

Read Me

Problem Statement: -

Aim: - To create a Solitaire Chess puzzle and use search algorithm for find the goal state in Solitaire Chess.

Algorithm Used for finding the Solution of the Solitaire Chess Puzzle are: -

Uninformed Search: -

1. BFS Search (Breadth First Search).
2. DFS Search (Depth First Search).
3. Depth – Limited Search.
4. Iterative – Deepening Search.

Informed Search: -

1. A* Search.
2. Iterative Deepening A* Search.

Solitaire Chess

Solitaire Chess is logic puzzle that uses the rule of Chess.

Characteristic of Solitaire Chess

1. Chess Board is 4*4 in dimensions.
2. Only 6 pieces from original chess can be used (King, Bishop, Rook, Queen, Pawn, Knight).
3. Allowed moves for each of the piece is same as their moves in original chess.
4. For move to be “legal” in solitaire chess, it is compulsory to kill a piece during each move.
5. Final State of Solitaire Chess is the board with just one piece remaining on it.

Heuristics Used for Solving the Puzzle.

1. All the above-mentioned rules must be followed for creating and solving the puzzle.
2. I have considered the following notation for the Puzzle

2.1 The Blank Space in puzzle will be represent by “.”

2.2 For representing the Pieces on Puzzle I had used the following symbol's

K for King

Q for Queen

N for Knight

B for Bishop

R for Rook

P for Pawn

3. At any stage the Puzzle will be having the representation as: -

```

=====
=====
K      .      N      .
.      .      .      .
.      Q      B      .
.      .      .      .
=====
=====

```

4. Heuristics for Informed Search Techniques.

- 4.1 For the search to be an Informed search (like A* search) it is necessary to give cost to each move and based on this cost, the movement of the Piece on the board is decided (i.e., which Piece to be move next from the possible allowed moves).
- 4.2 In these cases, apart for considering the legal moves of the piece the cost of moving the piece also plays an important role.

Problem in deciding Heuristics: -

1. Since in Solitaire Chess it doesn't matter which Piece kill the other Piece until and unless the move taken by the piece is a legal move and it leads us to a goal state.
2. Thus, as there is no partiality to any piece on the board, it is very difficult to select the priority order of the pieces.

Possible approach used for creating the solution: -

1. As the Priority of the piece cannot be predicted based on their moves, an alternative approach for prioritizing the piece could be based on the area covered by the piece.
2. For Instance: Queen can cover the entire board in 8 different direction whereas Rook and Bishop can cover the entire board in 4 different direction respectively. Thus, as we want to reach to the solution state quickly the heuristic value for Queen is very low compared to the heuristic value of Bishop and Rook as the probability of killing a piece from Queen is double than both Bishop and Rook.
3. Heuristic Values for A* and Iterative A* search based on the above assumptions will be: --

Piece	H(Piece)
Queen	1
Rook	3
Bishop	3
Knight	4
King	5
Pawn	6

4. As lower the heuristic value of a piece the higher the chances of getting the solution by expanding the piece.

Algorithms: -

DFS (Depth – First Search)

v = Pieces in the Solitaire Chess Board

G = Chessboard current configuration

1. procedure DFS-iterative (G, v):
2. let S be a stack
3. S.push(v)
4. while S is not empty
5. v = S.pop()
6. if v is the goal:
7. return v
8. if v is not labeled as discovered:
9. label v as discovered
10. for all legal moves from v to w in G.adjacentEdges(v) do
11. S.push(w)

BFS (Breadth First Search)

1. Breadth-First-Search(Graph, root):
2. create empty set S
3. create empty queue Q
- 4.
5. root.parent = NIL
6. Q.enqueue(root)
- 7.
8. while Q is not empty:
9. current = Q.dequeue()
10. if current is the goal:
11. return current
12. for each piece n that is killed by current:
13. if n is not in S:
14. add n to S
15. n.parent = current
16. Q.enqueue(n)

