//Weiheng Su (ws2ta)

//puzzle app report

//external source used: https://www.ocf.berkeley.edu/~jchu/publicportal/sudoku/0011047.pdf

//12/15/2015

Report

App introduction: My application is based on Donald E. Knuth’s Dancing Link algorithm (see: https://www.ocf.berkeley.edu/~jchu/publicportal/sudoku/0011047.pdf) , which is used to solve how to link puzzle pieces by doubly linked nodes and stack and also solve the tiling puzzle problem. What I did in my algorithm is that I created a main interface by using Java’s JFrame. Then in the main interface, I separated the interface into four sub-interfaces (modules), which are user panel, tile board, main board, and result board. The main board is used to view whether a list of tiles can form a puzzle. The tile board is used to show the tiles that read from a file. The user panel board is used to interact with the user and the app. The user panel provide users with three options, which are rotation, reflection, and no symmetry. The user can see the number of solutions to a puzzle based on the options that they select. Then the result board will show the number of solutions, total look-up time, and the time that cost to find out the first solution.

Algorithm ideas: Like what’s discussed above, the app algorithm strictly follows the dancing link algorithm. Besides implementing parsing file, board, extract, space, and configure modules for the purpose of interfaces and interactions, the app implements a critical module, which is search module. The search module uses depth-first search, doubly linked list and a 2D matrix. The 2D matrix is used to identify columns and rows. Each column will have a special node, which is used to track the number of nodes in its column. When search starts, the algorithm will select one column and a row in that column, if that column does not contain any row, then it returns unsolvable. Otherwise, it will set a tile in that column and row, then go on searching next tile position in another row and column. If found, two tiles will be linked. Otherwise, if the position is not found, the link between two tiles breaks and will gives unsolvable message. The algorithm will keep searching until all positions are found. For rotation and reflection, the algorithm will increases steps to find out each tile position. The algorithm will allow the tile to rotate in each column and row in order to fit into that column and row. Then connect to previous tile by using doubly linked list. It will repeat the steps until one solution is found. Then start another search until all solutions are found. The algorithm is not deterministic. The space complexity is fixed since it is 2D matrix, which is O(nxm). However, the time complexity can be different since the number of tiles is unknown. They may or may not all fit into the matrix. It may have insufficient tiles for a given matrix. The time complexity is between O(n) and O(nxm).

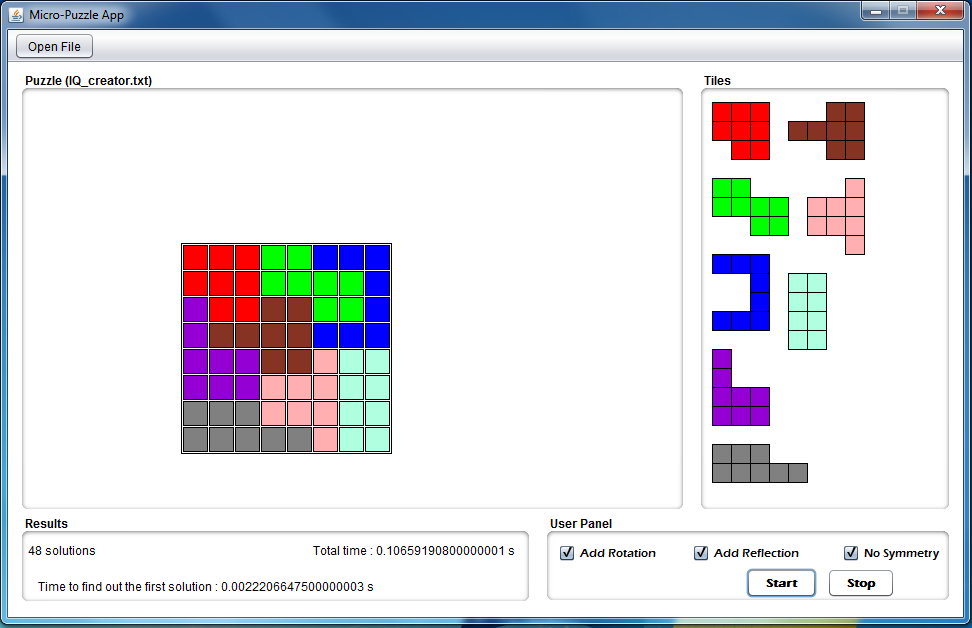
## A pointer to my app is : <https://www.dropbox.com/s/0jdus5naguc2z9f/puzzle.jar?dl=0>

