The 16-Puzzle Problem

A common puzzle is called a 16-puzzle. This puzzle consists of 15 tiles arranged in a 4 by 4 square. The tiles are numbers 1 to 15, and there is one position (the 16th) that is empty. In this way the tiles may be moved around. The object is to arrange the tiles in a pattern like the one below.

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

This problem can be solved by using a state space search. To be able to use a state space search, we need to: (1) Define a state. (2) Using the state definition, define the initial state and the goal state(s). (3) Define a set of rules. Each rule specifies how one state is transformed into another state along with the necessary conditions that must be present for the rule to be applied.

The following presents one solution to the problem. Let **`A[16]`** be an integer array. This array represents the board where the array positions are mapped to the puzzle in the following manner.

A[0]	A[1]	A[2]	A[3]
A[4]	A[5]	A[6]	A[7]
A[8]	A[9]	A[10]	A[11]
A[12]	A[13]	A[14]	A[15]

Let **`Space`** be an integer variable which hold the array index of the blank. Each tile 1 to 15 is represented by the corresponding integer and the blank is represented by 0. The state is represented as an ordered pair (**`A, Space`**).

An initial state is an ordered pair (**`A, Space`**), where the contents of **`A`** represent a permutation of the integers 0 … 15, and **`Space`** contains the index in **`A`** of the value 0.

The goal state is (**`A, Space`**) where **`Space`** = 0 and **`A`**[i] = i for 1 ≤ i ≤ 15.

There are four operators: **`Right`**, **`Left`**, **`Up`**, and **`Down`**. Each operator takes a state for input and produces a new state for output. This means that each operator is a mapping from the set of states to the set of states. Each operator has a condition that must be true if the operator can be applied and an action that it performs producing a new state if the condition is true. The definitions of the operators are:

Operator	Condition	Action Producing A New State
`Right`	**`Space % 4 != 3`**	**`Swap( A[Space], A[Space + 1] ), Space += 1`**
`Left`	**`Space % 4 != 0`**	**`Swap( A[Space], A[Space - 1] ), Space -= 1`**
`Up`	**`Space > 3`**	**`Swap( A[Space], A[Space - 4] ), Space -= 4`**
`Down`	**`Space < 12`**	**`Swap( A[Space], A[Space + 4] ), Space += 4`**