Depth Limit Search

v = Pieces in the Solitaire Chess Board

G = Chessboard current configuration

1. procedure DLS-iterative (G, v, depth):
2. let S be a stack
3. S.push(v)
4. while S is not empty
5. N = depth of v to root node;
6. If N > depth
7. Return "No possible solution".
8. else
9. v = S.pop()
10. if v is the goal:
11. return v
12. if v is not labeled as discovered:
13. label v as discovered
14. for all legal moves from v to w in G.adjacentEdges(v) do
15. S.push(w)

Iterative Deepening Search

v = Pieces in the Solitaire Chess Board

G = Chessboard current configuration

1. procedure IDDS-iterative (G, v, depth):
2. let S be a stack and Q be queue
3. S.push(v)
4. While(!stack.empty())
5. Current = S.pop()
6. Q.add(Current)
7. If Current is goalState
8. Return current.
9. If stack.size == 0
10. For QTop in Q
11. If(!visited(neighbor,QTop))
12. Visited.put(neighbor,QTop)
13. S.push(neighbor)
14. Q.removeAll(Q)

A* Search

v = Pieces in the Solitaire Chess Board

G = Chessboard current configuration

1. procedure Astar (G, v):
2. let Q be a queue
3. S.push(v)
4. // create a heuristic for the search.
5. If v contains "Queen"
6. Decider = Queen;
7. elif v contains "Rook"
8. Decider = Rook;
9. Elif v contain "Bishop"
10. Decider = Bishop;
11. Elif v contain "knight"
12. Decider = Knight;
13. Elif v contain "King"
14. Decider = King;
15. Else
16. Decider = Pawn;
17. Visited.put(v,null);
18. While(!Q.isEmpty() && Q1.contain(Decider))
19. Current = q.get(0)
20. Q.remove(0)
21. If Current is goal
22. Return goal
17. for each piece n that is killed by current:
18. if n is not in visited:
19. Visit n;
20. Q.add(n);
23. Manage Heuristic for n.

