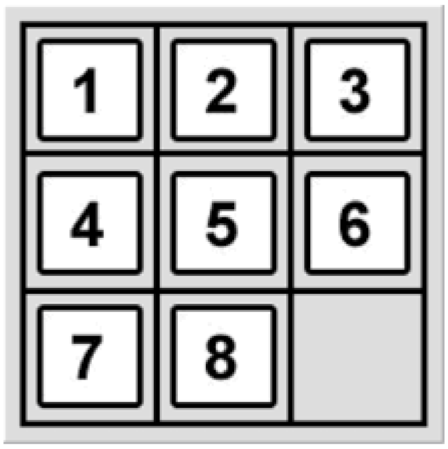
11/30/2023

Puzzle game application

Artificial intelligent.



By: Omar Alnayme
(omar.alnayme@outlook.com))

Table of Contents

Introduction:

This document explains my approach of developing application that give the player an option of playing 4 types of puzzle game (3,8,15,24), also provide auto solution solver using one of the three type of graph search techniques:

- Depth first search (DFS).
- Breadth first search (BFS).
- Heuristic search.

I have developed the application using:

Development tool	Visual studio 2015
Language used	C# . Net
Application type	Classic windows application
Data storage type	Local memory + text file

In this document, I will cover some examples one how the application process the search on each of the three-type search techniques, also I will explain a bit on the classes, and functions of each class, and the variable that they accept plus the function of each method within each class.

1 Graph Search techniques:

In this section I will explain how the application perform the graph search for each of the three types.

1.1 Depth first search:

To develop this type of search I used Linked list for each of the **open** and **closed** nodes.

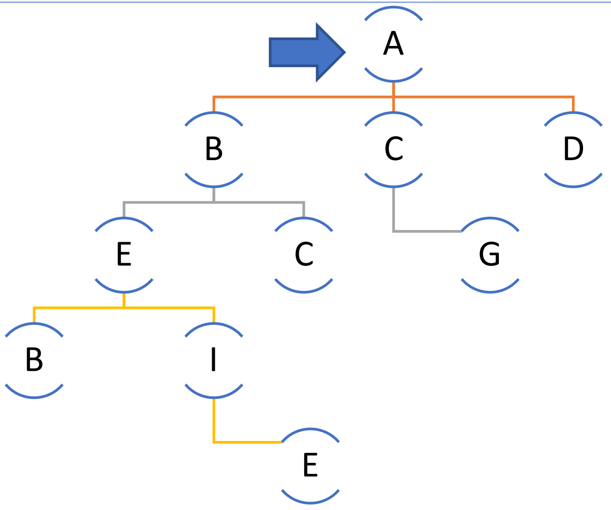
Open list will hold all the nodes that have not been processed (No children have been driven from yet) the **close list** will hold all the nodes that have been processed (children nodes have been driven from).

All new nodes will be added to the **beginning** of the **open list**.

Now I will explain how it works, let's assume that the start state is **A**, and the goal state is **G**, we will use the **DFS** to find our goal. Also, we will assume that we will search with depth limitation of **6**.

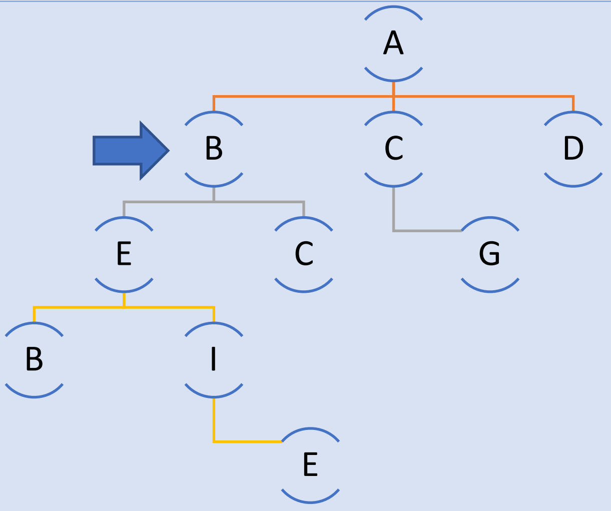
openlist=[A]
closedlist=[]

Check if A is the goal? No.
Check if the depth =6? No.
Check if A is already in Closed list? No.
Then get A children which is B,C,D.
Check if any of the children is the goal? No.
Check if any of A children is already in the open list? No.
Then add the children to the beginning of the open list. Remove A from open list and add A to the closed list.
openlist=[B,C,D]
closedlist=[A]



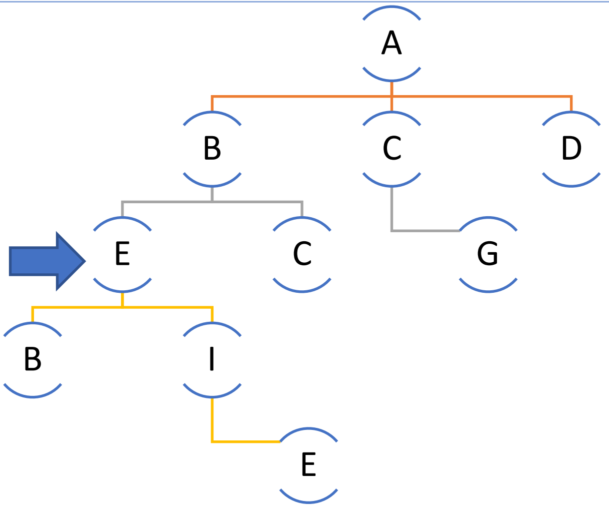
openlist=[B,C,D]
closedlist=[A]

Now we will select the first item in the open list.
Check if B is the goal? No.
Check if the depth =6? No.
Check if B is already in Closed list? No.
Then get B children which is E,C.
Check if any of the children is the goal? No.
Check if any of B children is in the open list? Yes.
Then add only the children that are not in the Open list to the beginning of the open list and remove B from open list and add B to the closed list.
openlist=[E,C,D]
closedlist=[B,A]



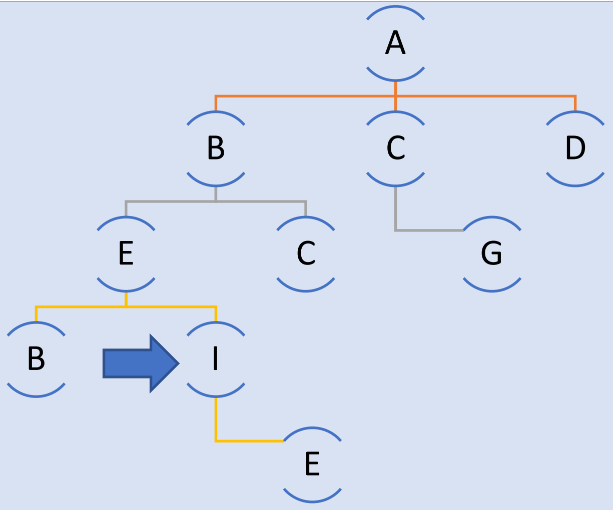
openlist=[E,C,D]
closedlist=[B,A]

Check if E is the goal? No.
Check if the depth =6? No.
Check if E is already in Closed list? No.
Then get E children which is B,I.
Check if any of the children is the goal? No.
Check if any of E children is in the open list? Yes.
Then add only the children that are not in the Open list to the beginning of the open list and remove E from open list and add E to the closed list.
openlist=[I,C,D]
closedlist=[E,B,A]



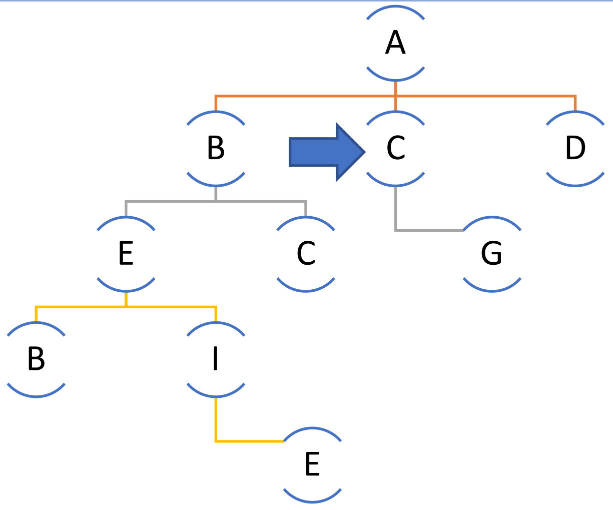
openlist=[I,C,D]
closedlist=[E,B,A]

Check if I is the goal? No.
Check if the depth =6? No.
Check if I is already in Closed list? No.
Then get I children which is E.
Check if any of the children is the goal? No.
Check if any of I children is in the open list? Yes.
Then add only the children that are not in the Open list to the beginning of the open list and remove I from open list and add I to the closed list.
openlist=[C,D]
closedlist=[I,E,B,A]



openlist=[C,D]
closedlist=[I,E,B,A]

Check if C is the goal? No.
Check if the depth =6? No.
Check if C is already in Closed list? No.
Then get C children which is G.
Check if any of the children is the goal? Yes.
Goal is found. Terminate process.



1.2 Breadth first search (BFS):

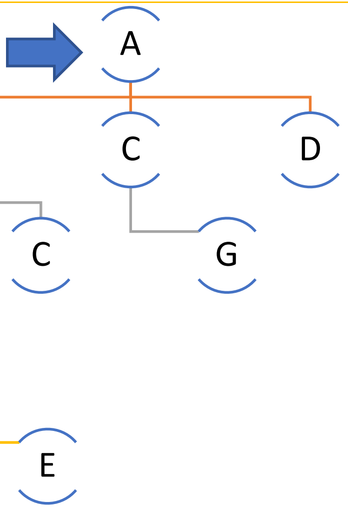
Open list will hold all the nodes that have not been processed (No children have been driven from yet) the **close list** will hold all the nodes that have been processed (children nodes have been driven from).

All new nodes will be added to the **end** of the **open list**, as for the DFS we added the new nodes to the beginning of the open list.

Now I will explain how it works, let's assume that the start state is **A**, and the goal state is **G**, we will use the **BFS** to find our goal.

