Summary of all the functions:

Our implementation is IDS for MINIMAX algorithm with alpha beta pruning. However we know that it is not possible to search the entire search space and hence we use the heuristics.

Successors(board, player):
This is the function to create successors of given board. Player attribute denotes the player who's board we have passed as input to this function. This implies the successors of this board will be the states for opponent. In short, we have created the possible states for opponent of the player. There will be at max 2*n successors (one for drop and one for rotate for each column). There are 3 conditions which are specially handled in this function. If the column is full then we cannot drop more pebble to this column. If the column is empty, then we cannot rotate the column. And if the number of pebbles on the board for opponent of the input player is equal to n*(n-3)/2 then we cannot drop any more pebble i.e., no successors for drop action.

isGoal(board, player):
This function will check that the input board is goal for input player or not. To check the goal state we are not considering bottom 3 rows of the board i.e., we are considering only top n rows of the board.
There are 3 rules for being any state a goal state (check in only top n rows of board):

1. If there are total n pebbles of input player in any column, then return true,
2. If there are total n pebbles of input player in any row, then return true,
3. If there are total n pebbles of input player in any diagonal, then return true,
4. Otherwise, return false.

isLeafNode(board):
This function is checking for the leaf node of the tree. We are implementing IDS algorithm to perform search operation. So, there is always a cut-off depth for search. So, there are 3 conditions for being leaf node for input board state:

1. The input board state can be the goal state for MAX player
2. The input board state can be the goal state for opponent player.
3. The depth the input board state is equal to cut-off depth

If any of these conditions satisfies that means the input board state is the leaf node.

MAXValue(board, alpha, beta, player) and MINValue(board, alpha, beta, player):
These are main functions which implements the minimax algorithm. MAX nodes are the nodes created after action performed by opponent player, and MIN nodes are the noes created after action performed by MAX player. The value associated with MAX nodes is alpha and value associated with the MIN nodes is beta. The basic thumb rule of the minimax algorithm is MIN node tries to minimize the alpha values coming from immediate next level MAX nodes and MAX nodes tries to maximize the beta values coming from the immediate next level MIN nodes.

MAXValue is the function which implements the action of maximization of beta values coming from immediate next level MIN nodes. For this, it calls MINValue function to get values of beta.
MINValues is the function which implements the action of minimization of alpha values coming from immediate next level MAX nodes. For this, it calls MAXValue function to get values of beta.
These two function will call recursive to the another function till it reaches to the cut-off depth of the tree.

AlphaBetaDecision(board):
This is the function which initiates the process minimax algorithm with alpha beta pruning. This function collects the beta values from the immediate next MIN node and will select a move with maximum beta value and returns this as the final decision.

Here are the following heuristics written, tried and tested by us:
For all the heuristic functions the positive value means that the state is favourable for the MAX player and the negative value means that the state is favourable for the MIN player.

Available Positions: The Available_pos function in the code represents this heuristic.
It gives the total number of available rows, columns and diagonals for MAX player minus the the total number of available rows, columns and diagonals for MIN player

Check Rows : checkInRows is the function which represents this.
It counts the number of rows for MAX player where the MAX pebbles where greater than (N/2) in a row.
The function returns the difference of this count for MAX and MIN players.

Check Columns : checkInColumns is the function which represents this.
It counts the number of rows for MAX player where the MAX pebbles where greater than (N+3/2) in a column.
The function returns the difference of this count for MAX and MIN players.

Count Consecutive number of pebbles in column: countConsecutiveColumnElements is the function which takes care of this implementation.
Here we are calculating the number of consecutive pebbles of a player in a given column.
We are doing this for all the columns and are returning the maximum value of the count.
We are calculating this count for MAX player and MIN player and are returning this value.
We are propagating a a very high positive or negative value when we encounter a consecutive count to be greater than or equal to N suggesting that if this state is reached then there's no way to stop the player from winning the game as far as this column is concerned.
We are playing defensive by propagating a negative value in case of the tie(i.e the count for MAX and MIN are same).

2(MaxCount) - 3(MinCount)

Distance to Complete Row: distancetoCompleteRow takes care of this
It calculates the number of columns to be rotated so as to form a row. This is done for all the rows for a given player and then the the min value of these values is returned for a given player.

The smaller the count, the closer is the goal state and hence the value should be greater.
We are calculating this value for MAX and MIN players as :
Count_MIN - Count_MAX
And hence the difference will encode the values propagated from the leaf values.

Distance to Goal Column: distancetoGoalColumn
This function does the same thing as the distance to Complete row does except for this time we are checking for the columns.
We are looking for the consecutive chunk of the pebbles >= N for a given player. Now we are looking for the distance of this chunk to its goal column.

Leaf Value: leafValue is the heart of the program.
Here we are propagating the values from the leaf nodes.
We have taken all the heuristics in consideration before coming up with some combination of the heuristics with their weights.
For the winning and losing condition, we are dividing the value with the current depth of the tree so as to promote or demote the closer goal states.

All the final heuristic values are between the minimum value(i.e. The closest losing state for MAX) and the maximum value(i.e the closest winning state for MAX)

The best heuristics for us where the distancetoGoalColumn and distancetoCompleteRow which were written with a lot of thought in it.
However we did not get enough time to experiment the weights of these heuristic to come up with the best possible combination.
Given the time, we have a feeling that a good combination of just these heuristics will make up a very good heuristic than it is for now.
We can say this because we have checked the combinatory values of all the heuristics and those are under the minimum and maximum value window.