IDA* Search

v = Pieces in the Solitaire Chess Board

G = Chessboard current configuration

1. procedure IDA (G, v):
2. let Q be a queue
3. S.push(v)
4. // create a heuristic for the search.
5. If v contains "Queen"

```

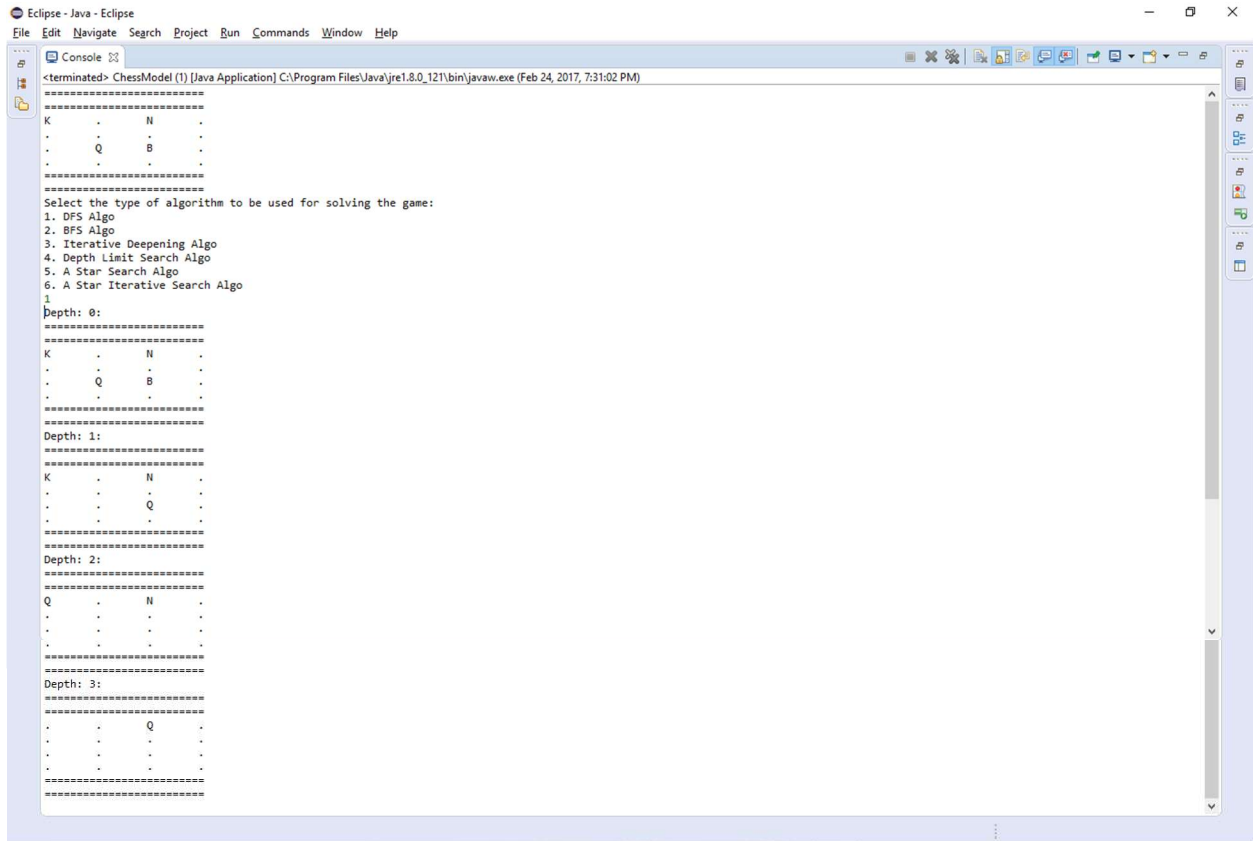
6.         Decider = Queen;
7.     elif v contains "Rook"
8.         Decider = Rook;
9.     Elif v contain "Bishop"
10.        Decider = Bishop;
11.    Elif v contain "knight"
12.        Decider = Knight;
13.    Elif v contain "King"
14.        Decider = King;
15.    Else
16.        Decider = Pawn;
17.    Q.add(Current)
18.    While( !Q.isEmpty() && Q1.contains(Decider))
19.        Current = q.get(0)
20.        Q.remove(0)
21.        If Current is goal
22.            Return goal
23.        If stack.size == 0
24.            For QTop in Q
25.                If(!visited(neighbor,QTop))
26.                    Visited.put(neighbor,QTop)
27.                    Find Heuristic Decider for neighbor
28.                    S.push(neighbor)
29.        Q.removeAll(Q)

```

Testing Results.

Output Snippets

1. DFS



The screenshot shows the Eclipse IDE interface with the console window open. The console output displays the execution of a Java application named 'ChessModel (1)'. The output includes a chessboard state, a list of search algorithms, and the selection of the DFS algorithm. It then shows the search process at depths 0, 1, 2, and 3, with corresponding chessboard states.

```
<terminated> ChessModel (1) [Java Application] C:\Program Files\Java\jre1.8.0_121\bin\javaw.exe (Feb 24, 2017, 7:31:02 PM)
=====
K      .      N      .
.      .      .      .
.      Q      B      .
.      .      .      .
=====
Select the type of algorithm to be used for solving the game:
1. DFS Algo
2. BFS Algo
3. Iterative Deepening Algo
4. Depth Limit Search Algo
5. A Star Search Algo
6. A Star Iterative Search Algo
1
Depth: 0:
=====
K      .      N      .
.      .      .      .
.      Q      B      .
.      .      .      .
=====
Depth: 1:
=====
K      .      N      .
.      .      .      .
.      .      Q      .
.      .      .      .
=====
Depth: 2:
=====
Q      .      N      .
.      .      .      .
.      .      .      .
.      .      .      .
=====
Depth: 3:
=====
.      .      Q      .
.      .      .      .
.      .      .      .
.      .      .      .
=====
```