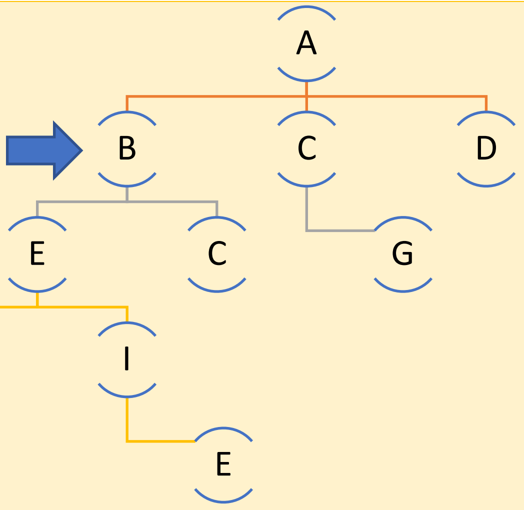
openlist=[A]
closedlist=[]

Check if A is the goal? No.
Check if A is already in Closed list? No.
Then get A children which is B,C,D.
Check if any of B,C,D is the goal? No.
Check if any of A children is already in the open list? No.
Then add the children to the end of the open list.
Remove A from open list and add A to the closed list.
openlist=[B,C,D]
closedlist=[A]



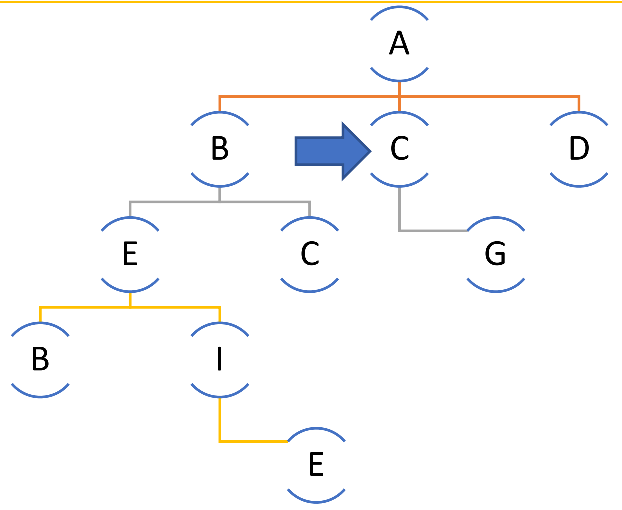
openlist=[B,C,D]
closedlist=[A]

Now we will select the first item in the open list.
Check if B is the goal? No.
Check if B is already in Closed list? No.
Then get B children which is E,C.
Check if E or C are the goal? No.
Check if any of B children is in the open list? Yes.
Then add only the children that are not in the Open list to the end of the open list and remove B from open list and add B to the closed list.
openlist=[C,D,E]
closedlist=[B,A]



openlist=[C,D,E]
closedlist=[B,A]

Check if C is the goal? No.
Check if C is already in Closed list? No.
Then get C children which is G.
Check if G is the goal? Yes.
Then goal is found. Terminate process.



1.3 Heuristic search:

The following evaluation equation is used:

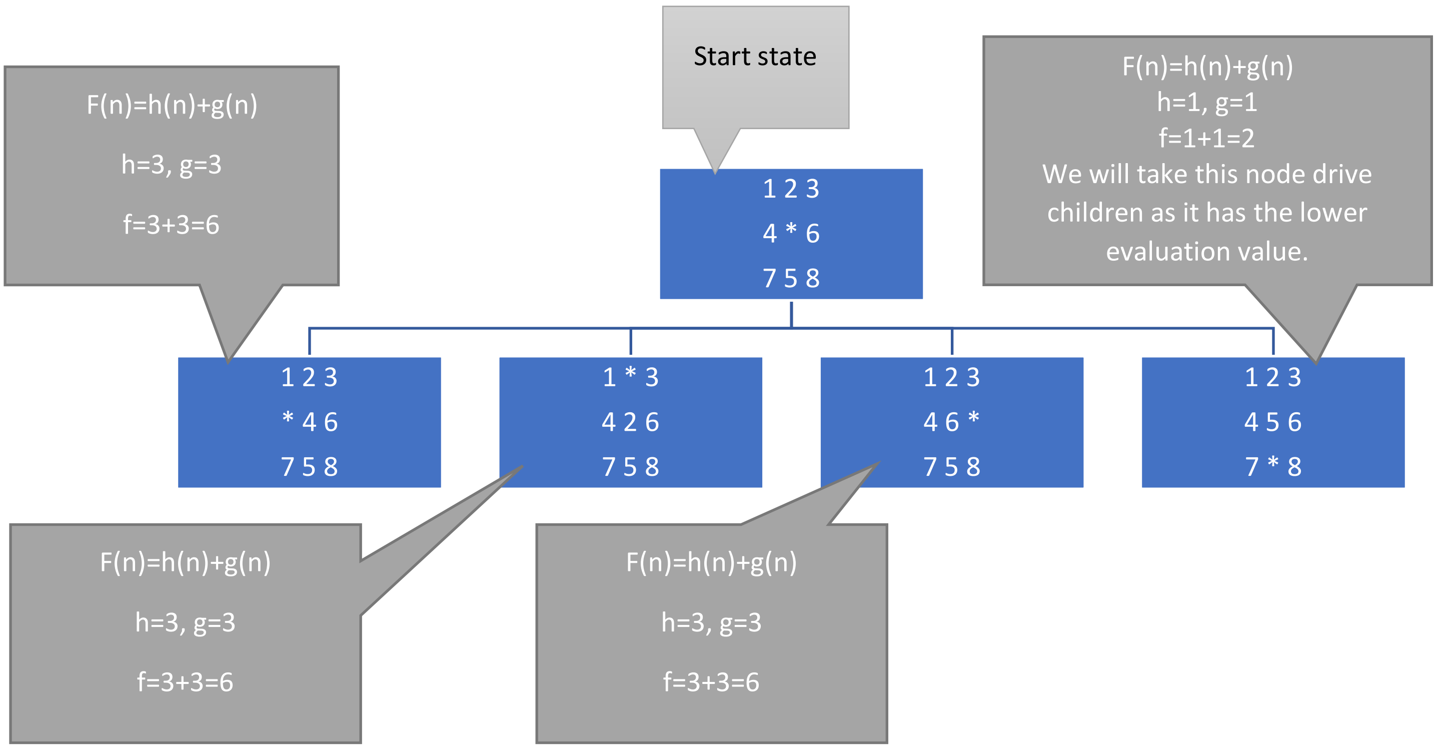
F(n) = h(n) + g(n).

F: is the evaluation value.
h= is number of tiles not in place.
g= number of moves for the tile to reach its goal.

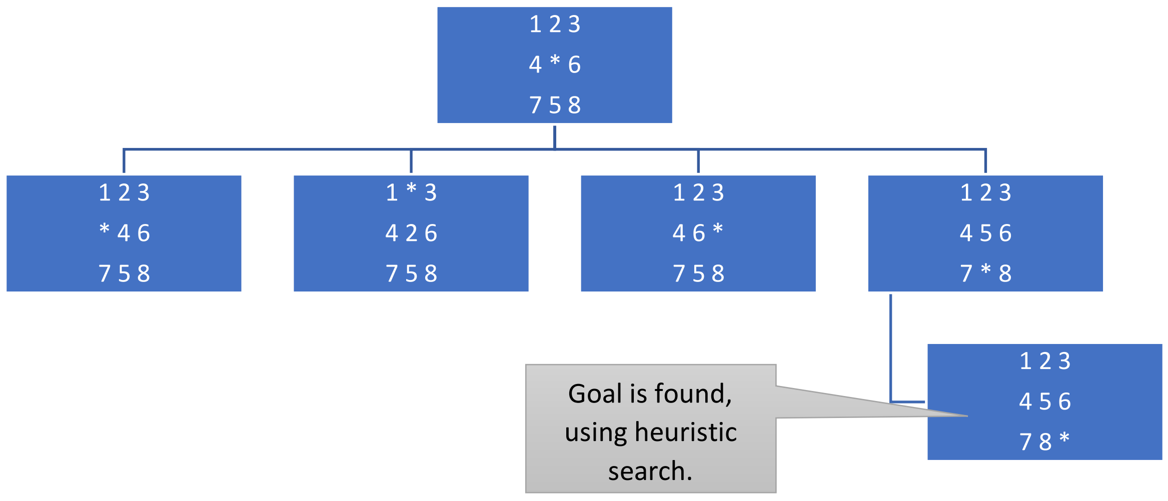
I will calculate this for each node, and the node with low f value will be processed first.

I will give short example how it works:

Stage 1:



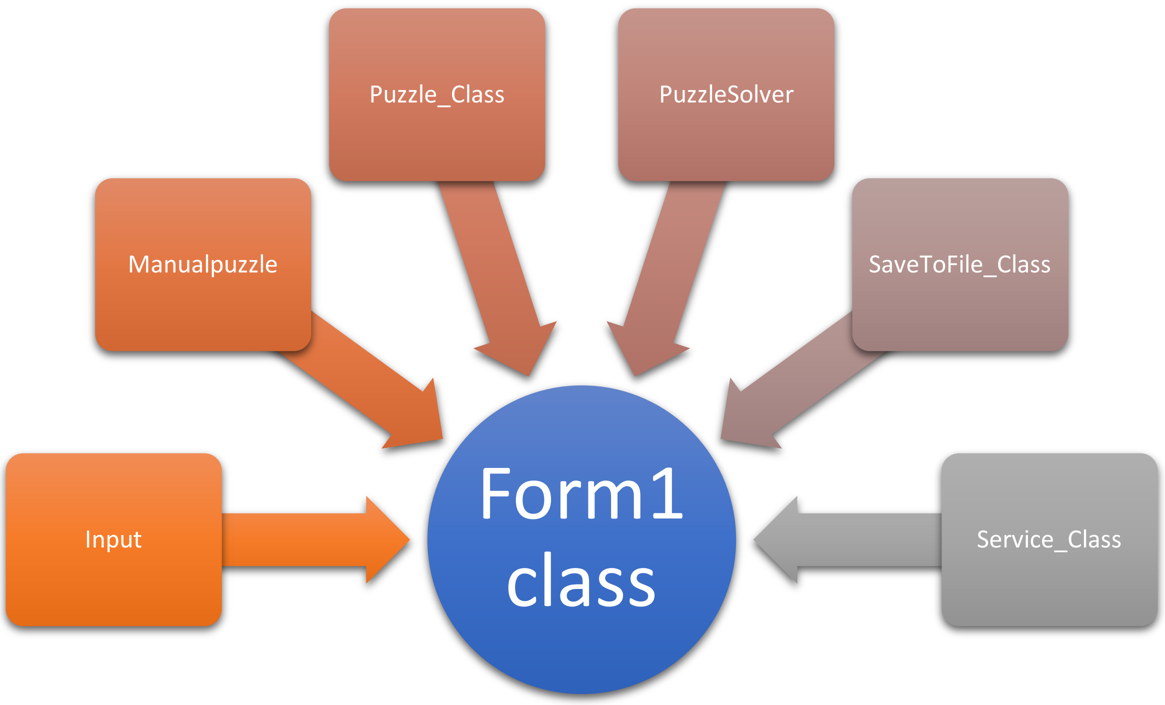
Stage 2:



2 Application classes and methods:

In this section I will explain application classes and methods as well as the function and use for each class and method.

Application classes



2.1 Form1:

It's the first class that will be executed when you run the application. All the other classes are connected to this class. This class contain as well the form1.designer.cs which will draw the application interface for the user.

Form1.cs contain the following variables:

Name	Type	Description
ST	`TimeSpan`	Hold the start time for the search graph.
MaxDepth	`int`	DFS depth limitation value
step	`int`	Hold number of steps to the goal.
puzzle	`string[,]`	Start state of the puzzle, as well as the current state we are in.
goal	`string[,]`	The goal state of the puzzle game

Form1.cs contain the following methods:

Name	Method variables	Description
`cmd_start_Click`	`object sender, EventArgs e`	Build the puzzle depending on user selection. And assign goal state.
`DrawPuzzle`	`string[,]`	Keeps track of moves history by adding the current state to listbox1
`Form1_Load`	`object sender, EventArgs e`	Generate max depth combo box.
`cmd_reshuffle_Click`	`object sender, EventArgs e`	Reshuffle the puzzle
`cmd_up_Click`	`object sender, EventArgs e`	Move * up.
`cmd_left_Click`	`object sender, EventArgs e`	Move * left.
`cmd_down_Click`	`object sender, EventArgs e`	Move * down.
`cmd_right_Click`	`object sender, EventArgs e`	Move * right.
`cmd_quit_Click`	`object sender, EventArgs e`	Clear game state.
`StartSolving_Click`	`object sender, EventArgs e`	Start one of the selected search graph.
`Solveit`	`Returns:` `Task<PuzzleSolver.SendBackInfo>`	Function that will be called by `StartSolving_Click` method to solve the puzzle.
`StartSolving`	`string searchType` `Return:` `SendBackInfo`	This will be called by Solveit method and it will run it in different thread.
`CreatePuzzleManually`	`string`	To create start state puzzle, and goal state.

2.2 Input class:

This class will be used to display the input form that will be used to enter cell value when selecting manually start state or goal state.

Methods:

Name	Method variables	Description

ShowDialog	`string text, string caption`
	Text = cell location.
	Caption= input form title.
		Display input message box to enter cell value.

2.3 ManualPuzzle class:

This class contain the following ***methods***:

Name	Method variables	Description

`CreatePuzzleManualy`	`int x`
	x = puzzle type.
		Initiate the pop form process for entering puzzle cells values.

2.4 Puzzle_Class:

This class contain the following ***methods:***

Name	Method variables	Description

`GetPuzzle`	`int N`
	`Returns:`
	`string[,]`
	`N=puzzle type.`
		The method creates the puzzle and return it as an array of string.

2.5 Service_Class:

This class contain the following ***methods:***

Name	Method variables	Description

`GetLocationOfEmpty`	`string[,] Puzzel`
	`Return:`
	`obj_location`
		Get the location of * within the puzzle. It will return I,j as object.

`MovePuzzel`	`string input, string[,] Puzzle.`
	`input= move direction.`
	`Puzzle= current puzzle you want to update.`
	`Return:`
	`string[,]`
		You pass the direction you want to move "*" as well as the puzzle "current state". The method will return new puzzle state after moving *.

Class objects:

Name	Description

`obj_location`	`public int i { get; set; }`
	`public int j { get; set; }`

2.6 SaveToFile_Class:

This class is used to save all the created nodes during the search process categorized by parent and depth into text file.

Class methods:

Name	Method variables	Description
savetofile	`List<nodes_info>` **Nodes_info** `is object defiend in PuzzelSolver class.`	Save all the node into text file. The nodes will be categorized by node parent and depth.

2.7 PuzzleSolver Class:

This class contain the three types of search techniques. It contains the following methods:

Name	Method variables	Description
DBFS	`string[,] Goale, string[,] State, int maxDepth, string SearchType Returns: SendBackInfo`	It will take goal state, start state, max depth search limitation value and search type (DFS or BFS). And it will return object `SendBackInfo`.
BestFS	`string[,] Goale, string[,] State. Goal= goal state. State= current state to start search from.`	It will take current state and goal state. It will perform Best First Search. And it will return object `SendBackInfo`.
getNodeChildren	`string[,] parentNode return: List<string[,]> “List of nodes that will present the children for the parent node.”`	Get the children for the given node. Which will be considered as parent node.
_MovePuzzel	`string input, string[,] PUZ Return: string[,] “Return puzzle after shifting the tiles”`	Is used by all types of search to drive the children for the given node. Input: is the moving direction. PUZ: is the parent node.
GetStepsToPostionANDTileInWrongPlace	`string[,] node, string[,] goal Return: int`	Get the number of misplace tile and number of moves for each misplaced tile to reach its goal. Return integer number that will present number of misplaced tiles + number of moves to reach goal for all misplaced tiles.
ISitTheSame	`string[,] a, string[,] b Return: bool`	Pass two nodes and the method will compare them to see if they are equal. Its used to compare to see if we reached the goal.

Objects used in this class:

Name	Details

SendBackInfo	`public List<nodes_info> ListNodesInfo { get; set; }`
	`public Boolean SoloutionFound { get; set; }`
	`public string depth { get; set; }`
	`public string Qleft { get; set; }`
	`public string startTime { get; set; }`
	`public string endTime { get; set; }`
	`public string totalTimeinMS { get; set; }`
	`public string totalParents { get; set; }`
	`public string totalNodes { get; set; }`
	`public string SearchType { get; set; }`

nodes_info	`public string[,] node_parent { get; set; }`
	`public List<string[,]> node_childrens {get; set;}`
	`public int depth { get; set; }`

Qsearch_Obj	`public string[,] Qnode;`
	`public int depth { get; set; }`

BestNodes_info	`public string[,] node_parent { get; set; }`
	`public List<BestNodeChildrens> node_childrens { get; set; }`
	`public int depth { get; set; }`
	`public int H { get; set; }`
	`public int HplusDepth { get; set; }`

BestNodeChildrens	`public string[,] node_childerns { get; set; }`
	`public int H { get; set; }`
	`public int depth { get; set; }`
	`public int HplusDepth { get; set; }`

bestQsearch_Obj	`public string[,] Qnode;`
	`public int depth { get; set; }`
	`public int H { get; set; }`
	`public int HplusDepth { get; set; }`

3 Application manual:

The application is not designed as final product to be shipped, so there will be some un tuned design as well as functions.

3.1 Main screen:

first time you start the application, the main screen will be displayed with multi options allowing you to select puzzle type, search type and more. See figure 1.

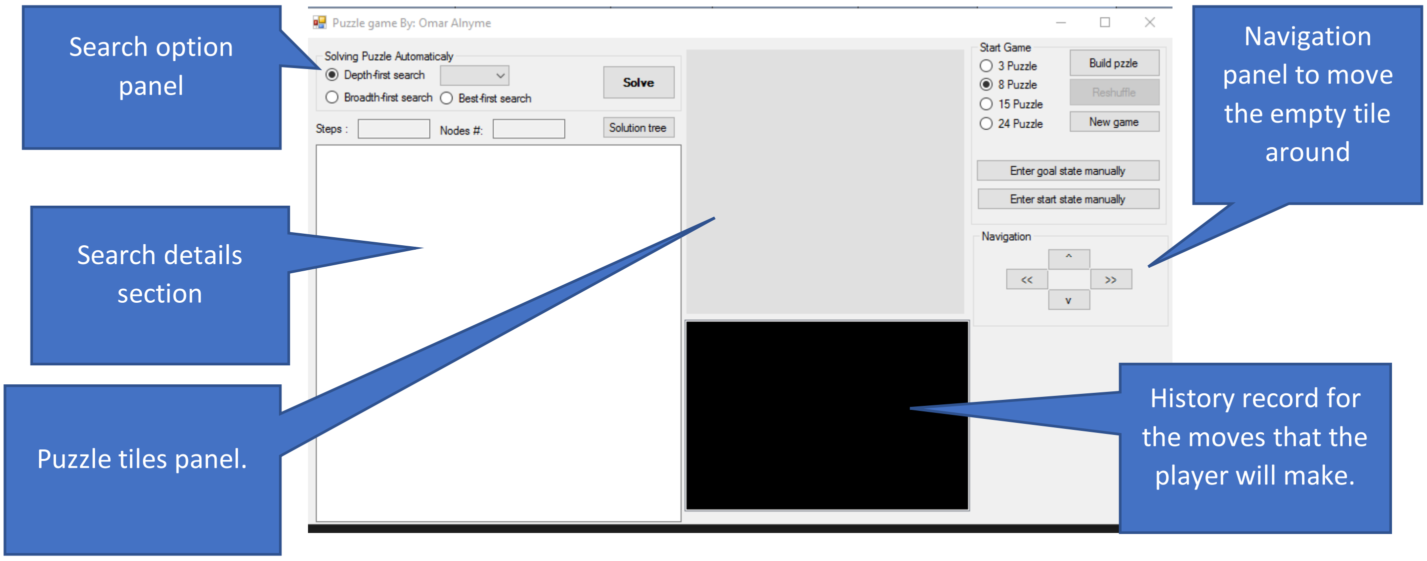
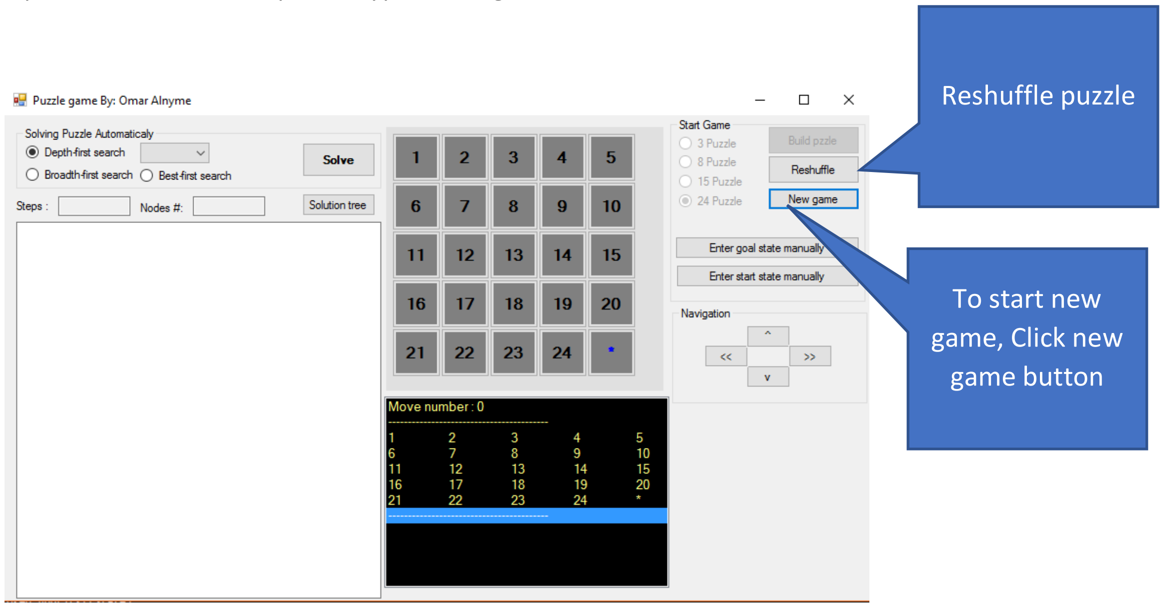


Figure 1

3.2 Build puzzle:

To build puzzle game, just select the puzzle type: 3, 8, 15, 24. And click **build puzzle** button. In our example, we will select 24 puzzle type. See figure 2.



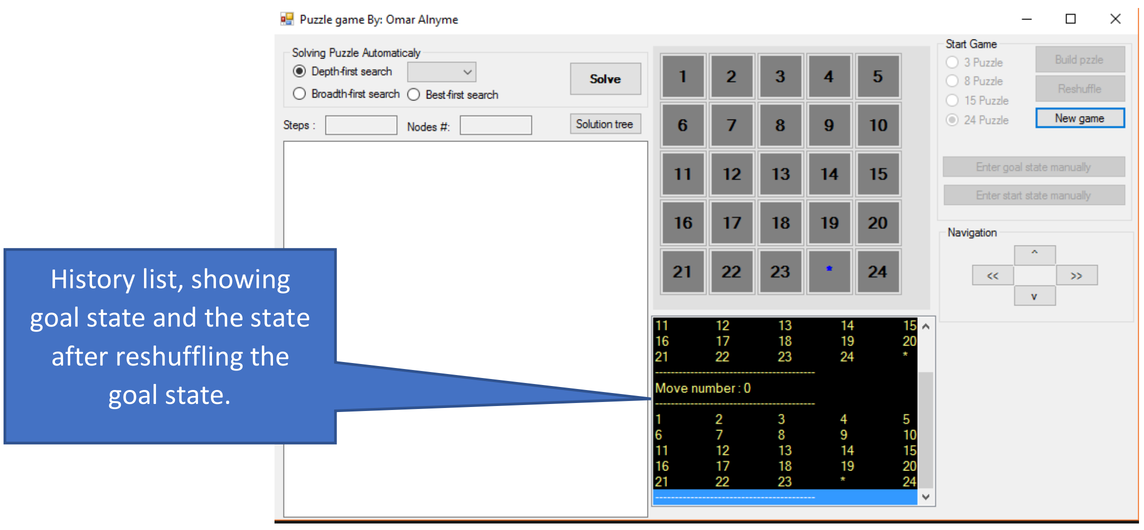
```
Move number : 0
------------------------------------------
1        2        3        4        5
6        7        8        9        10
11       12       13       14       15
16       17       18       19       20
21       22       23       24       *
------------------------------------------
```

Figure 2

By clicking build puzzle, reshuffle button the navigation and solving panel will be activated and build button will be deactivated. See figure 2.

3.3 Reshuffle puzzle tiles:

To reshuffle puzzle tiles just click **reshuffle** button Reshuffle. See figure 3.


Puzzle game By: Omar Alnyme
Solving Puzzle Automaticaly
Depth-first search
Breadth-first search
Best-first search
Solve
Steps :
Nodes #:
Solution tree
Start Game
3 Puzzle
8 Puzzle
15 Puzzle
24 Puzzle
Build pzzle
Reshuffle
New game
Enter goal state manually
Enter start state manually
Navigation
History list, showing goal state and the state after reshuffling the goal state.

```
11          12          13          14          15
16          17          18          19          20
21          22          23          24          *
---------------------------------
Move number : 0
---------------------------------
1           2           3           4           5
6           7           8           9           10
11          12          13          14          15
16          17          18          19          20
21          22          23          *           24
---------------------------------
```

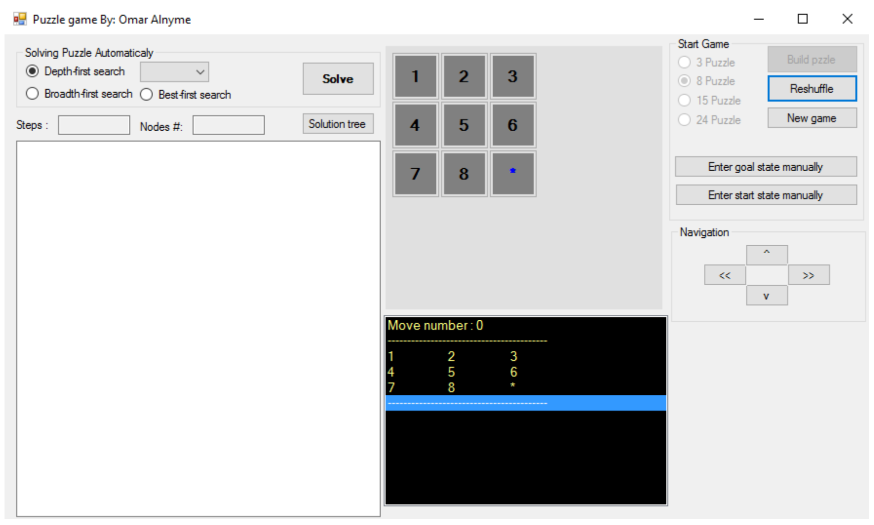
Figure 3

3.4 Set goal and start state manually:

You can create Goal/Start state manually, that is if you want the tiles to be arranged in different way in the goal state and/or start state.

3.4.1 Set goal state manually:

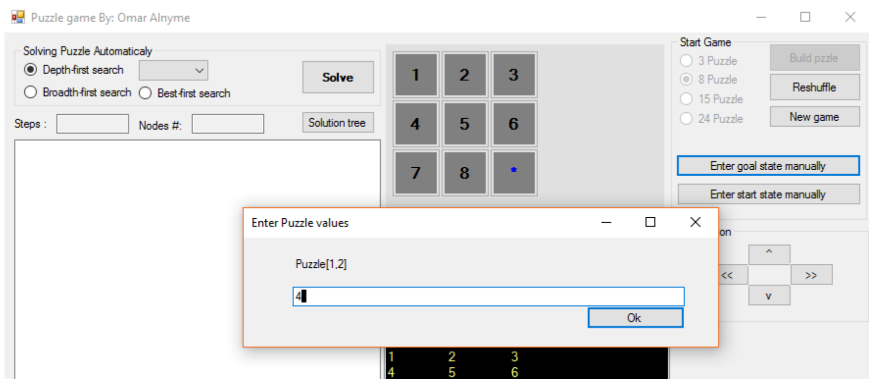
To set the goal state manually, you must build the puzzle first by selecting the type puzzle then click **build puzzle** button. In out example, we will create 8 tile puzzle. See figure 4.



```
Move number : 0
---------------------------------
1           2           3
4           5           6
7           8           *
---------------------------------
```

Figure 4

Now we want to change the goal state, so click Enter goal state manually button, after doing so the application will popup form for you to enter the puzzle values for each cell individually. See figure 5.



Puzzle[1,2]

4

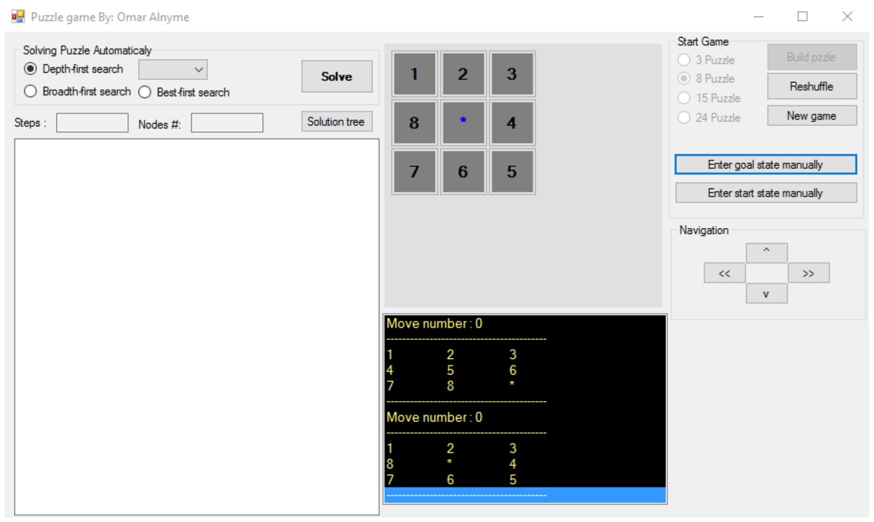
```
1        2        3
4        5        6
```

Figure 5

In our example, I will change the goal state from:

Old goal		New goal
1 2 3		1 2 3
4 5 6	→	8 * 4
7 8 *		7 6 5

See figure 6.

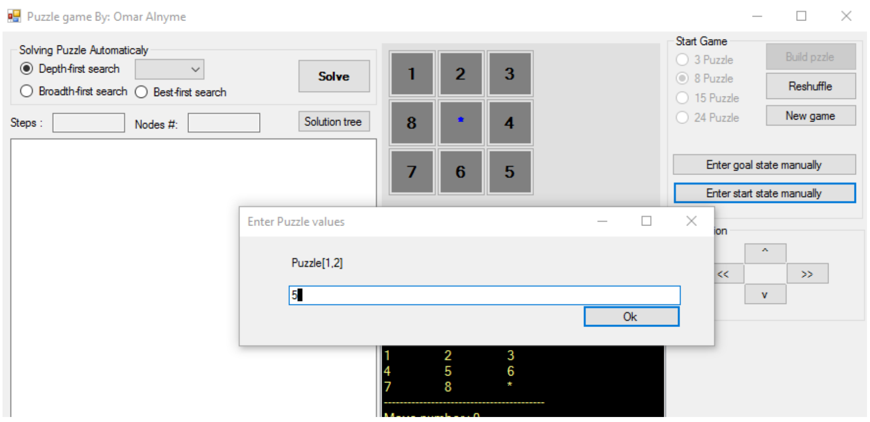


```
Move number : 0
---------------------------------
1        2        3
4        5        6
7        8        *
---------------------------------
Move number : 0
---------------------------------
1        2        3
8        *        4
7        6        5
---------------------------------
```

Figure 6

3.4.2 Set start state manually:

After building the puzzle and setting the goal either manually or using application default goal state, you can manually set the start state by clicking **Enter start state manually** button. By doing so, the application will popup form for you to enter the puzzle values cell by cell. See figure 7.



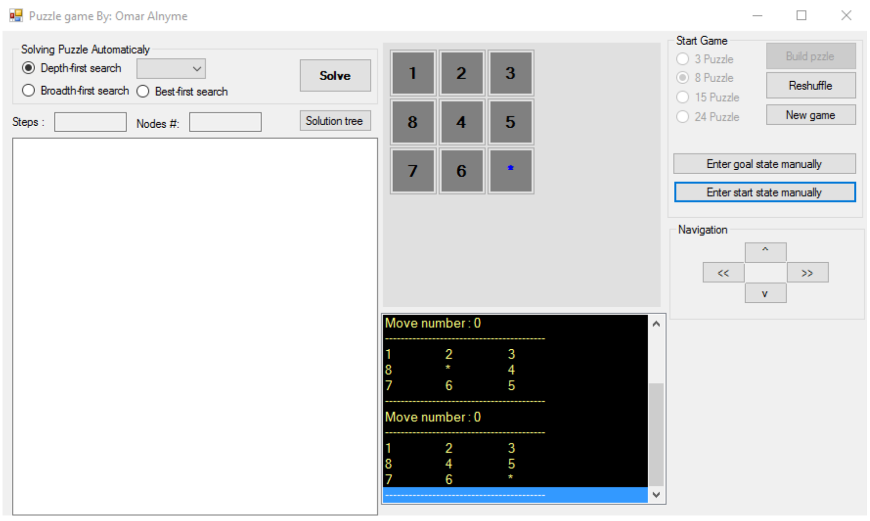
```
1        2        3
4        5        6
7        8        *
------------------------------
```

Figure 7

In our example, we will set the start state from:

Current state		New start state
1 2 3		1 2 3
8 * 4	→	8 4 5
7 6 5		7 6 *

See figure 8.

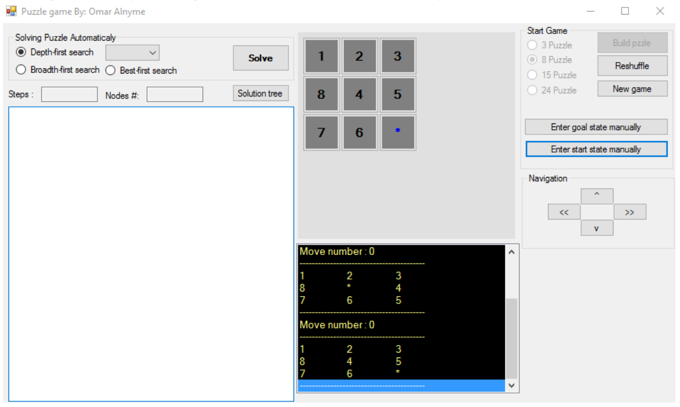


```
Move number : 0
------------------------------
1        2        3
8        *        4
7        6        5
------------------------------
Move number : 0
------------------------------
1        2        3
8        4        5
7        6        *
------------------------------
```

Figure 8

3.5 Solve the puzzle manually:

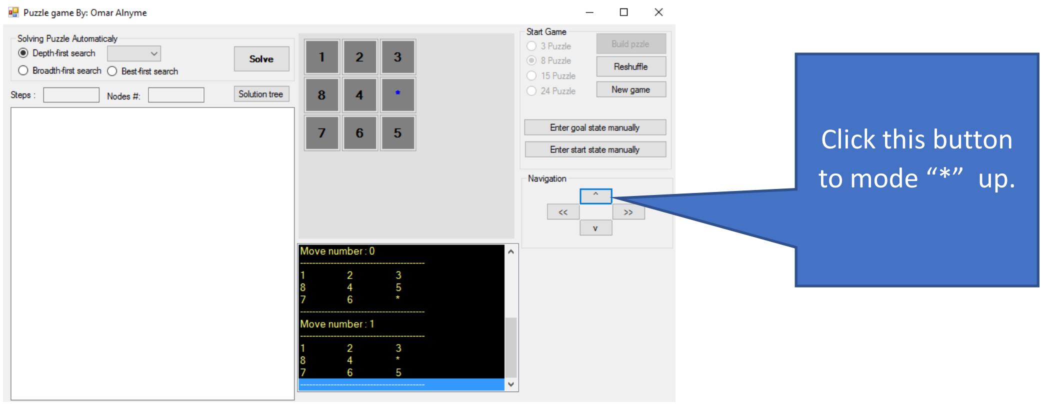
We can solve the puzzle manually by using navigation buttons.
We can move “*” right – up – right – down until the puzzle is solved. See figure 9.



```
Move number : 0
--------------------
1      2      3
8      *      4
7      6      5
--------------------
Move number : 0
--------------------
1      2      3
8      4      5
7      6      *
```

Figure 9

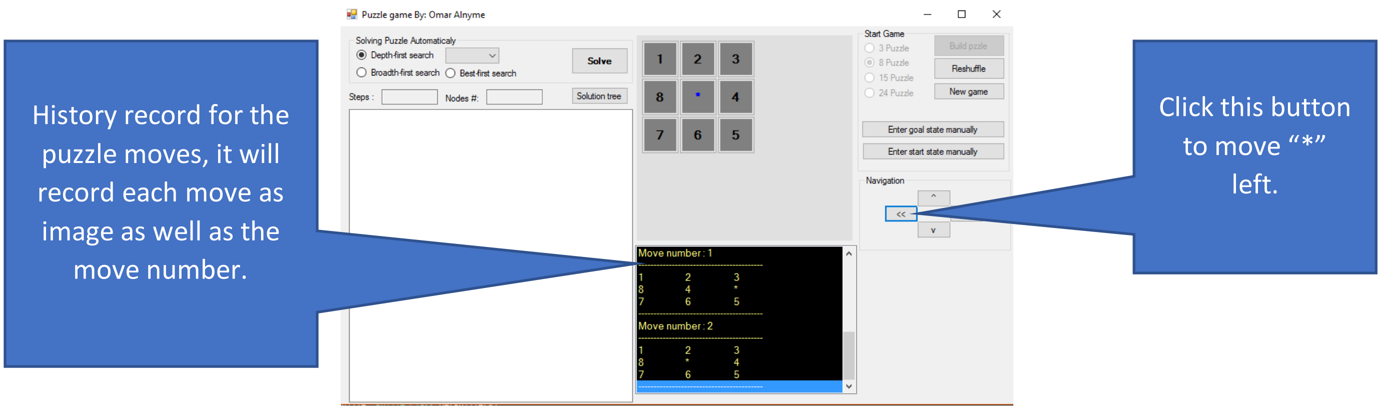
To solve this puzzle, we need to move “*” around, and we will start by moving it Up by clicking . See figure 10


Puzzle game By: Omar Alnyme
Navigation
Click this button to mode “*” up.

```
Move number : 0
--------------------
1      2      3
8      4      5
7      6      *
--------------------
Move number : 1
--------------------
1      2      3
8      4      *
7      6      5
```

Figure 10

Now we need to click to move “*” to the left. See figure 11.


Puzzle game By: Omar Alnyme
History record for the puzzle moves, it will record each move as image as well as the move number.
Click this button to move “*” left.

```
Move number : 1
--------------------
1      2      3
8      4      *
7      6      5
--------------------
Move number : 2
--------------------
1      2      3
8      *      4
7      6      5
```

Figure 11

3.6 solve puzzle automatically:

Now we have seen how we can solve manually, it's time to see how we can instruct the application to solve the puzzle for us automatically.

We have three types of graph search techniques that we can select from to solve the puzzle, and they are:

- ✓ Depth first search. We must specify the depth limitation or we will get stuck in a loop.
- ✓ Breadth first search.
- ✓ Best search.

3.6.1 Depth first search (DFS):

We will solve puzzle automatically by using DFS, in our example we will take 15 puzzle and try to solve it using DFS technique.

To build 15 puzzle:

- Select 15 puzzle radio button. See figure 12.

