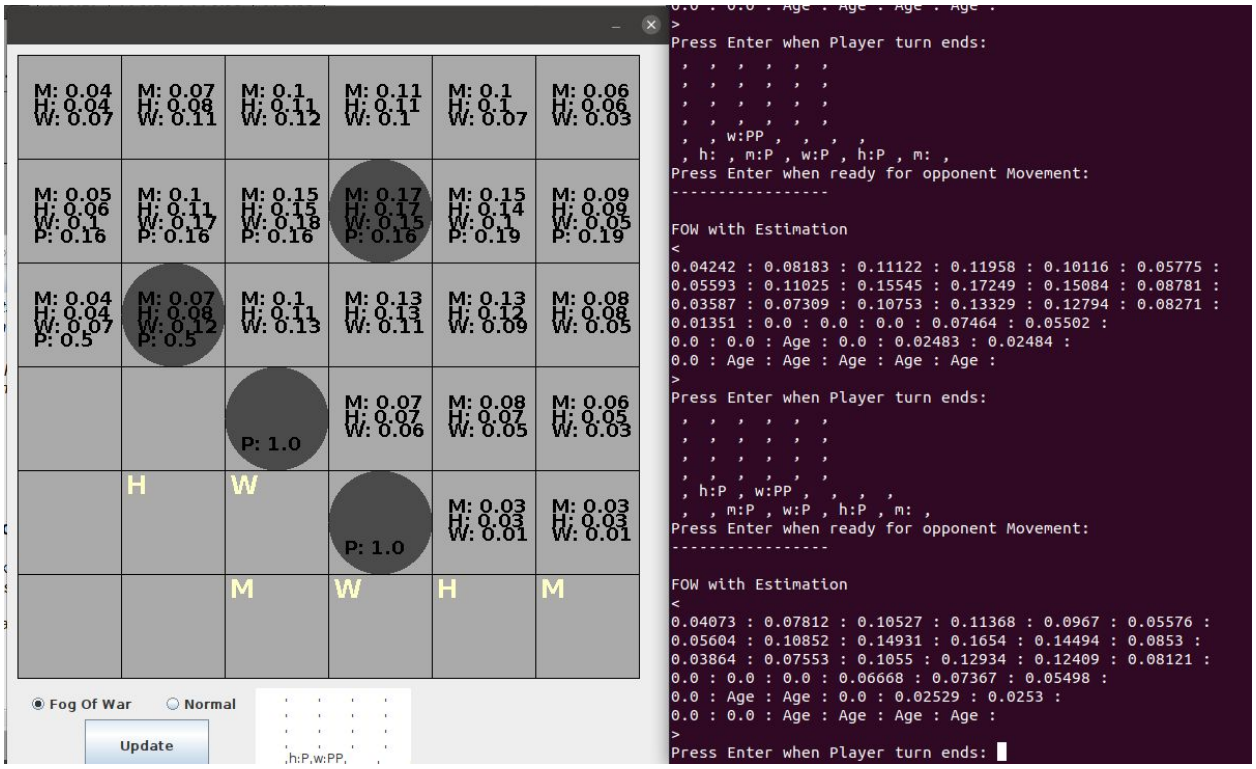
Next, we built a method for tracking the probability distribution of the location of each of the pieces given by the opponent, as well as the probability distribution of the pits.

Initially, because the starting character lineup is always the same, we know where the opponent's pieces will “spawn”. We will also always know if there is a pit in the first row.



Here, we can clearly see that pits (denoted as a 1.0 chance) are not seen by the agent unless the agent's pieces are directly in contact with a pit. Also note that the initial starting position of the opposing player is denoted by 1.0 chance of W, H, M being in their initial positions.

Past the first few moves, the distribution of where various pieces or pits will are updated after every move.


M: 0.04 H: 0.04 W: 0.07
M: 0.07 H: 0.08 W: 0.11
M: 0.1 H: 0.11 W: 0.12
M: 0.11 H: 0.11 W: 0.1
M: 0.1 H: 0.1 W: 0.07
M: 0.06 H: 0.06 W: 0.03
M: 0.05 H: 0.06 W: 0.1 P: 0.16
M: 0.1 H: 0.11 W: 0.17 P: 0.16
M: 0.15 H: 0.15 W: 0.18 P: 0.16
M: 0.17 H: 0.17 W: 0.15 P: 0.16
M: 0.15 H: 0.14 W: 0.1 P: 0.19
M: 0.09 H: 0.09 W: 0.05 P: 0.19
M: 0.04 H: 0.04 W: 0.07 P: 0.5
M: 0.07 H: 0.08 W: 0.12 P: 0.5
M: 0.1 H: 0.11 W: 0.13
M: 0.13 H: 0.13 W: 0.11
M: 0.13 H: 0.12 W: 0.09
M: 0.08 H: 0.08 W: 0.05
P: 1.0
M: 0.07 H: 0.07 W: 0.06
M: 0.08 H: 0.07 W: 0.05
M: 0.06 H: 0.05 W: 0.03
H
W
P: 1.0
M: 0.03 H: 0.03 W: 0.01
M: 0.03 H: 0.03 W: 0.01
M
W
H
M
Fog Of War
Normal
Update
Press Enter when Player turn ends:
Press Enter when ready for opponent Movement:
FOW with Estimation
0.04242 : 0.08183 : 0.11122 : 0.11958 : 0.10116 : 0.05775 :
0.05593 : 0.11025 : 0.15545 : 0.17249 : 0.15084 : 0.08781 :
0.03587 : 0.07309 : 0.10753 : 0.13329 : 0.12794 : 0.08271 :
0.01351 : 0.0 : 0.0 : 0.0 : 0.07464 : 0.05502 :
0.0 : 0.0 : Age : 0.0 : 0.02483 : 0.02484 :
0.0 : Age : Age : Age : Age : Age :
Press Enter when Player turn ends:
Press Enter when ready for opponent Movement:
FOW with Estimation
0.04073 : 0.07812 : 0.10527 : 0.11368 : 0.0967 : 0.05576 :
0.05604 : 0.10852 : 0.14931 : 0.1654 : 0.14494 : 0.0853 :
0.03864 : 0.07553 : 0.1055 : 0.12934 : 0.12409 : 0.08121 :
0.0 : 0.0 : 0.0 : 0.06668 : 0.07367 : 0.05498 :
0.0 : Age : Age : 0.0 : 0.02529 : 0.0253 :
0.0 : 0.0 : Age : Age : Age : Age :
Press Enter when Player turn ends: