Solving the Sliding Puzzle Game

Author: Thomas Weber

Date: 20th May 2010

1. Sliding Puzzle

Quotation from Wikipedia [1]:

> *A sliding puzzle, sliding block puzzle, or sliding tile puzzle challenges a player to slide usually flat pieces along certain routes (usually on a board) to establish a certain end-configuration.*
>
> *The fifteen puzzle is the oldest type of sliding block puzzle. [...] Unlike other tour puzzles, a sliding block puzzle prohibits lifting any piece off the board. This property separates sliding puzzles from rearrangement puzzles. Hence finding moves, and the paths opened up by each move, within the two-dimensional confines of the board, are important parts of solving sliding block puzzles.*

Figure 1: a commercial fifteen sliding puzzle [2]

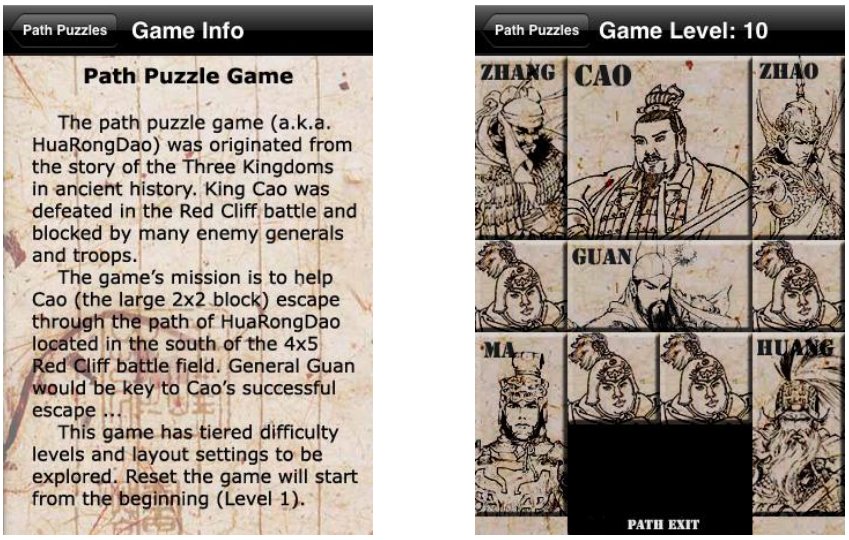

Figure 2: a path puzzle game as iPhone© app [3]

2. Principle of the solving algorithm

The solving algorithm presented here is based on the Dijkstra's algorithm. But before getting to the explanation of the Dijkstra's algorithm an introduction into graph theory will be provided to the reader.

2.1. Graph theory

In mathematics and computer science a graph consists of a certain number of nodes and a collection of edges that connects pairs of nodes. Each node as well as the edges might have a single value or a whole set of properties. In our case each node represents a unique state of the puzzle, while the edges indicate that both connected states can be reached each other by just moving one stone for one field.

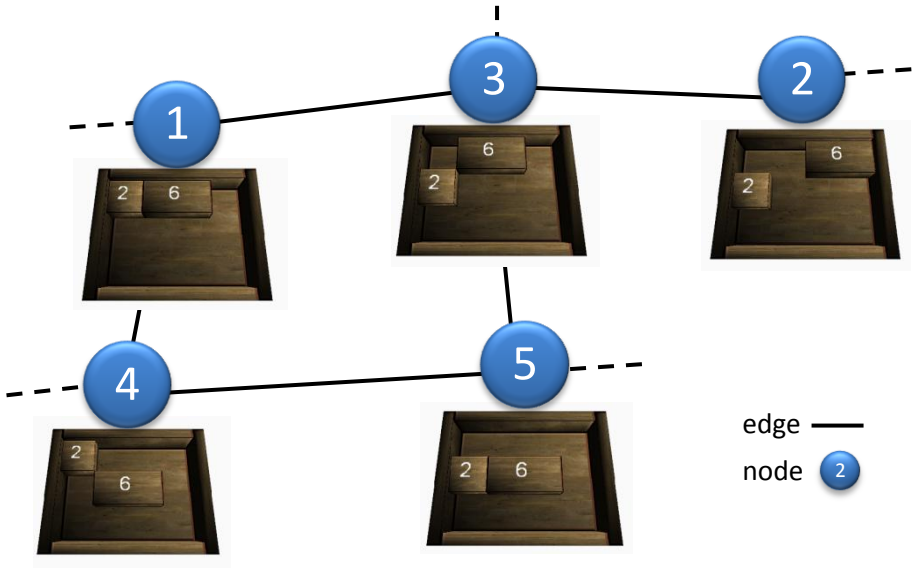

Figure 3: sketch of a simple graph

Finding a solution is done by performing the following three steps:

- Find all unique states and store them into a list of nodes
- Find all connected states and put them into a list of edges
- Find the shortest path between the initial and final state using Dijkstra's algorithm

2.2. Dijkstra's algorithm

Quotation from Wikipedia [4]:

> *Dijkstra's algorithm, conceived by Dutch computer scientist Edsger Dijkstra in 1956 and published in 1959, is a graph search algorithm that solves the single-source shortest path problem for a graph with nonnegative edge path costs, producing a shortest path tree. This algorithm is often used in routing and as a subroutine in other graph algorithms.*
>
> *For a given source vertex (node) in the graph, the algorithm finds the path with lowest cost (i.e. the shortest path) between that vertex and every other vertex. It can also be used for finding costs of shortest paths from a single vertex to a single destination vertex by stopping the algorithm once the shortest path to the destination vertex has been determined. For example, if the vertices of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm can be used to find the shortest route between one city and all other cities. As a result, the shortest path first is widely used in network routing protocols.*

Let the node at which we are starting be called the initial node. Let the distance of node Y be the distance from the initial node to Y. Dijkstra's algorithm will assign some initial distance values and will try to improve them step by step.

1. *Assign to every node a distance value: set it to zero for our initial node and to infinity for all other nodes.*
2. *Mark all nodes as unvisited. Set initial node as current.*
3. *For current node, consider all its unvisited neighbors and calculate their tentative distance. For example, if current node (A) has distance of 6, and an edge connecting it with another node (B) is 2, the distance to B through A will be 6+2=8. If this distance is less than the previously recorded distance, overwrite the distance.*
4. *When we are done considering all neighbors of the current node, mark it as visited. A visited node will not be checked ever again; its distance recorded now is final and minimal.*
5. *If all nodes have been visited, finish. Otherwise, set the unvisited node with the smallest distance (from the initial node, considering all nodes in graph) as the next "current node" and continue from step 3.*

3. Implementation

development environment:	MS Visual Studio 2008
programming language:	C++
used libraries:	Standard Template Library (STL)
	MS Win32 API
	MS DirectX 9 SDK (April 2007)
library files:	Msimg32.lib, d3d9.lib, d3dx9.lib, ddraw.lib, comctl32.lib, shlwapi.lib

3.1.3.1 Program hierarchy

The following Figure 4, modeled on the cpp-source code files, shall give the reader a broad overview of the program structure. Since the whole project has been growing gradually here, unfortunately, no stringent use of a class hierarchy has been implemented. Nevertheless, the traga2Class, traga2Win, ButtonSystem and direct3dWin are independently deployable components, e.g. can also be used for other mechanical puzzles games.

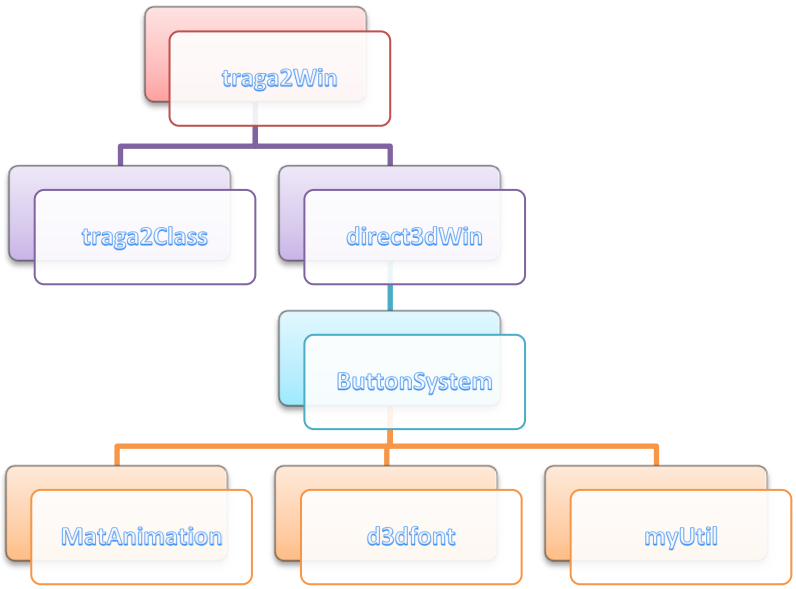

Figure 4: sketch of the program hierarchy

4. The graphical user interface

Directly after starting the program displays a window with three boards of which the left represents the chosen initial state, the middle the current state and the right the selected final state, see Figure 5.

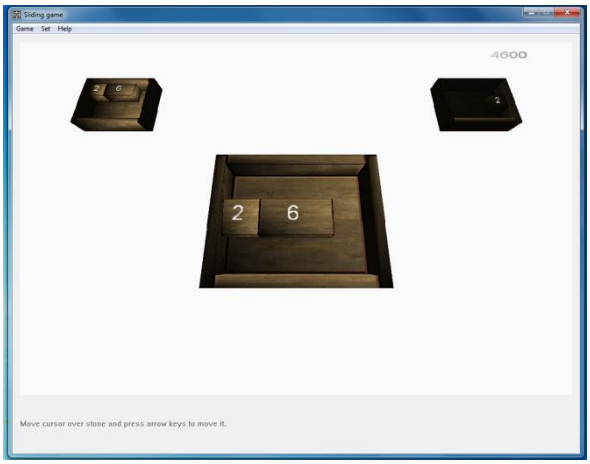

Figure 5: initial situation after starting the program

The first step should be the selection of the desired board size using the menu bar at the top [menu bar ➞ Set ➞ Set Board Size]. The second and third step are to define the initial and final states [menu bar ➞ Set ➞ Set Initial / Final State | see Figure 6]. The left parts/state are/is, so to speak, the parts depot, while the right board corresponds to the desired initial and final states. Select one stone by clicking left on it and transfer it to the right state with a second left-click.

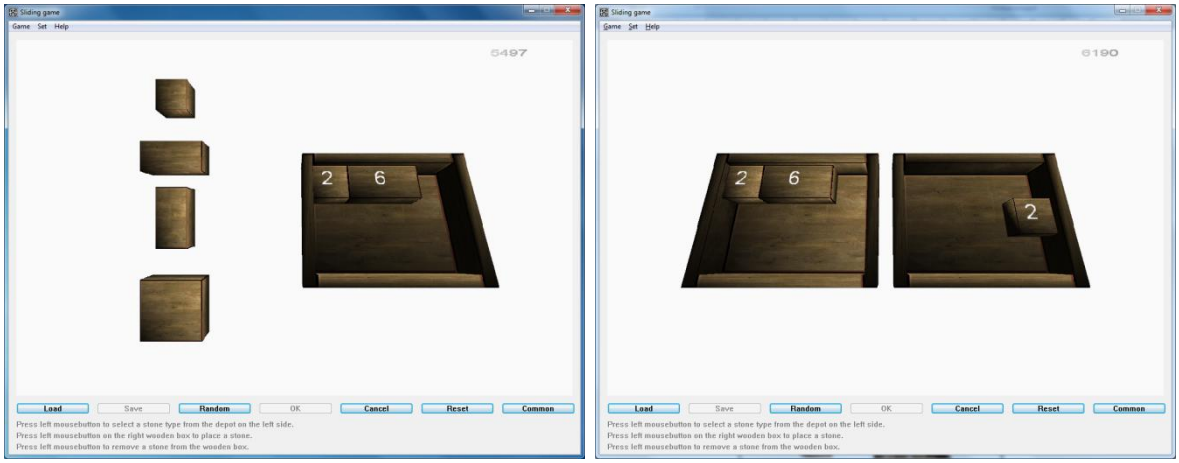

Figure 6: definition of the initial and end states

Now, we can already start. To solve the sliding puzzle manually select "Game" and then "Solve" in the menu bar, see Figure 7.

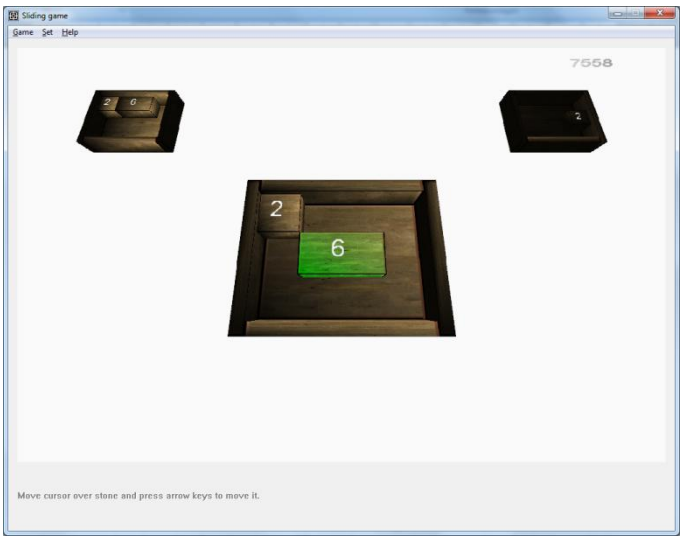

Figure 7: manual solving mode

In order to calculate numerically a solution, perform the following action: [menu bar → Game → Solve | see Figure 8]. Although a separate thread is initialized, the user has to wait until the whole computation is finished until the first steps of the solution path can be displayed.

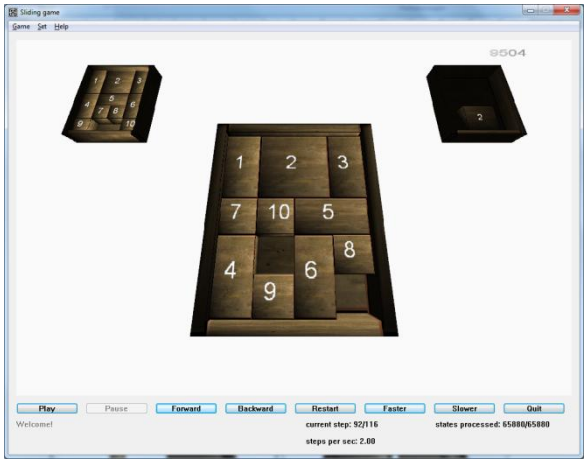

Figure 8: calculation and presentation of a proper solution

References

1. http://en.wikipedia.org/wiki/Sliding_puzzle
2. http://www.yo-promotions.co.uk/images/products/OTHER/Sliding-Puzzle-Game.jpg
3. http://www.aicio.com/path/
4. http://en.wikipedia.org/wiki/Dijkstra's_algorithm