Team:
Sangam Sahai
Deepti Bhatia

Squeeze-It

I. Design of the program:

Our program has the following modules that make our code flexible and modifiable:

1. GlobalParameters.java:
   To maintain the following global parameters of our game:
   a. myColor: The color of the pieces we are playing with
   b. otherColor: The color of the opponent's pieces
   c. number of rows and columns on the board
   d. level_of_look_ahead: depth of the search tree

2. CoOrdinate.java:
   To store and retrieve the position (x, y coordinate) of each piece on the board

3. BoardState.java:
   Includes the following functions to take actions on the board:
   a. checkSqueezePatternDown/Left/Middle/MiddleVertical/Right/Up:
      return YES if the squeeze pattern is formed when the piece moves to destination in that direction (down/left/middle/middlevertical/right/up). Calls getDown/Left/Right/Up function to check this.
   b. checkSqueezePattern: Calls above functions by moving the piece one block at a time in all possible directions and returns true if any kind of pattern is formed
   c. getHeuristicScoreForThisConfiguration: Calculates heuristic for the input board configuration and returns it
   d. getDown/Left/Right/Up: returns piece coordinate when the piece moves from the given source to one block in left/right/up/down direction
   e. getOutputStateForGivenMove: takes the piece, source and destination coordinates as input and returns board configuration with score (actual+heuristic) for the move from source to destination
   f. printConfig: prints the board configuration

4. TreeNode.java:
   Defines the structure of a node of the search tree

5. MiniMax.java:
   Includes the following functions:
   a. populateTree: generates state space search tree with the initial board configuration and desired number of levels
   b. getTheMiniMaxScore: returns the score of the given root considering it's children and whether it is a max or min node

6. TestClass.java:
   A temporary class to test the working of our game

II. Heuristic Function:

We calculate our heuristic in the following way:

For a given board configuration:

a. For every move that we can make which completes the squeeze pattern, we add a 10 to the heuristic value
b. For every move that our opponent can make to complete the squeeze pattern and eliminate us, we subtract a 10 from the heuristic value

For example, consider that we are black pieces and the opponent is playing white pieces.

For the given board configuration:

<table>
<tr>
<td>
<table>
<tr><td>w</td><td></td><td>w</td><td>w</td></tr>
<tr><td></td><td>b</td><td>w</td><td></td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td>b</td><td></td><td>b</td><td>b</td></tr>
</table>
</td>
<td>Heuristic value: 10
Here, we can move from (1,1) to (0,1) and complete one squeeze pattern. But our opponent cannot make any pattern with a single move. So the total heuristic value for this configuration is 10.</td>
</tr>
<tr>
<td>
<table>
<tr><td>w</td><td>w</td><td></td><td></td></tr>
<tr><td></td><td>b</td><td>w</td><td>w</td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td>b</td><td></td><td>b</td><td>b</td></tr>
</table>
</td>
<td>Heuristic value: -10
Here, white piece from (0,0) can move to (1,0) and complete the squeeze pattern to eliminate us. But we cannot make a move to make a squeeze pattern. So the total heuristic value for this configuration is -10 (i.e. in favor of the opponent)</td>
</tr>
<tr>
<td>
<table>
<tr><td>w</td><td>w</td><td></td><td>b</td></tr>
<tr><td></td><td>b</td><td>w</td><td></td></tr>
<tr><td></td><td></td><td></td><td>w</td></tr>
<tr><td>b</td><td></td><td>b</td><td></td></tr>
</table>
</td>
<td>Heuristic value: -10
Here, white piece from (0,0) can move to (1,0) and complete one squeeze pattern. Also, the white piece from (2,3) can move to (2,1) to complete another squeeze pattern. So the score for white moves is -20. Black piece from (0,3) can move to (1,3) to complete one pattern i.e. score +10. Finally, total heuristic for this configuration is -20+10=-10</td>
</tr>
</table>

III. Features that make our program smart:

- While generating the search tree, if we generate a board configuration which forms a squeeze pattern with our move, we assign that configuration a score (called ‘actual score’) of +1000 and if it forms a squeeze pattern with the opponent’s move, we assign it a score of -1000.
- We do this to avoid searching the tree any further because we know that it is the best move to make (if it is +1000).

For example, considering we are black pieces:

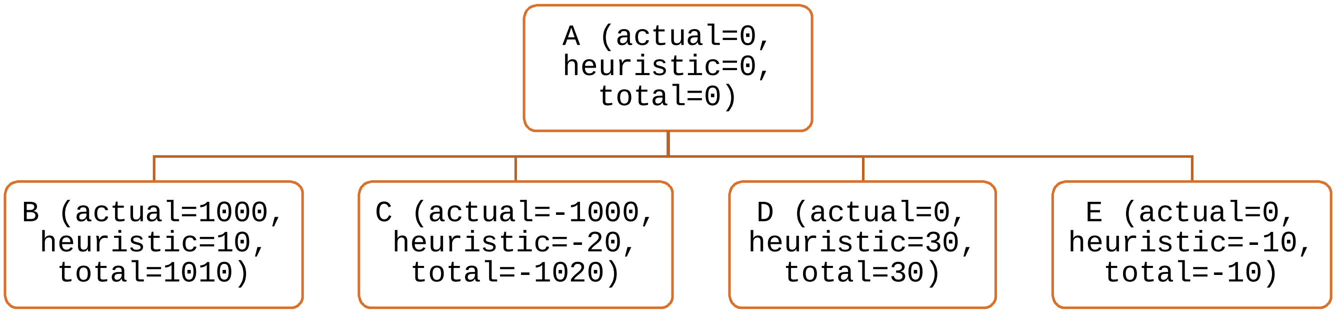
w	w		b
	b	w	
			w
b		b	

From (0.3) to (1.3) →

w	w		
	b	w	b
			w
b		b	

- In this case, we give board configuration on the left an actual heuristic score of 1000 because our move (black move) creates a squeeze pattern. Thus we do not need to generate other possible children for this board configuration in the search tree.
- Once we have generated our state space tree, we do a Minimax search through the tree considering the total score (actual score + heuristic score)

If A is a Max node, the heuristic value of A will be chosen to be among total value of B (1010), total value of C (-1020), total value of D (30) and total value of E (-10). Thus we fix the heuristic value of A to be 1010. Which intuitively means that we have a guarantee of getting a squeeze pattern and eliminating the opponent if we choose a move that leads to configuration B.


A (actual=0, heuristic=0, total=0)
B (actual=1000, heuristic=10, total=1010)
C (actual=-1000, heuristic=-20, total=-1020)
D (actual=0, heuristic=30, total=30)
E (actual=0, heuristic=-10, total=-10)

Whereas if A is a Min node, we choose heuristic of A among total value of B (1010), total value of C (-1020), total value of D (30) and total value of E (-10). Thus, heuristic value of A is chosen as -1020. Which intuitively means that a move to board configuration C is a move where the minimizing player (opponent) wins.