Klotski Puzzle and Klotski Solver

Introduction:

Klotski is a sliding block puzzle which was invented around 20th century. The purpose of this game is to move the biggest square to the exit via sliding blocks. The players can only move blocks in vertical or horizontal direction, and cannot remove any block from the game board. Through many years of development, Klotski has many variants and different openings (Figure 1).

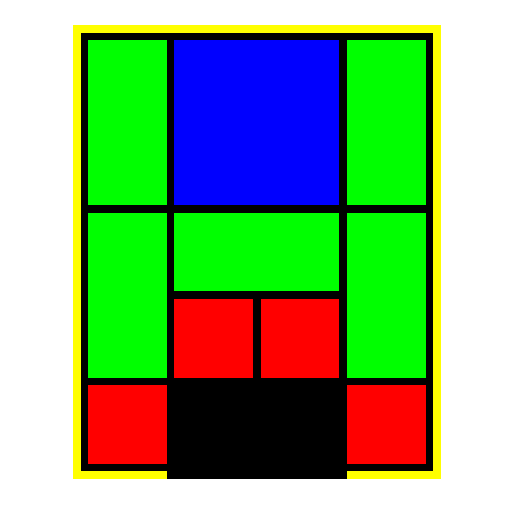


Figure 1: the most basic opening of Klotski

Exit

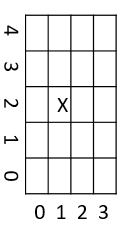
Although Klotski’s rules are very simple, it actually is not an easy puzzle. For example, the Klotski in Figure 1 requires at least 81 steps to win. Many people give up midway and no longer want to play Klotski anymore. Therefore, I write this Ocaml program, which allows users to

1. play Klotski with three different openings
2. get help from the computer to automatically solve the puzzle at any time, and watch the solution.

Design:

1. **Playing Klotski with three different openings**

The gamw2 board itself can be represented as a 4X5 table (Figure 2). Each cell in this table can be described by its x coordinate and y coordinate (named as box\_number in the actual program). For example, the cell marked X in Figure 2 is described as (1, 2).



There are total ten blocks in Klotski. Each of them is assigned with a name and color (Table 1).

|  |  |  |
| --- | --- | --- |
| Description | Assigned Name | Assigned Color |
| 2X2 square | A | Blue |
| 1X2 rectangle | B1 | Orange |
| 1X2 rectangle | B2 | Orange |
| 1X2 rectangle | B3 | Orange |
| 1X2 rectangle | B4 | Orange |
| 2X1 rectangle | C | Red |
| 1X1 square | D1 | Green |
| 1X1 square | D2 | Green |
| 1X1 square | D3 | Green |
| 1X1 square | D4 | Green |

Figure 2: A graphics representation of the gaming board

Table 1: Blocks name and color assignment

Each block can then be described as a tuple. The first element of the tuple is the name of the block, and the second element is the property of the block. The property of a block is a variant type which contains 5 integer values: x, y, w, h and c. x and y indicate the location of the block. w and h are the width and height of the block, and c is the color of the block. For example, a block named “D1”, whose color is green, height is 1, width is 1 and location is at position 0, 0, is represented as ("D1", {x=0; y=0; w=1; h=1; c=0x5cf442}).

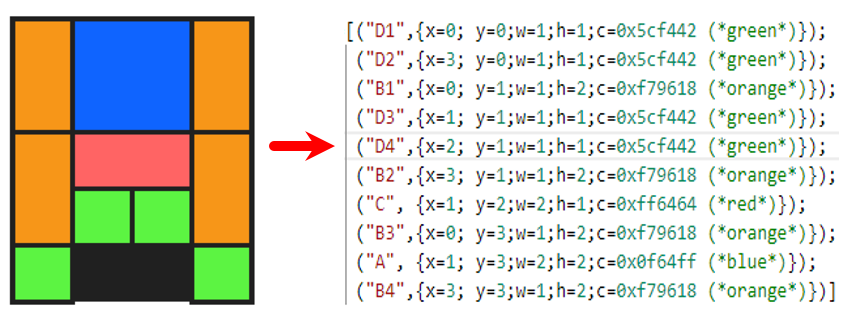
Thus, the game board can be easily represented as a block list (Figure 3). For some blocks which occupy more than one cell, e.g. block A, the x and y indicate the lower-left corner of the block. However, the program still knows they occupy more than one cell, because their width and/or height are larger than 1. 