CPU time

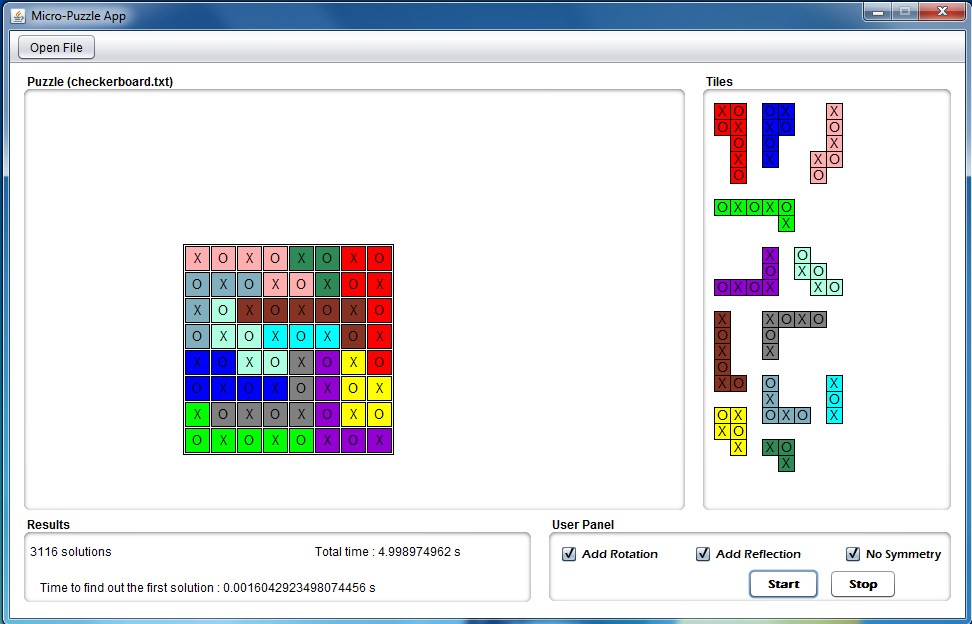
|  |  |  |  |
| --- | --- | --- | --- |
| Puzzle Name | Number of Solutions | CPU time for 1 solution(1st solution) | CPU time for all solutions |
| IQ\_creator | 48 | ~0.00222s | ~0.10659s |
| checkerboard | 3116 | ~0.00160s | ~4.99897s |
| lucky13 | None | N/A | N/A |
| Partial\_cross | 160 | ~0.00008s | ~0.12352s |
| Pentominoes3x20 | 8 | ~0.01592s | ~0.12737s |
| Pentominoes4x15 | 1472 | ~0.00194s | ~2.85881s |
| Pentominoes5x12 | 1472 | ~0.00196s | ~2.88700s |
| Pentominoes6x10 | 9356 | ~0.00138s | ~12.92485s |
| Pentominoes8x8\_corner\_missing | 10054 | ~0.00136s | ~13.70644s |
| Pentominoes8x8\_four\_missing\_corners | 17360 | ~0.00139s | ~24.09316s |
| Pentominoes8x8\_four\_missing\_diagonal | 296 | ~0.00155s | ~0.45910s |
| Pentominoes8x8\_four\_missing\_near\_middle | 168 | ~0.00245s | ~0.41185s |
| Pentominoes8x8\_four\_missing\_offset\_near\_corners | 216 | ~0.00956s | ~2.06589s |
| Pentominoes8x8\_four\_missing\_offset\_near\_middle | 504 | ~0.00132s | ~0.66456s |
| Pentominoes8x8\_four\_missing\_near\_corners | 1504 | ~0.00435s | ~6.54333s |
| Pentominoes8x8\_middle\_missing | 520 | ~0.00191s | ~0.99168s |
| Pentominoes8x8\_side\_missing | 2576 | ~0.00270s | ~6.97087s |
| Test1 | 8 | ~0.00287s | ~0.02296s |
| Test2 | None | N/A | N/A |
| Thirteen\_holes | 8 | ~0.00392s | ~0.03139s |
| trivial | 1 | ~0.00398s | ~0.00398s |

Output and Screenshots:

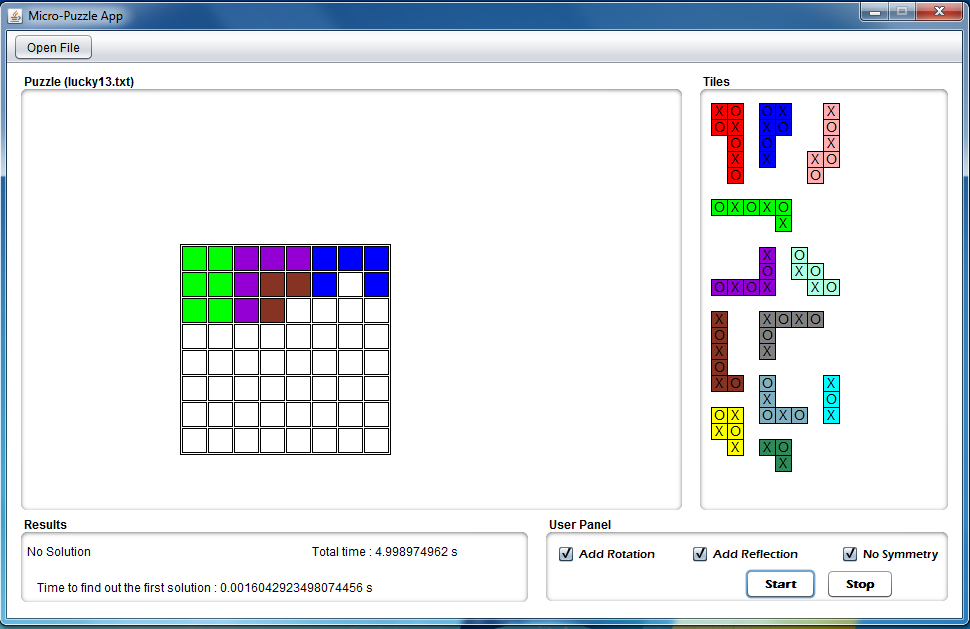
1. IQ\_Creator:



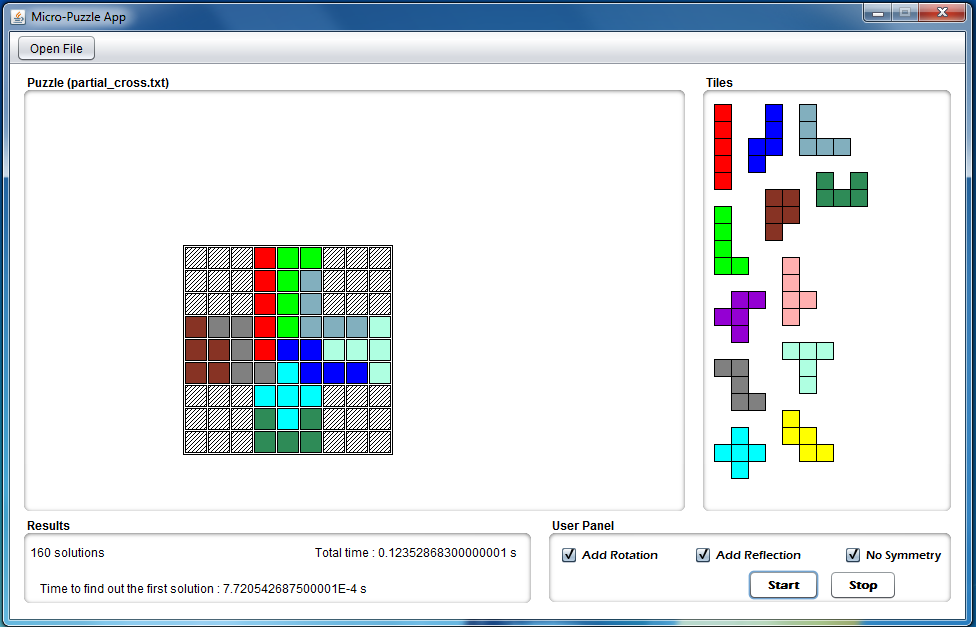
1. Checkerboard



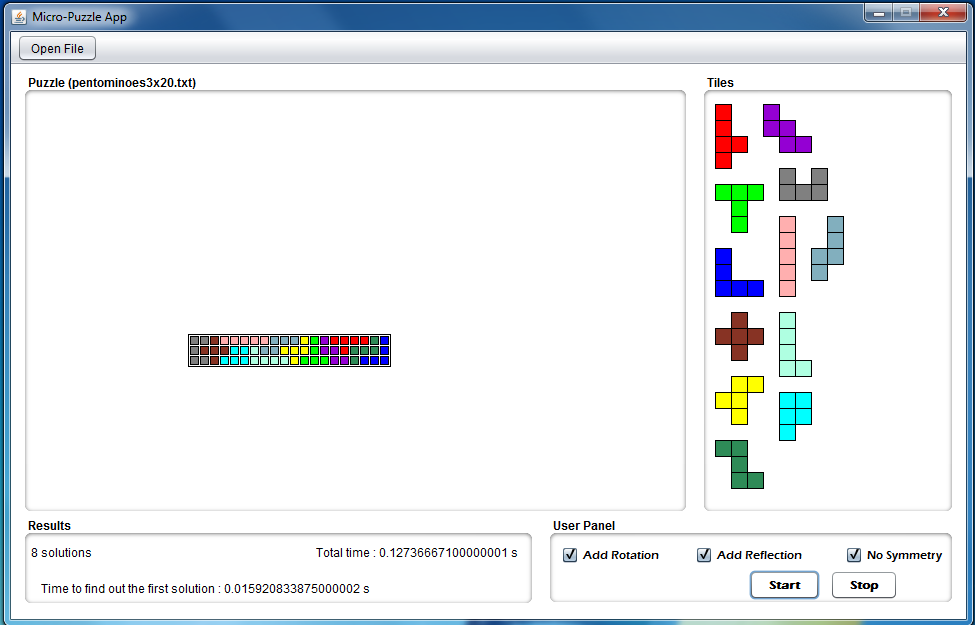
1. Lucky13



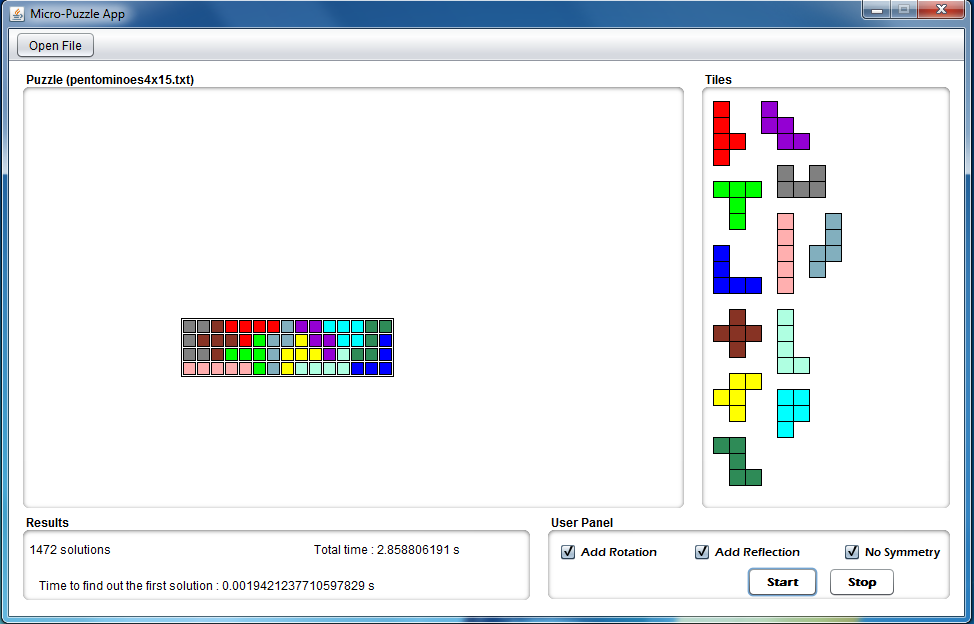
1. Partial-cross:



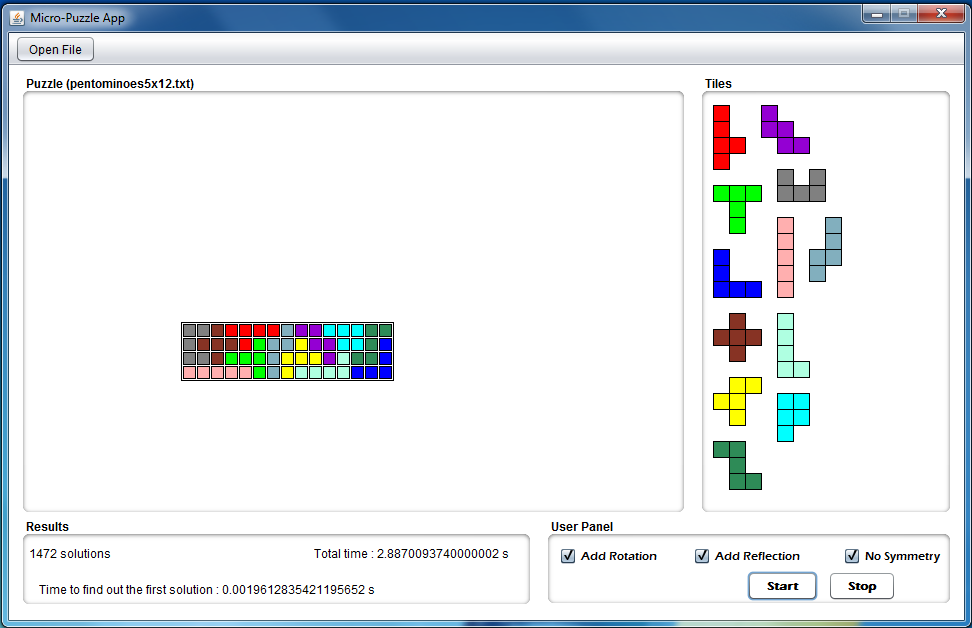
1. Pentomines3x20



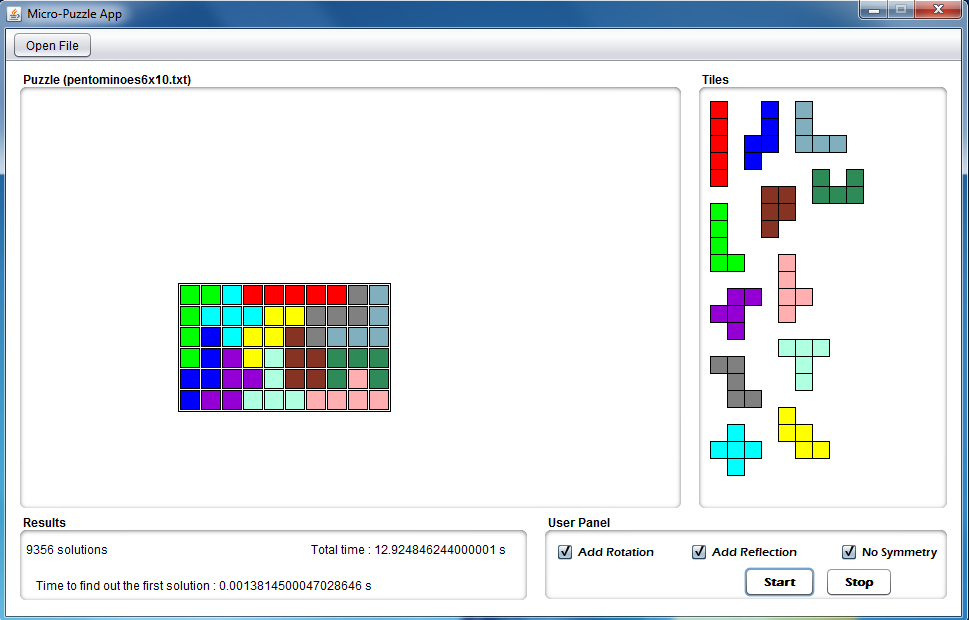
1. Pentomines4x15



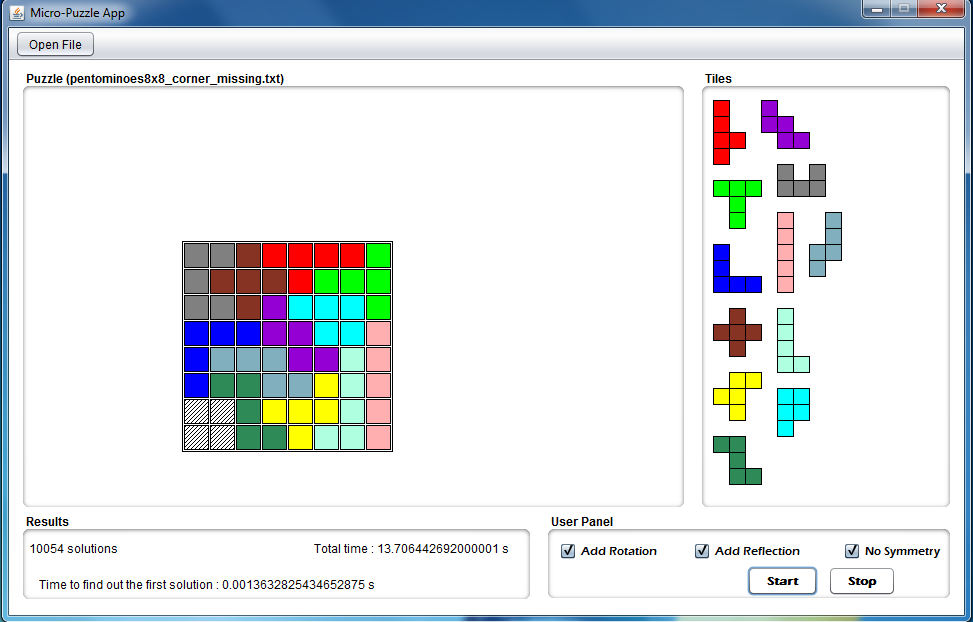