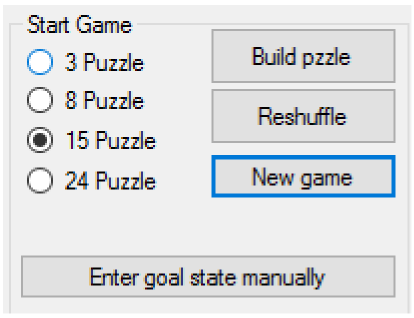
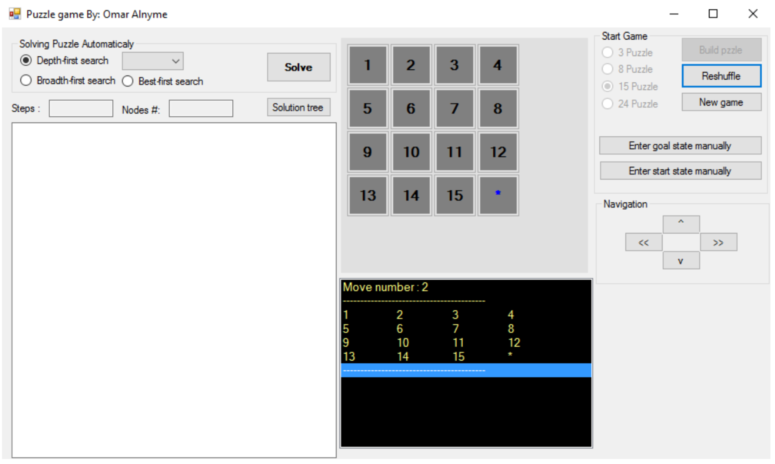


Figure 12

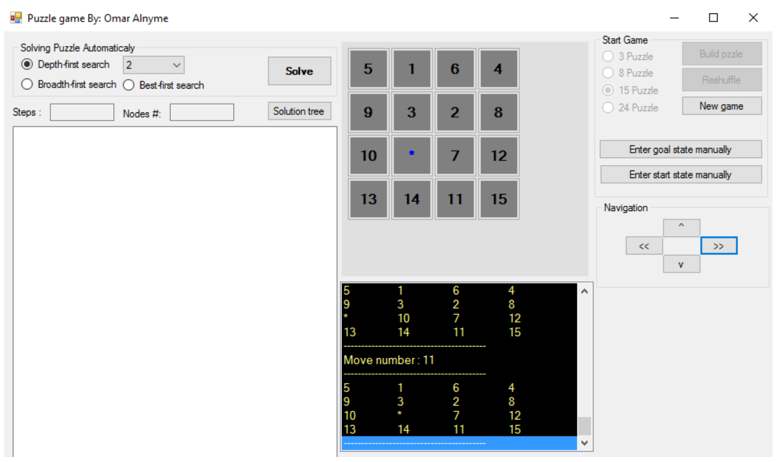
- Click **build puzzle** button. This will create puzzle of 15. See figure 13.



```
Move number : 2
----------------------------------------
1        2        3        4
5        6        7        8
9        10       11       12
13       14       15       *
----------------------------------------
```

Figure 13

- Now we will reshuffle the puzzle by clicking **Reshuffle** button. See figure 14.



Depth-first search: 2

```
5       1       6       4
9       3       2       8
*       10      7       12
13      14      11      15
----------------------------------
Move number : 11
----------------------------------
5       1       6       4
9       3       2       8
10      *       7       12
13      14      11      15
----------------------------------
```

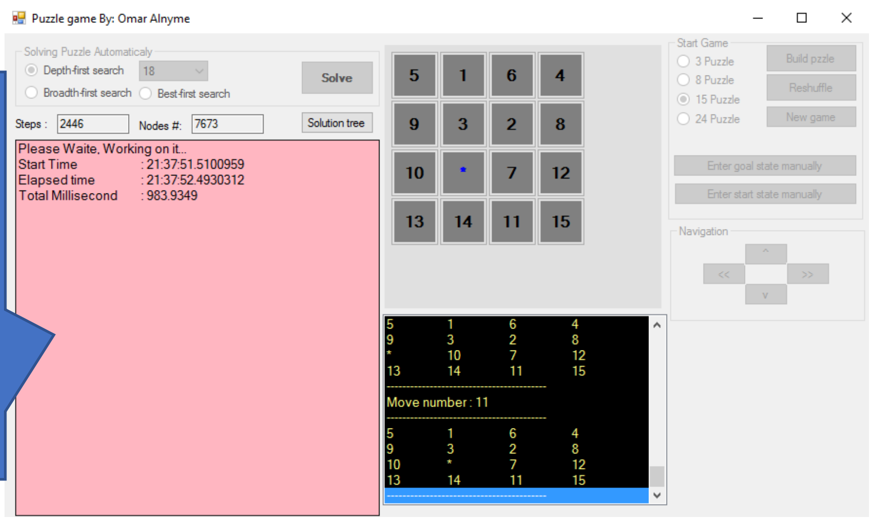
Figure 14

- To use DF search, select the ◉ Depth-first search .
- Select the depth limitation. ◉ Depth-first search 4 .
- Click **Solve** button. By doing so, the application will start the search process according to your selection.
    - During the search, the application search details section will be in red color displaying the detail of the search. See figure 15.

Search detail section.

Display search details such as:

- Start time.
- Elapse time.
- Total millisecond



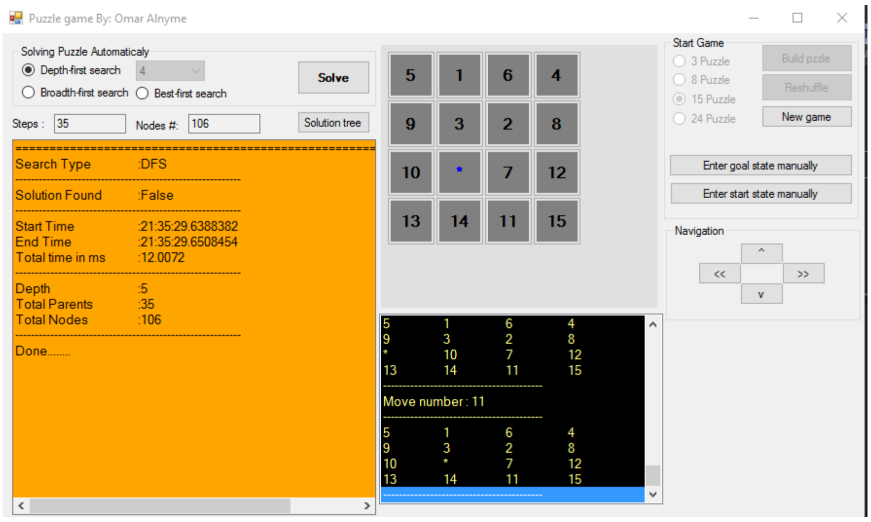
Depth-first search: 18
Steps : 2446 Nodes #: 7673

```
Please Waite, Working on it..
Start Time          : 21:37:51.5100959
Elapsed time        : 21:37:52.4930312
Total Millisecond   : 983.9349
```

```
5       1       6       4
9       3       2       8
*       10      7       12
13      14      11      15
----------------------------------
Move number : 11
----------------------------------
5       1       6       4
9       3       2       8
10      *       7       12
13      14      11      15
----------------------------------
```

Figure 15

- If the search is finished and the results is not found, the search detail section will be in orange color, and it will display some information regarding the search. See figure 16.



Depth-first search: 4

Steps : 35 Nodes #: 106

```
=====================================================
Search Type          :DFS
--------------------------------------------------
Solution Found       :False
--------------------------------------------------
Start Time           :21:35:29.6388382
End Time             :21:35:29.6508454
Total time in ms     :12.0072
--------------------------------------------------
Depth                :5
Total Parents        :35
Total Nodes          :106
--------------------------------------------------
Done........
```

5	1	6	4
9	3	2	8
10	*	7	12
13	14	11	15

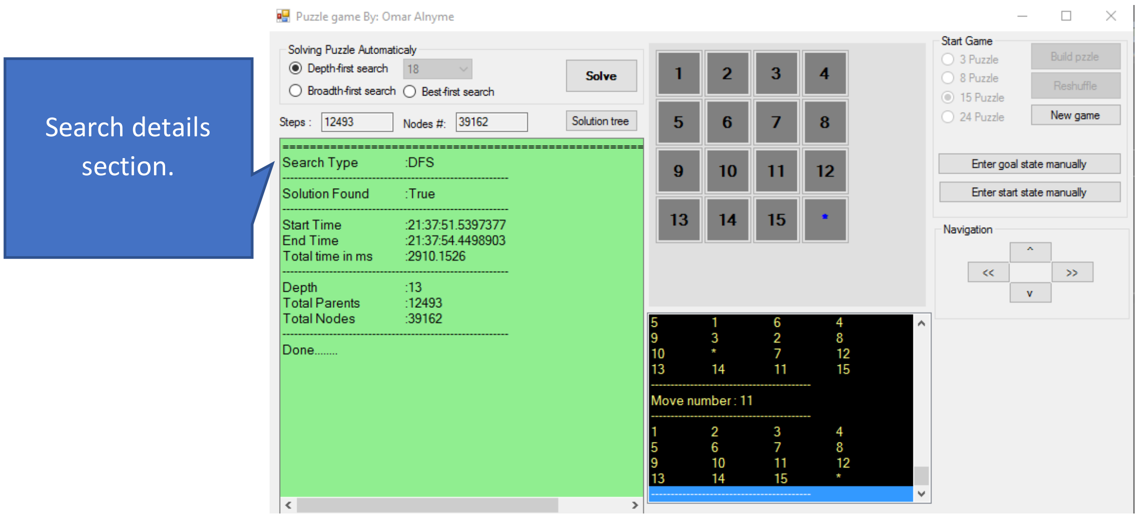
```
5       1       6       4
9       3       2       8
*       10      7       12
13      14      11      15
------------------------------------
Move number : 11
------------------------------------
5       1       6       4
9       3       2       8
10      *       7       12
13      14      11      15
------------------------------------
```

Figure 16

The search detail section will display the following:

- ✓ Search type.
- ✓ Search results either found or not.
- ✓ Start time.
- ✓ End time.
- ✓ Total time in milliseconds.
- ✓ Depth that the search was terminated.
- ✓ Total parent nodes.
- ✓ Total nodes.
- ✓ Total steps.

- ○ If the solution is found, the solution section will be in green color see figure 17.



Depth-first search: 18

Steps : 12493 Nodes #: 39162

```
=====================================================
Search Type          :DFS
--------------------------------------------------
Solution Found       :True
--------------------------------------------------
Start Time           :21:37:51.5397377
End Time             :21:37:54.4498903
Total time in ms     :2910.1526
--------------------------------------------------
Depth                :13
Total Parents        :12493
Total Nodes          :39162
--------------------------------------------------
Done........
```

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	*

```
5       1       6       4
9       3       2       8
10      *       7       12
13      14      11      15
------------------------------------
Move number : 11
------------------------------------
1       2       3       4
5       6       7       8
9       10      11      12
13      14      15      *
------------------------------------
```

Figure 17

3.6.2 Breadth first search (BFS):

We will solve puzzle automatically by using BFS, in our example we will take 15 puzzle and try to solve it using BFS technique.

To build 15 puzzle:

- Select 15 puzzle radio button. See figure 18.

