Implementing an intelligent version of the classical sliding-puzzle game for unix terminals using Golang's concurrency primitives

Pravendra Singh

Department of Computer Science and Engineering • Indian Institute of Technology, Roorkee, India

http://pravj.github.io • hackpravj@gmail.com

Abstract

A smarter version of the sliding-puzzle game is developed using the Go programming language. The game runs in computer system's terminals. Mainly, it was developed for UNIX-type systems but bacause of cross-platform compatibility of the programming language used, it works very well in nearly all the operating systems.

The game uses Go's concurrency primitives to simplify most of the hefty parts of the game. Real time notification functionality is also developed using language's in-built concurrency support.

Keywords: Artificial Intelligence, Concurrent Programming, Programming Languages, Golang, Game Development

1. Introduction

Sliding puzzles have their own reputation in the world of artificial intelligence and graph theory since a long. The oldest type of sliding puzzle game is known as fifteen puzzle. It is believed to be invented in 1874 by Noyes Palmer Chapman, a postmaster in New York state. The game consists of a 4x4 board with 16 tiles in it. Each tile have numbers drawn on it except one tile, which is either blank or sometimes have digit '0' on it. The task for the game is to re-order all the tiles in a particular manner, where you are only allowed to move the blank tile at a time.

This type of puzzles have many variants also. The 3x3 board puzzle being fairly popular among them, also known as the 8-puzzle game. In this type, the game board consists of 9 tiles. In this paper, we will use the 8-puzzle game board into consideration.

On the other hand, Go is a new programming language initially developed at Google, also known as Golang. It's a compiled and statically-typed language with in-built support for concurrent programming. Golang is gaining popularity as the language for system programming and developer operations.

2. Solvability

Here, the standard 8-puzzle game board is being considered, where the task of the game is to put the tiles in such a way that the last tile is blank and all other tiles have numbers in increasing order(from 1 to 8). A total of 362880(9!) board configurations are possible but only half of them are actually solvable according to our constraints.

So the game generates random configurations of the game board and uses an in-built package named 'scanner' to scan the board for its solvability. If the board is not really solvable then it generates a new one recursively.

For its implementation, package 'scanner' uses the discussion from the paper "Notes on the 15 puzzle" by Wm. Woolsey Johnson. A simplified version of it can be found on the "Analysis of Sixteen Puzzle" by Kevin Gong.

Package 'scanner' implements this algorithm to check for solvability of any board configuration. Its time complexity is O(n^2). The algorithm returns a boolean value and an integer value, respectively implying whether a board is solvable or not and the position of blank tile in the board, if any.

```
func IsLegal(size int, values []int) (bool, int) {
    var inversions int
    n := len(values)

    for i := 0; i < (n - 1); i++ {
        if (values[i] != 0) && (values[i] != 1) {
            for j := i + 1; j < n; j++ {
                if (values[j] != 0) && (values[i] > values[j]) {
                    inversions++
                }
            }
        }
    }

    if (size%2 == 1) && (inversions%2 == 0) {
        return true, zeroIndex(values)
    }
    return false, -1
}
```