1. Pentomines5x12



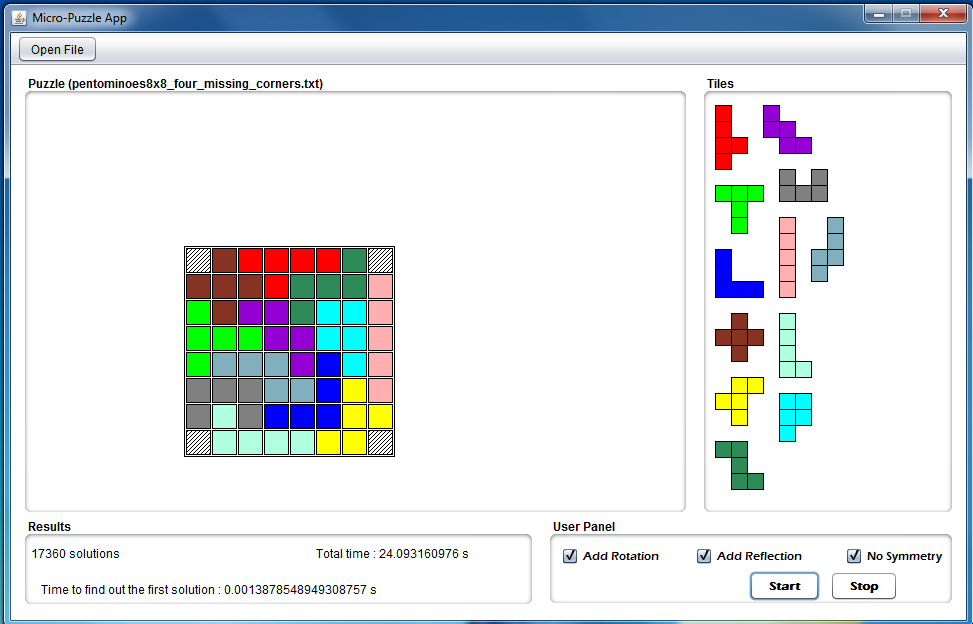
1. Pentomines6x10



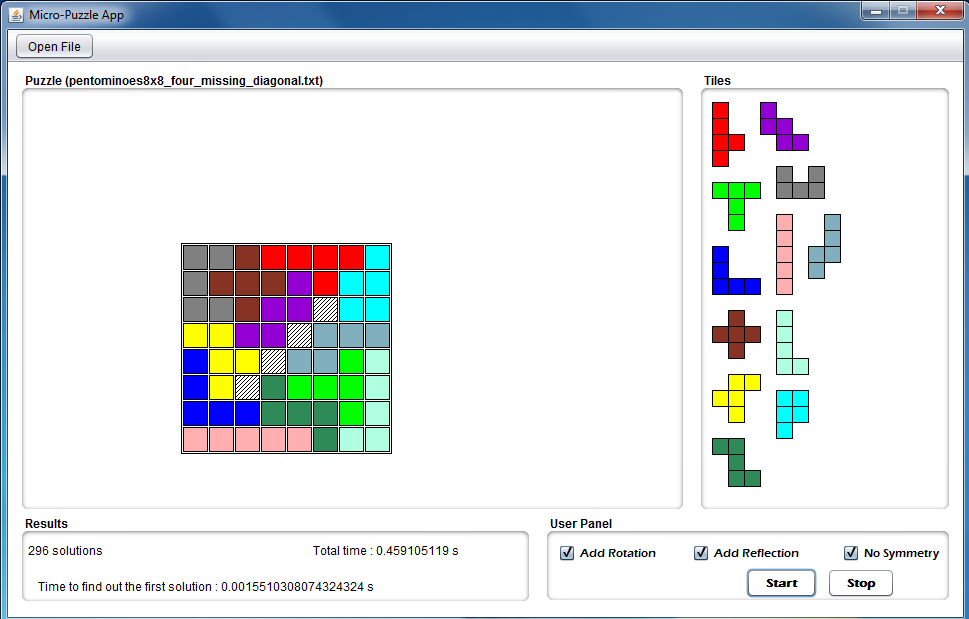
1. Pentomines8x8\_corner\_missing