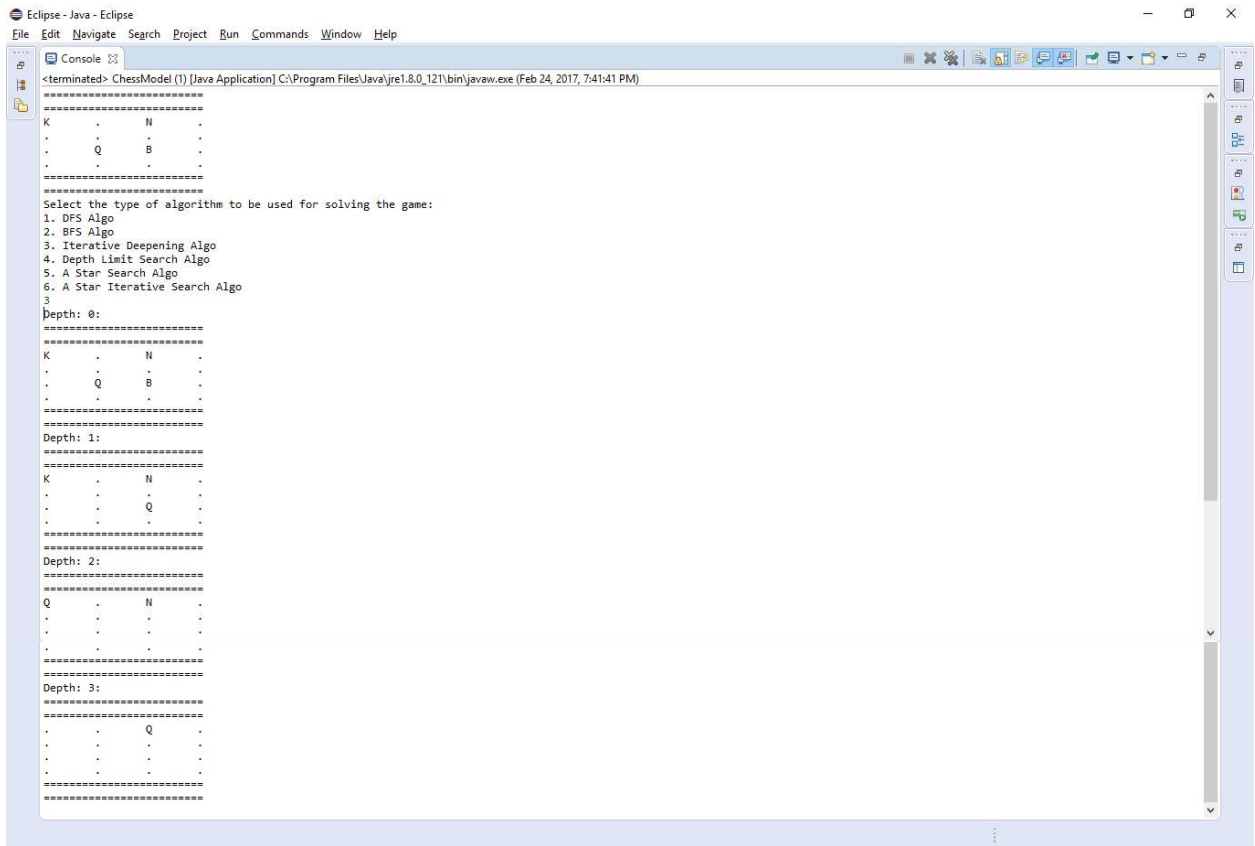
2. BFS

```
Eclipse - Java - Eclipse
File Edit Navigate Search Project Run Commands Window Help

Console
<terminated> ChessModel (1) [Java Application] C:\Program Files\Java\jre1.8.0_121\bin\javaw.exe (Feb 24, 2017, 7:36:27 PM)

=====
K      .      N      .
.      .      .      .
.      Q      B      .
.      .      .      .
=====
Select the type of algorithm to be used for solving the game:
1. DFS Algo
2. BFS Algo
3. Iterative Deepening Algo
4. Depth Limit Search Algo
5. A Star Search Algo
6. A Star Iterative Search Algo
2
Depth: 0:
=====
K      .      N      .
.      .      .      .
.      Q      B      .
.      .      .      .
=====
Depth: 1:
=====
K      .      .      .
.      .      .      .
.      N      B      .
.      .      .      .
=====
Depth: 2:
=====
N      .      .      .
.      .      .      .
.      .      B      .
.      .      .      .
=====
Depth: 3:
=====
B      .      .      .
.      .      .      .
.      .      .      .
.      .      .      .
=====
```


