1 Part 1

The program starts by building the starting board state. This is represented as a tuple (the set of all rows), containing a tuple (the cells in that row), containing a tuple to hold the number of stones and the current owner of that cell. This board is then used in a loop that checks for a win condition and alternates players. The random player generates all possible children, then selects one at random. The rational player generates all possible children and recurses on them. If a win condition has been reached, or the depth limit is reached the function applies the evaluation heuristic and returns that value. As this function recurses it applies the min max algorithm along with alpha beta pruning to get more optimal values. Eventually it finishes and returns the best possible next move.

The function to generate children works by iterating through all squares on the board and checking if the current user has stones there. If they do it checks all possible directions of movement and if movement is possible it generates a new child board state as if the player were to move that way next.

2 Part 2

The search agent uses the win condition of either player having no available children (or a max depth reached) as its leaf node case. For each potential child state the search function keeps track of the optimal play in a msx/minVal variable. This variable is also used to shift the current alpha or beta values. For convience of return values, the board state of the current best child is stored in a bestChild variable. If the condition for alpha beta pruning is acheived (beta being less than alpha), then the loop is broken and the current best value is returned.

3 Part 3

A few different evaluation heuristics were tried when creating this program. These were evaluated based on the percentage of games they won, and the number of turns it took them to win to break ties.

The first was evaluating the number of potential moves that the rational player could make and subtract the number of moves the random player could make. The second was to evaluate the number of squares owned by the rational player and subtract the number owned by the random player. The final heuristic was to apply a weighting scheme to squares when evaluating owner ship, based on the number of directions you could move in from there. For instance corners were only worth 3 while center squares were worth 8.

Games that exceeded 50 moves were considered invalid to allow for more games to be run within time constraints.

Heuristic	**Win Rate**	**Average Turns**
Number of Moves	81%	131
Number of Squares	65%	432
Weighted Squares	79%	312

A large portion of games ended in a draw as they worked themselves into a looping state. This resulted in all of the heruistics converging on roughly 60% and the max of 500 rounds. Because of this these results were removed from the above values.

All three heuristics performed reasonably well, but the heuristic that counted the number of moves available was the best in both categories. This is probably due to it most closely representing reality in its evaluation.

4 Part 4

Here is some sample output from a game in which the rational player won. For brevity squares owned by the rational player are denoted A and B for the random player.

0: 0	0: 0	0: 0	0: 0
A: 1	0: 0	0: 0	0: 0
A: 2	0: 0	0: 0	0: 0
A: 7	0: 0	0: 0	B: 10

Rational players turn 1

0: 0	0: 0	0: 0	0: 0
A: 1	0: 0	0: 0	0: 0
A: 2	0: 0	0: 0	0: 0
A: 7	0: 0	0: 0	B: 10

Random players turn 1

B: 7	0: 0	0: 0	0: 0
A: 1	B: 2	0: 0	0: 0
A: 2	0: 0	B: 1	0: 0
A: 7	0: 0	0: 0	0: 0

Rational players turn 2

B: 7	0: 0	0: 0	A: 4
A: 1	B: 2	A: 2	0: 0
A: 2	A: 1	B: 1	0: 0
0: 0	0: 0	0: 0	0: 0

Random players turn 2

B: 9	0: 0	0: 0	A: 4
A: 1	0: 0	A: 2	0: 0
A: 2	A: 1	B: 1	0: 0
0: 0	0: 0	0: 0	0: 0

Rational players turn 3

B: 9	0: 0	A: 1	A: 4
A: 1	A: 1	A: 2	0: 0
0: 0	A: 1	B: 1	0: 0
0: 0	0: 0	0: 0	0: 0

This move really helped the rational player because it allowed them to corner 9 of the random players stones. By moving here the rational player greatly reduces the number of potential moves the random player could take while leaving themselves relatively free to move.