Figure 3: Use a block list to represent a board

Through this method, the game board can be understood by the computer. The three different openings is achieved easily via using different block lists.

Once a user makes a click on the screen, the program will determine which item it clicks. If it clicks

* 1. a button, then perform this button’s action.
  2. a block, then check the block’s neighboring cells.
     1. if this block is movable and can only be moved toward one direction, move the block, generate a new block list and refresh the screen.
     2. if this block is movable but can be moved toward two different directions, ask for the user to determine which direction it should go, move the block, generate a new block list and refresh the screen.
     3. if this block is not movable, do not do anything but refresh the screen with the original block list.
  3. anything else, then do not do anything but refresh the screen with the original block list.

Every time the screen is refreshed, the program will check the board list. If the board list is in a victory stage, i.e. block A’s x =1 and y=0, program will generate a victory screen to congratulate the user and end the game.

There is a special case when the mouse clicks at the cross section, horizontal aisle or vertical aisle between cells. If this happens, the program will think this click does not hit on any block. However, this is not correct, sometimes. Thus, an additional function is added - if the mouse clicks at cross section, horizontal aisle or vertical aisle, the program will check the neighboring cells. If they are belongs to the same block, the mouse actually hits a block!

1. **Auto solve**

When a user clicks the “Auto Solve” button, the program will take the current block list as an input, and use breadth-first search (BFS) to generate a solution tree (Figure 4).

After the solution tree is generate, a backtrack function is called to backtrack the solution tree to get a path. This path is our solution. A small animation will display the solution to the user.

Future Directions:

The auto solving algorithm can be improved. The current solution of Figure 1 generated by the program is 117 steps. However, the lowest possible solution is only 85 steps. Thus, there is some rooms of improvement.

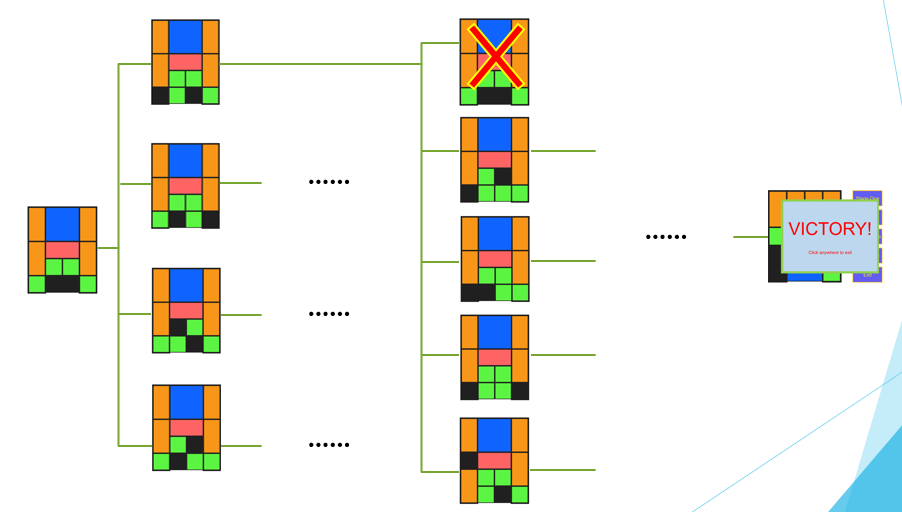


Figure 4: The BFS of the auto solve function

The program will try to move every block in the input to generate a different block list. All the different block lists generated will be stored as “leaves” and then used as new inputs to generate more “leaves”. This BFS will continue until the victory state is achieved.

If a “leaf” is encountered before, e.g. the “leaf” marked with a red cross in the figure, will be ignored to save calculation. No new “leaf” will be generated from it.