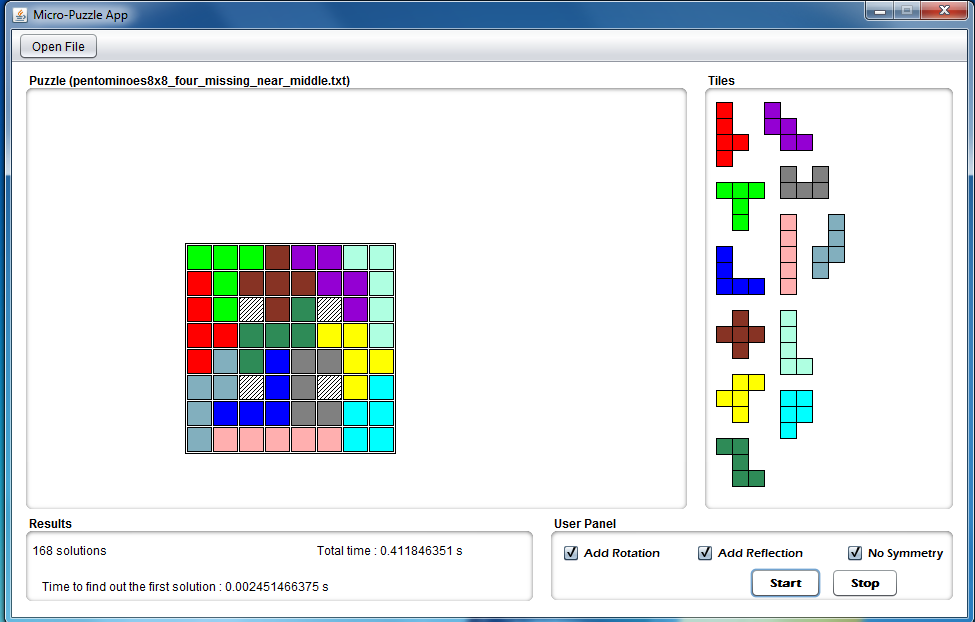
1. Pentomines8x8\_four\_missing\_corners



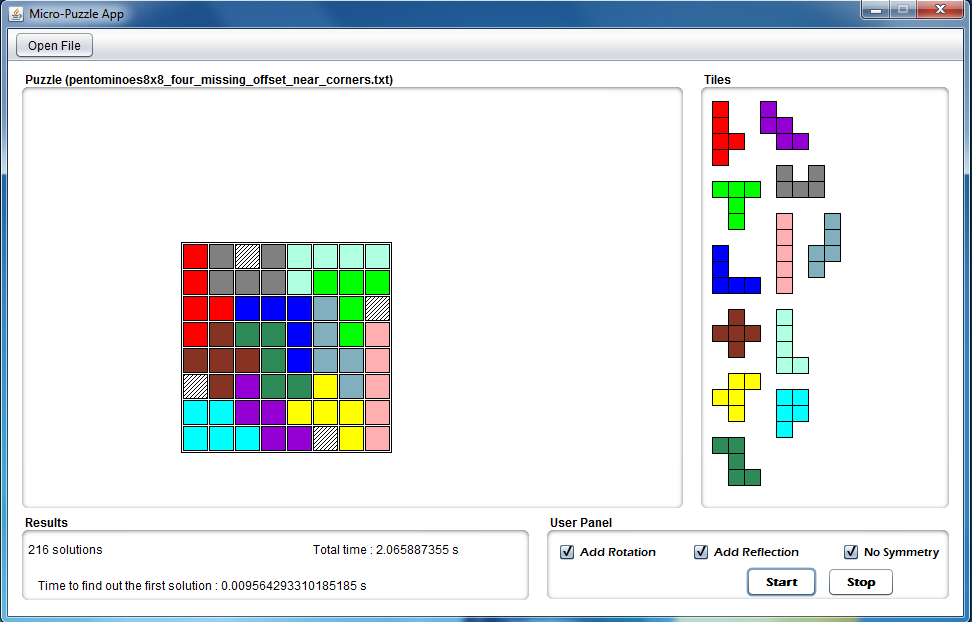
1. Pentomines8x8\_four\_missing\_diagonal