Random players turn 3

0: 0	B: 9	A: 1	A: 4
A: 1	A: 1	A: 2	0: 0
0: 0	A: 1	B: 1	0: 0
0: 0	0: 0	0: 0	0: 0

Rational players turn 4

0: 0	B: 9	A: 1	0: 0
A: 1	A: 1	A: 2	A: 1
0: 0	A: 1	B: 1	A: 2
0: 0	0: 0	0: 0	A: 1

Random players turn 4

0: 0	B: 9	A: 1	0: 0
A: 1	A: 1	A: 2	A: 1
0: 0	A: 1	0: 0	A: 2
0: 0	0: 0	B: 1	A: 1

Rational players turn 5

0: 0	B: 9	A: 1	0: 0
A: 1	A: 1	A: 2	0: 0
0: 0	A: 1	A: 1	A: 2
0: 0	0: 0	B: 1	A: 1

Random players turn 5

B: 9	0: 0	A: 1	0: 0
A: 1	A: 1	A: 2	0: 0
0: 0	A: 1	A: 1	A: 2
0: 0	0: 0	B: 1	A: 1

Rational players turn 6

B: 9	A: 1	0: 0	0: 0
A: 1	A: 1	A: 2	0: 0
0: 0	A: 1	A: 1	A: 2
0: 0	0: 0	B: 1	A: 1

Random players turn 6

B: 9	A: 1	0: 0	0: 0
A: 1	A: 1	A: 2	0: 0
0: 0	A: 1	A: 1	A: 2
0: 0	B: 1	0: 0	A: 1

Rational players turn 7

B: 9	A: 1	0: 0	A: 1
A: 1	A: 1	A: 2	A: 1
0: 0	A: 1	A: 1	0: 0
0: 0	B: 1	0: 0	A: 1

Random players turn 7

B: 9	A: 1	0: 0	A: 1
A: 1	A: 1	A: 2	A: 1
0: 0	A: 1	A: 1	0: 0
0: 0	0: 0	B: 1	A: 1

Rational players turn 8

B: 9	A: 1	A: 1	0: 0
A: 1	A: 1	A: 2	A: 1
0: 0	A: 1	A: 1	0: 0
0: 0	0: 0	B: 1	A: 1

Random players turn 8

B: 9	A: 1	A: 1	0: 0
A: 1	A: 1	A: 2	A: 1
0: 0	A: 1	A: 1	0: 0
0: 0	B: 1	0: 0	A: 1

Rational players turn 9

B: 9	A: 1	A: 1	0: 0
A: 1	A: 1	A: 2	A: 1
0: 0	A: 1	A: 1	0: 0
0: 0	B: 1	A: 1	0: 0

Random players turn 9

B: 9	A: 1	A: 1	0: 0
A: 1	A: 1	A: 2	A: 1
0: 0	A: 1	A: 1	0: 0
B: 1	0: 0	A: 1	0: 0

Rational players turn 10

B: 9	A: 1	A: 1	0: 0
A: 1	A: 1	A: 2	A: 1
0: 0	A: 1	0: 0	0: 0
B: 1	A: 1	A: 1	0: 0

Random players turn 10

B: 9	A: 1	A: 1	0: 0
A: 1	A: 1	A: 2	A: 1
B: 1	A: 1	0: 0	0: 0
0: 0	A: 1	A: 1	0: 0

Rational players turn 11

B: 9	A: 1	A: 1	0: 0
A: 1	A: 1	A: 2	A: 1
B: 1	A: 1	0: 0	A: 1
0: 0	A: 1	0: 0	0: 0

Random players turn 11

B: 9	A: 1	A: 1	0: 0
A: 1	A: 1	A: 2	A: 1
0: 0	A: 1	0: 0	A: 1
B: 1	A: 1	0: 0	0: 0

Rational players turn 12

B: 9	A: 1	A: 1	0: 0
A: 1	A: 1	0: 0	A: 1
0: 0	A: 1	A: 1	A: 1
B: 1	A: 1	A: 1	0: 0

Random players turn 12

B: 9	A: 1	A: 1	0: 0
A: 1	A: 1	0: 0	A: 1
B: 1	A: 1	A: 1	A: 1
0: 0	A: 1	A: 1	0: 0

Rational players turn 13

B: 9	A: 1	0: 0	0: 0
A: 1	A: 1	A: 1	A: 1
B: 1	A: 1	A: 1	A: 1
0: 0	A: 1	A: 1	0: 0

Random players turn 13

B: 9	A: 1	0: 0	0: 0
A: 1	A: 1	A: 1	A: 1
0: 0	A: 1	A: 1	A: 1
B: 1	A: 1	A: 1	0: 0

Rational players turn 14

B: 9	A: 1	0: 0	0: 0
0: 0	A: 1	A: 1	A: 1
A: 1	A: 1	A: 1	A: 1
B: 1	A: 1	A: 1	0: 0

This move does not seem like the most optimal move to make because it frees up 9 of the random player's stones for movement, but if you look at the other potential moves this is in a 4 way tie with other moves. There was no way to block off the stone in the bottom left corner without leaving room for either it or the set of 9 stones to move. The algorithm probably looked ahead and saw that it could win quickly using this move so it took it.

Random players turn 14

0: 0	A: 1	0: 0	0: 0
B: 9	A: 1	A: 1	A: 1
A: 1	A: 1	A: 1	A: 1
B: 1	A: 1	A: 1	0: 0

Rational players turn 15

0: 0	A: 1	A: 1	0: 0
B: 9	A: 1	0: 0	A: 1
A: 1	A: 1	A: 1	A: 1
B: 1	A: 1	A: 1	0: 0

This move looks fairly pointless since the rational player moved a stone currently doing nothing to help cage in the random player, but looking ahead you can see that the rational player is moving that stone closer to help fill in the gap there. Once again, this was probably discovered using the algorithm's look ahead.

Random players turn 15

B: 9	A: 1	A: 1	0: 0
0: 0	A: 1	0: 0	A: 1
A: 1	A: 1	A: 1	A: 1
B: 1	A: 1	A: 1	0: 0

Rational players turn 16

B: 9	0: 0	A: 1	0: 0
A: 1	A: 1	0: 0	A: 1
A: 1	A: 1	A: 1	A: 1
B: 1	A: 1	A: 1	0: 0

Random players turn 16

0: 0	B: 9	A: 1	0: 0
A: 1	A: 1	0: 0	A: 1
A: 1	A: 1	A: 1	A: 1
B: 1	A: 1	A: 1	0: 0

Rational players turn 17

0: 0	B: 9	A: 1	0: 0
A: 1	A: 1	A: 1	0: 0
A: 1	A: 1	A: 1	A: 1
B: 1	A: 1	A: 1	0: 0

Random players turn 17

B: 9	0: 0	A: 1	0: 0
A: 1	A: 1	A: 1	0: 0
A: 1	A: 1	A: 1	A: 1
B: 1	A: 1	A: 1	0: 0

Rational players turn 18

B: 9	A: 1	0: 0	0: 0
A: 1	A: 1	A: 1	0: 0
A: 1	A: 1	A: 1	A: 1
B: 1	A: 1	A: 1	0: 0

Rational player wins Rational player won 0.1 Random player won 0.0