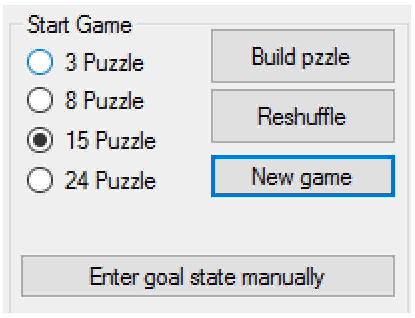
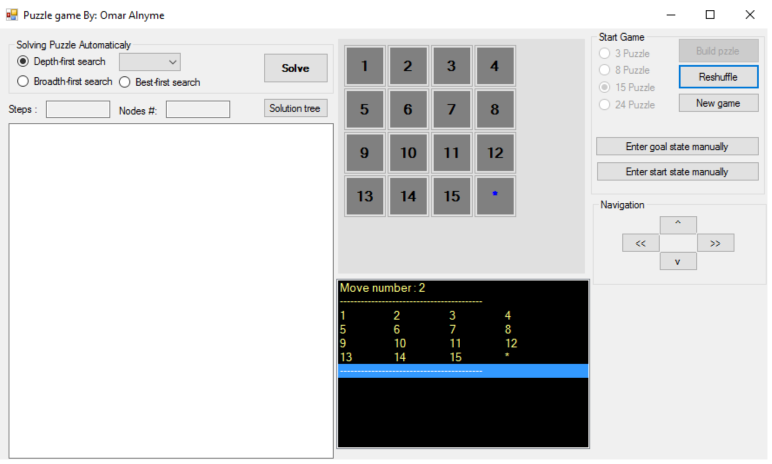


Figure 18

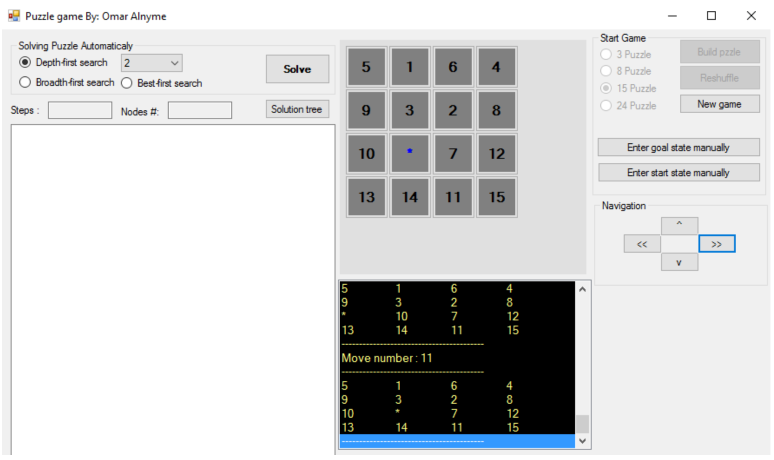
- Click **build puzzle** button. This will create puzzle of 15. See figure 19.



```
Move number : 2
----------------------------------------
1        2        3        4
5        6        7        8
9        10       11       12
13       14       15       *
----------------------------------------
```

Figure 19

- Now we will reshuffle the puzzle by clicking **Reshuffle** button. See figure 20.


Puzzle game By: Omar Alnyme
Solving Puzzle Automaticaly
Solve
Solution tree
Start Game
Build pzzle
Reshuffle
New game
Enter goal state manually
Enter start state manually
Navigation
<<
>>
^
v

[✓ Depth-first search] 2 ; Breadth-first search ; Best-first search

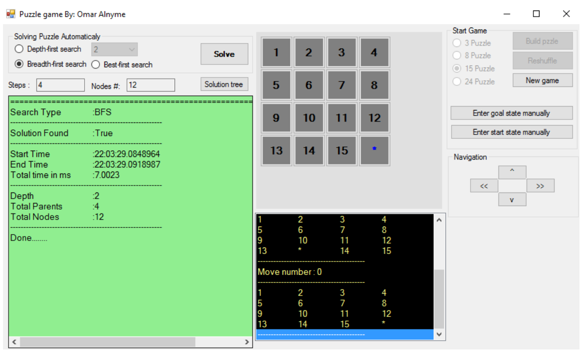
Steps : Nodes #:

3 Puzzle ; 8 Puzzle ; [✓ 15 Puzzle] ; 24 Puzzle

```
5       1       6       4
9       3       2       8
*       10      7       12
13      14      11      15
---------------------------------------
Move number : 11
---------------------------------------
5       1       6       4
9       3       2       8
10      *       7       12
13      14      11      15
---------------------------------------
```

Figure 20

- To use BF search, select the Breadth-first search .
- Click Solve button. By doing so, the application will start the search process according to your selection.
  - During the process of the search, search details section will be in red color Displaying the following:
    - Start time.
    - End time.
    - Total time in milliseconds.
  - If the search ends and the solution is not found, the search details will be in orange color. Displaying the following:
    - Search type.
    - Search results either found or not.
    - Start time.
    - End time.
    - Total time in milliseconds.
    - Depth that the search was terminated.
    - Total parent nodes.
    - Total nodes.
    - Total steps.
  - If the solution is found, search detail section will be green color. See figure 21.


Puzzle game By: Omar Alnyme
Solving Puzzle Automaticaly
Solve
Solution tree
Start Game
Build pzzle
Reshuffle
New game
Enter goal state manually
Enter start state manually
Navigation
<<
>>
^
v

Depth-first search 2 ; [✓ Breadth-first search] ; Best-first search

Steps : 4 Nodes #: 12

3 Puzzle ; 8 Puzzle ; [✓ 15 Puzzle] ; 24 Puzzle

```
==========================================
Search Type          :BFS
-----------------------------------------
Solution Found       :True
-----------------------------------------
Start Time           :22:03:29.0848964
End Time             :22:03:29.0918987
Total time in ms     :7.0023
-----------------------------------------
Depth                :2
Total Parents        :4
Total Nodes          :12
-----------------------------------------
Done........
```

```
1       2       3       4
5       6       7       8
9       10      11      12
13      *       14      15
---------------------------------------
Move number : 0
---------------------------------------
1       2       3       4
5       6       7       8
9       10      11      12
13      14      15      *
---------------------------------------
```

Figure 21

3.6.3 Best-first search:

We will solve puzzle automatically by using Best-first search, in our example we will take 15 puzzle and try to solve it using Best-first search technique.

To build 15 puzzle:

- Select 15 puzzle radio button. See figure 22.

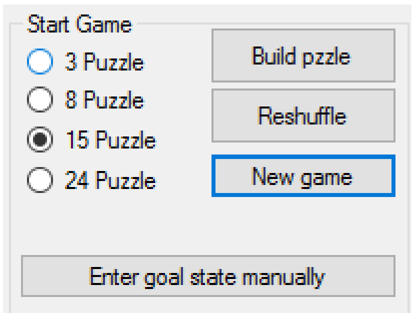
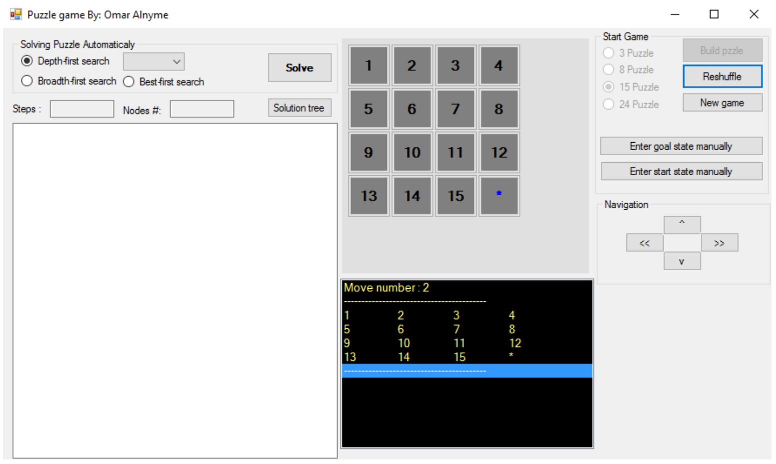


Figure 22

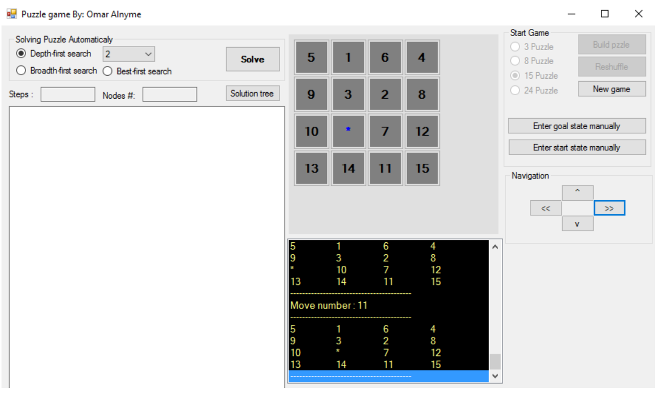
- Click **build puzzle** button. This will create puzzle of 15. See figure 23.



```
Move number : 2
------------------------------------
1        2        3        4
5        6        7        8
9        10       11       12
13       14       15       *
------------------------------------
```

Figure 23

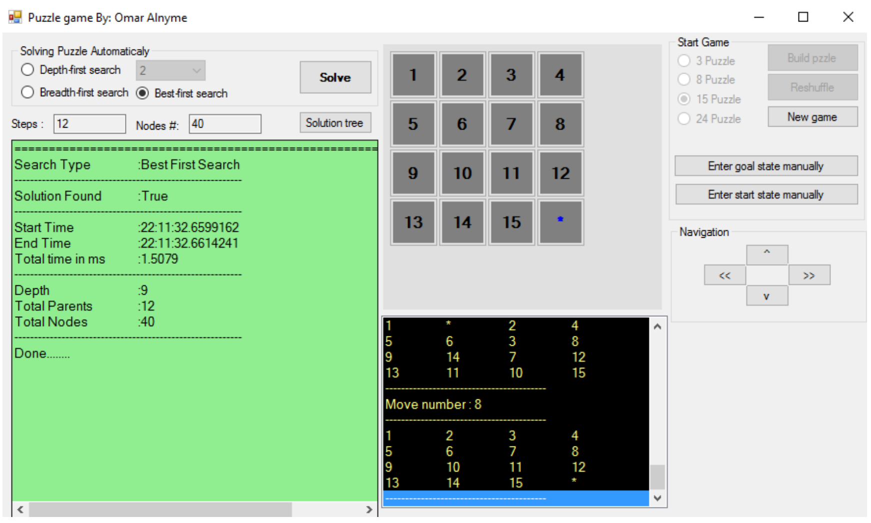
- Now we will reshuffle the puzzle by clicking Reshuffle button. See figure 24.



```
5        1        6        4
9        3        2        8
*        10       7        12
13       14       11       15
------------------------------------
Move number : 11
------------------------------------
5        1        6        4
9        3        2        8
10       *        7        12
13       14       11       15
------------------------------------
```

Figure 24

- To use Best-firstF search, select the ◉ Best-first search.
- Click Solve button. By doing so, the application will start the search processs according to your selection.
    - During the process of the search, search details section will be in red color Displaying the following:
        - Start time.
        - End time.
        - Total time in milliseconds.
    - If the search ends and the solution is not found, the search details will be in orange color. Displaying the following:
        - Search type.
        - Search results either found or not.
        - Start time.
        - End time.
        - Total time in milliseconds.
        - Depth that the search was terminated.
        - Total parent nodes.
        - Total nodes.
        - Total steps.
    - If the solution is found, search detail section will be green color. See figure 25.