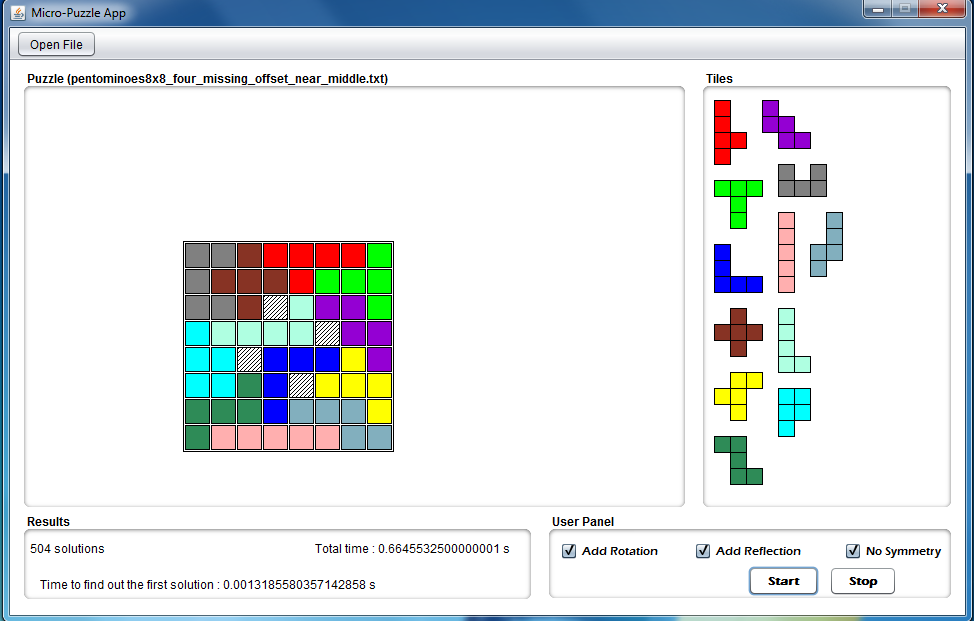
1. Pentomines8x8\_four\_near\_middle



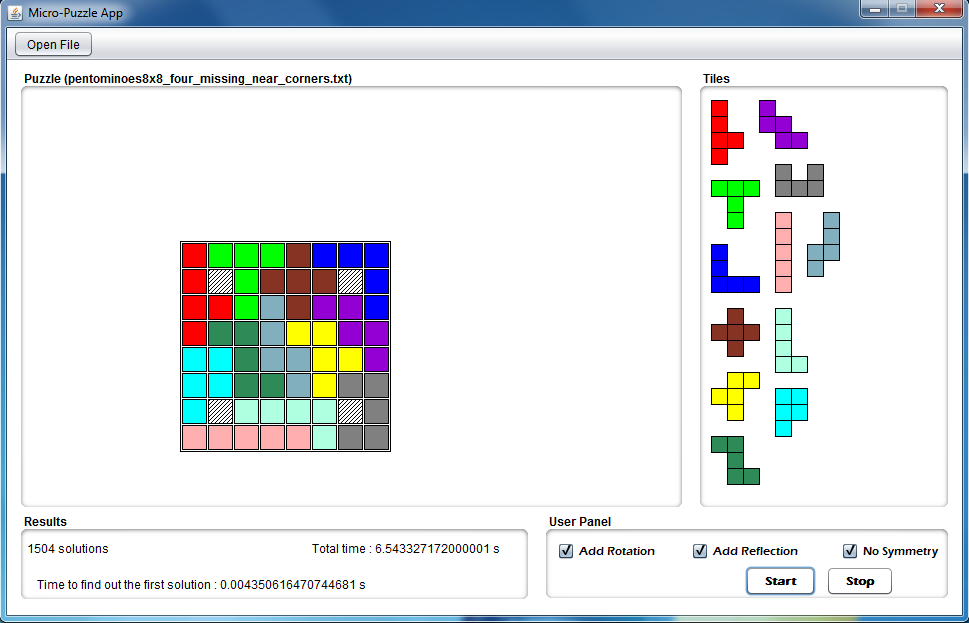
1. Pentomines8x8\_ missing\_offset\_near\_corners



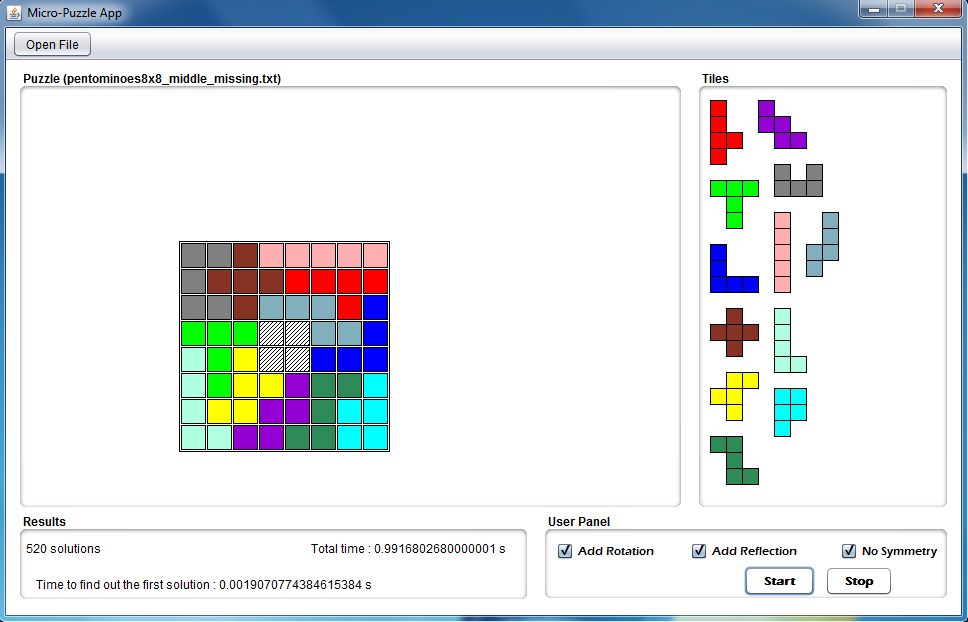
1. Pentomines8x8\_ missing\_offset\_near\_middle