3. Iterative Deepening Search

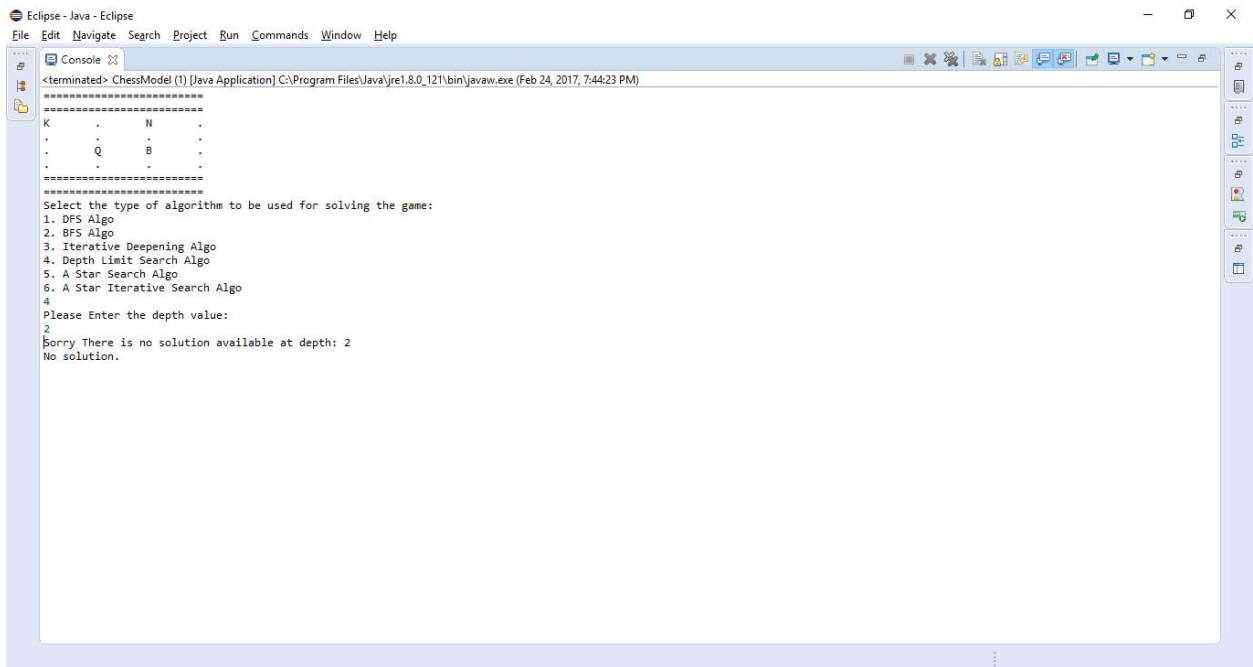


```
Eclipse - Java - Eclipse
File Edit Navigate Search Project Run Commands Window Help

Console
<terminated> ChessModel (1) [Java Application] C:\Program Files\Java\jre1.8.0_121\bin\javaw.exe (Feb 24, 2017, 7:41:41 PM)

=====
K      .      N      .
.      .      .      .
.      Q      B      .
.      .      .      .
=====
Select the type of algorithm to be used for solving the game:
1. DFS Algo
2. BFS Algo
3. Iterative Deepening Algo
4. Depth Limit Search Algo
5. A Star Search Algo
6. A Star Iterative Search Algo
3
Depth: 0:
=====
K      .      N      .
.      .      .      .
.      Q      B      .
.      .      .      .
=====
Depth: 1:
=====
K      .      N      .
.      .      .      .
.      .      Q      .
.      .      .      .
=====
Depth: 2:
=====
Q      .      N      .
.      .      .      .
.      .      .      .
.      .      .      .
=====
Depth: 3:
=====
.      .      Q      .
.      .      .      .
.      .      .      .
.      .      .      .
=====
```