Steps : 12 Nodes #: 40

```
==========================================================
Search Type          :Best First Search
---------------------------------------------
Solution Found       :True
---------------------------------------------
Start Time           :22:11:32.6599162
End Time             :22:11:32.6614241
Total time in ms     :1.5079
---------------------------------------------
Depth                :9
Total Parents        :12
Total Nodes          :40
---------------------------------------------
Done........
```

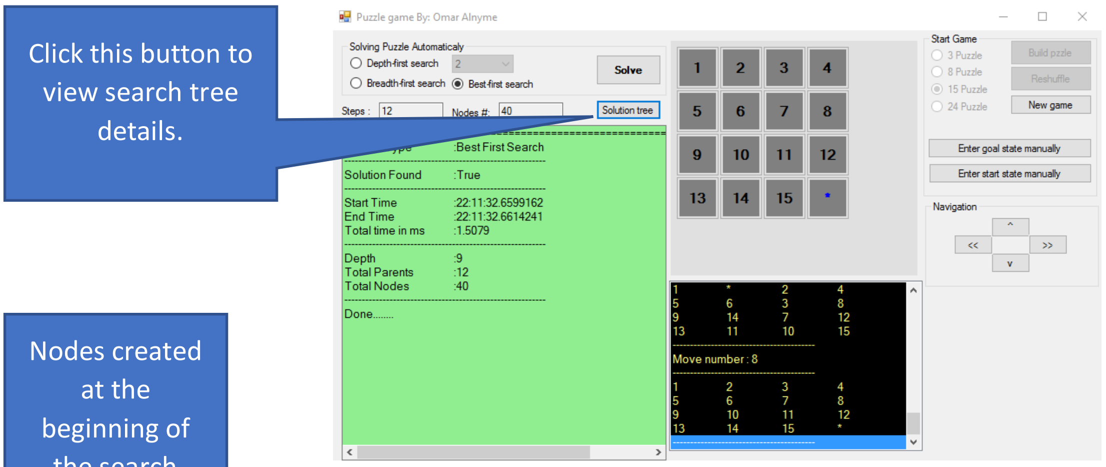
```
1       *       2       4
5       6       3       8
9       14      7       12
13      11      10      15
------------------------------
Move number : 8
------------------------------
1       2       3       4
5       6       7       8
9       10      11      12
13      14      15      *
------------------------------
```

Figure 25

4 Search nodes details:

After completing each search either it was performed using DFS, BFS or Best-first search, the application will generate text file that contain all the nodes that was created with their parents and depth. You can view this file by clicking Solution tree.

Now let's see the nodes that was created using Best-first search in our previous example. See figure 26, and figure



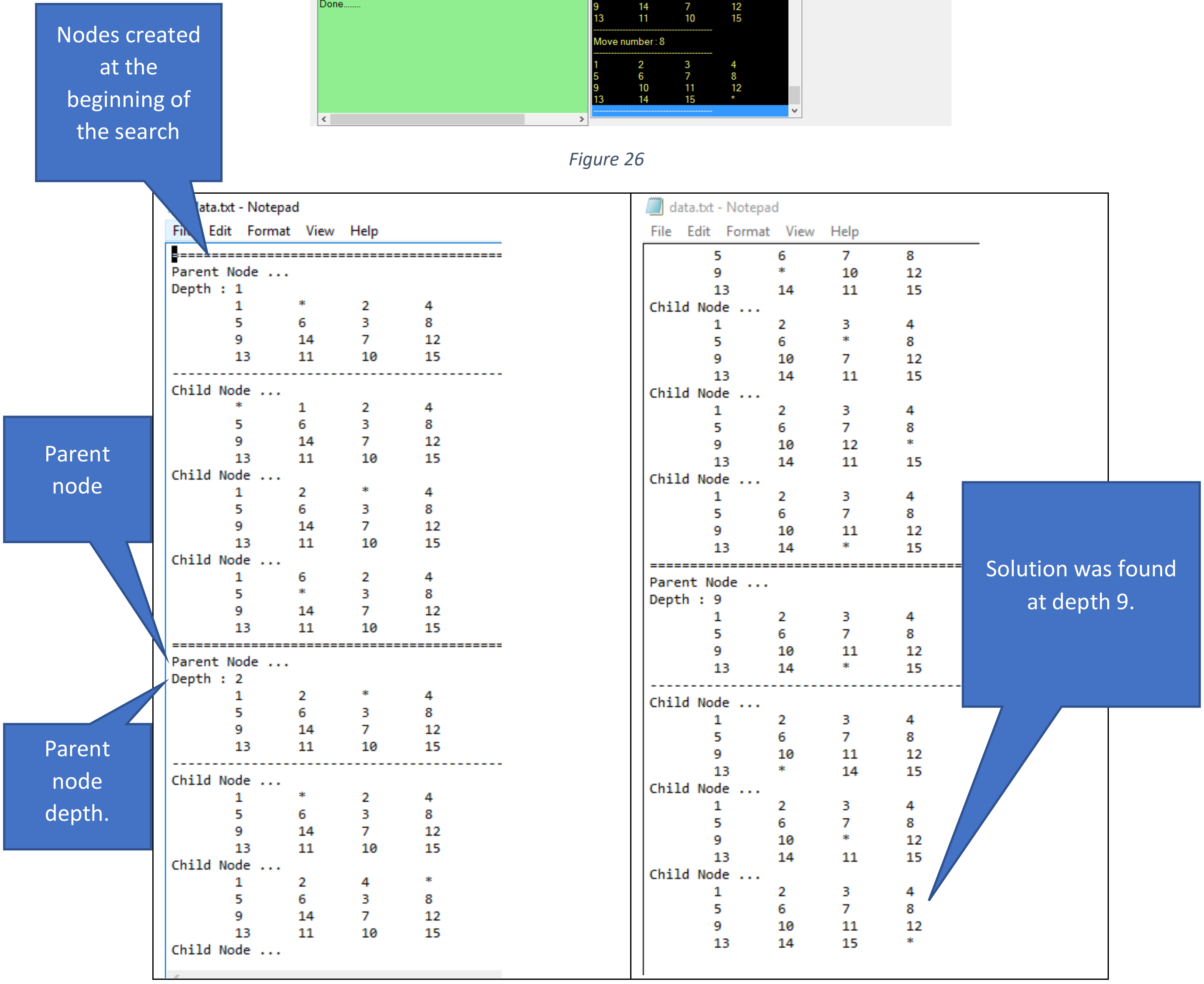
Solving Puzzle Automaticaly: ( ) Depth-first search [2] ( ) Breadth-first search [✓ Best-first search]

Steps : 12 Nodes #: 40

```
=================================
            :Best First Search
---------------------------------
Solution Found     :True
---------------------------------
Start Time         :22:11:32.6599162
End Time           :22:11:32.6614241
Total time in ms   :1.5079
---------------------------------
Depth              :9
Total Parents      :12
Total Nodes        :40
---------------------------------
Done........
```

```
1       *       2       4
5       6       3       8
9       14      7       12
13      11      10      15
---------------------------------
Move number : 8
---------------------------------
1       2       3       4
5       6       7       8
9       10      11      12
13      14      15      *
---------------------------------
```

Figure 26


Nodes created at the beginning of the search
Parent node
Parent node depth.
Solution was found at depth 9.
ata.txt - Notepad
data.txt - Notepad
File Edit Format View Help

```
========================================
Parent Node ...
Depth : 1
         1       *       2       4
         5       6       3       8
         9       14      7       12
         13      11      10      15
----------------------------------------
Child Node ...
         *       1       2       4
         5       6       3       8
         9       14      7       12
         13      11      10      15
Child Node ...
         1       2       *       4
         5       6       3       8
         9       14      7       12
         13      11      10      15
Child Node ...
         1       6       2       4
         5       *       3       8
         9       14      7       12
         13      11      10      15
========================================
Parent Node ...
Depth : 2
         1       2       *       4
         5       6       3       8
         9       14      7       12
         13      11      10      15
----------------------------------------
Child Node ...
         1       *       2       4
         5       6       3       8
         9       14      7       12
         13      11      10      15
Child Node ...
         1       2       4       *
         5       6       3       8
         9       14      7       12
         13      11      10      15
Child Node ...
```

```
         5       6       7       8
         9       *       10      12
         13      14      11      15
Child Node ...
         1       2       3       4
         5       6       *       8
         9       10      7       12
         13      14      11      15
Child Node ...
         1       2       3       4
         5       6       7       8
         9       10      12      *
         13      14      11      15
Child Node ...
         1       2       3       4
         5       6       7       8
         9       10      11      12
         13      14      *       15
========================================
Parent Node ...
Depth : 9
         1       2       3       4
         5       6       7       8
         9       10      11      12
         13      14      *       15
----------------------------------------
Child Node ...
         1       2       3       4
         5       6       7       8
         9       10      11      12
         13      *       14      15
Child Node ...
         1       2       3       4
         5       6       7       8
         9       10      *       12
         13      14      11      15
Child Node ...
         1       2       3       4
         5       6       7       8
         9       10      11      12
         13      14      15      *
```

Summary:

The application provide user options to select from four different type of puzzle:

- 3 puzzle.
- 8 puzzle.
- 15 puzzle.
- 24 puzzle.

The puzzle can be solved either manually using the navigation panel, or automatically by the application using one of the following graph search techniques:

1. Depth first search (DFS).
2. Breadth first search (BFS).
3. Best-first search.

After the application process the selected graph search technique, and in both cases, find the goal or did not manage to find the goal, the application will present the following information:

- Search type.
- Search results either found or not.
- Start time.
- End time.
- Total time in milliseconds.
- Depth that the search was terminated.
- Total parent nodes.
- Total nodes.
- Total steps.

Also, the application will hold in memory all the nodes that was generated during the search process and these nodes will be categorized by parents and depth. This information will be held in memory as list of objects. this list will be save to text file after the search is completed.

The text file can be accessed by clicking ***Solution tree*** button.