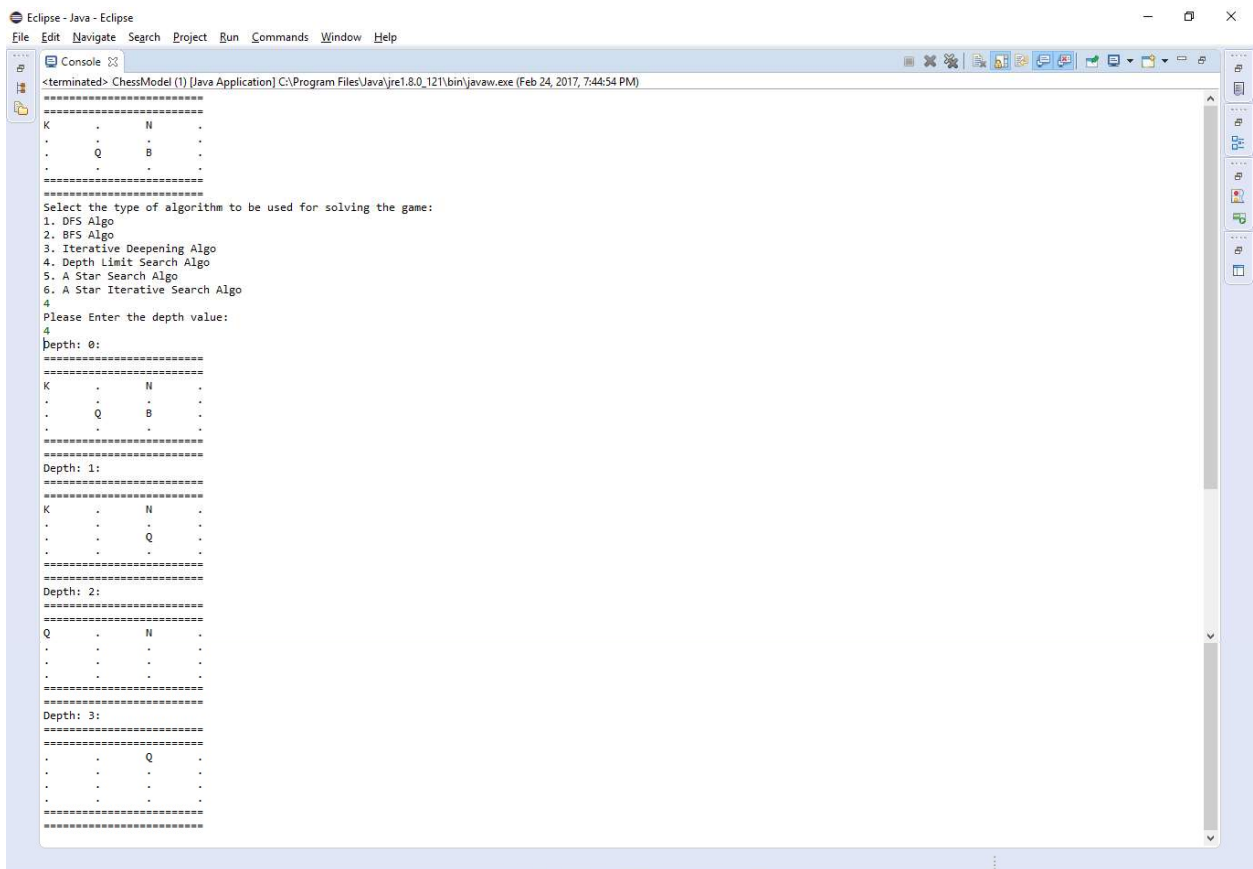
4. Depth Limit Search



```
Eclipse - Java - Eclipse
File Edit Navigate Search Project Run Commands Window Help

Console
<terminated> ChessModel (1) [Java Application] C:\Program Files\Java\jre1.8.0_121\bin\javaw.exe (Feb 24, 2017, 7:44:23 PM)

=====
K      .      N      .
.      .      .      .
.      Q      B      .
.      .      .      .
=====
Select the type of algorithm to be used for solving the game:
1. DFS Algo
2. BFS Algo
3. Iterative Deepening Algo
4. Depth Limit Search Algo
5. A Star Search Algo
6. A Star Iterative Search Algo
4
Please Enter the depth value:
2
Sorry There is no solution available at depth: 2
No solution.
```

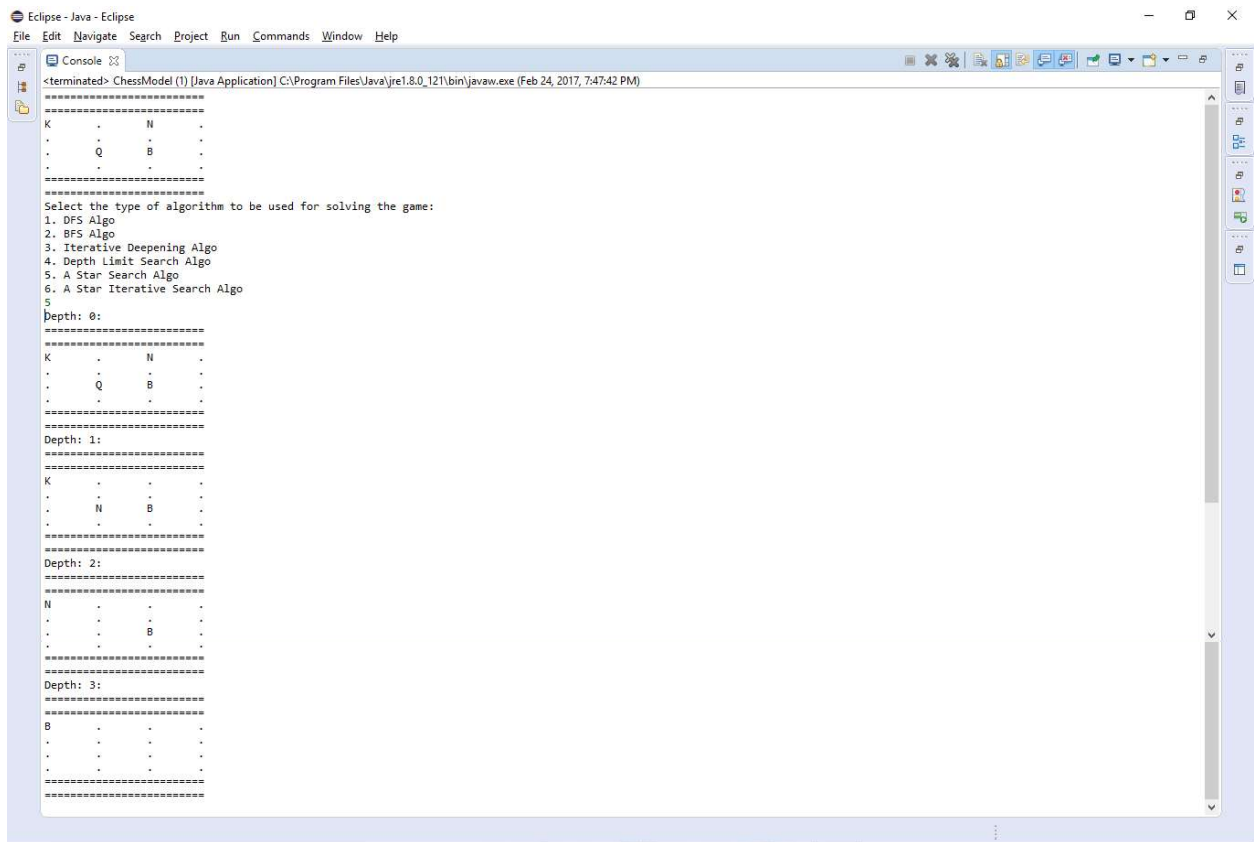


```
Eclipse - Java - Eclipse
File Edit Navigate Search Project Run Commands Window Help

Console
<terminated> ChessModel (1) [Java Application] C:\Program Files\Java\jre1.8.0_121\bin\javaw.exe (Feb 24, 2017, 7:44:54 PM)

=====
K      .      N      .
.      .      .      .
.      Q      B      .
.      .      .      .
=====
Select the type of algorithm to be used for solving the game:
1. DFS Algo
2. BFS Algo
3. Iterative Deepening Algo
4. Depth Limit Search Algo
5. A Star Search Algo
6. A Star Iterative Search Algo
4
Please Enter the depth value:
4
Depth: 0:
=====
K      .      N      .
.      .      .      .
.      Q      B      .
.      .      .      .
=====
Depth: 1:
=====
K      .      N      .
.      .      .      .
.      .      Q      .
.      .      .      .
=====
Depth: 2:
=====
Q      .      N      .
.      .      .      .
.      .      .      .
.      .      .      .
=====
Depth: 3:
=====
.      .      Q      .
.      .      .      .
.      .      .      .
.      .      .      .
=====
```

5. A* Search



The screenshot shows the Eclipse IDE interface. The main window displays a Java application titled "ChessModel (1) [Java Application] C:\Program Files\Java\jre1.8.0_121\bin\javaw.exe (Feb 24, 2017, 7:47:42 PM)". The console output shows a chess game simulation. It starts with a board state, followed by a prompt to select an algorithm. The user selects "A Star Search Algo". The simulation then proceeds through several depths, showing the board state at each step. The board is represented as a 3x3 grid with letters K, N, Q, B, and empty spaces.

```
<terminated> ChessModel (1) [Java Application] C:\Program Files\Java\jre1.8.0_121\bin\javaw.exe (Feb 24, 2017, 7:47:42 PM)
=====
K   .   N   .
.   .   .   .
.   Q   B   .
.   .   .   .
=====
Select the type of algorithm to be used for solving the game:
1. DFS Algo
2. BFS Algo
3. Iterative Deepening Algo
4. Depth Limit Search Algo
5. A Star Search Algo
6. A Star Iterative Search Algo
5
Depth: 0:
=====
K   .   N   .
.   .   .   .
.   Q   B   .
.   .   .   .
=====
Depth: 1:
=====
K   .   .   .
.   .   .   .
.   N   B   .
.   .   .   .
=====
Depth: 2:
=====
N   .   .   .
.   .   .   .
.   .   B   .
.   .   .   .
=====
Depth: 3:
=====
B   .   .   .
.   .   .   .
.   .   .   .
.   .   .   .
=====
```

6. Iterative A* Algorithm

```
Eclipse - Java - Eclipse
File Edit Navigate Search Project Run Commands Window Help

Console
<terminated> ChessModel (1) [Java Application] C:\Program Files\Java\jre1.8.0_121\bin\javaw.exe (Feb 24, 2017, 7:49:48 PM)

=====
K      .      N      .
.      .      .      .
.      Q      B      .
.      .      .      .
=====
Select the type of algorithm to be used for solving the game:
1. DFS Algo
2. BFS Algo
3. Iterative Deepening Algo
4. Depth Limit Search Algo
5. A Star Search Algo
6. A Star Iterative Search Algo
6
Depth: 0:
=====
K      .      N      .
.      .      .      .
.      Q      B      .
.      .      .      .
=====
Depth: 1:
=====
K      .      N      .
.      .      .      .
.      .      Q      .
.      .      .      .
=====
Depth: 2:
=====
Q      .      N      .
.      .      .      .
.      .      .      .
.      .      .      .
=====
Depth: 3:
=====
.      .      Q      .
.      .      .      .
.      .      .      .
.      .      .      .
=====
```