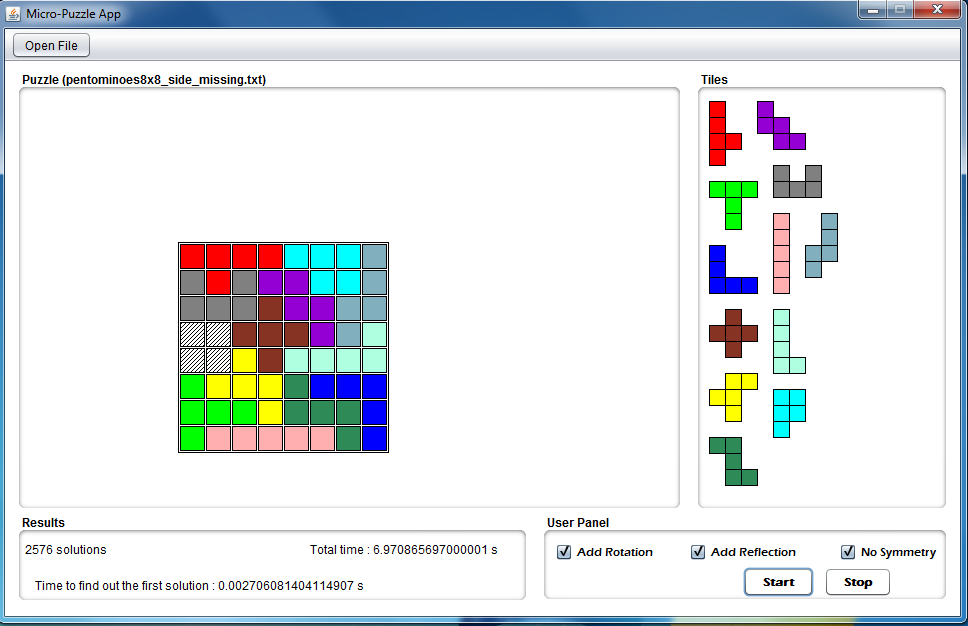
1. Pentomines8x8\_ four\_missing\_near\_corners



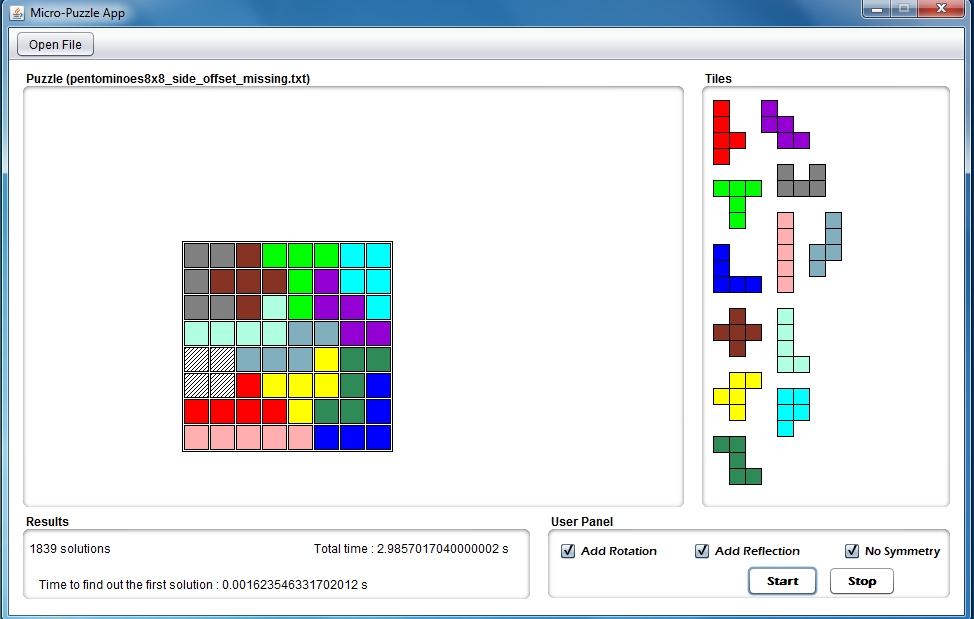
1. Pentomines8x8\_ middle\_missing



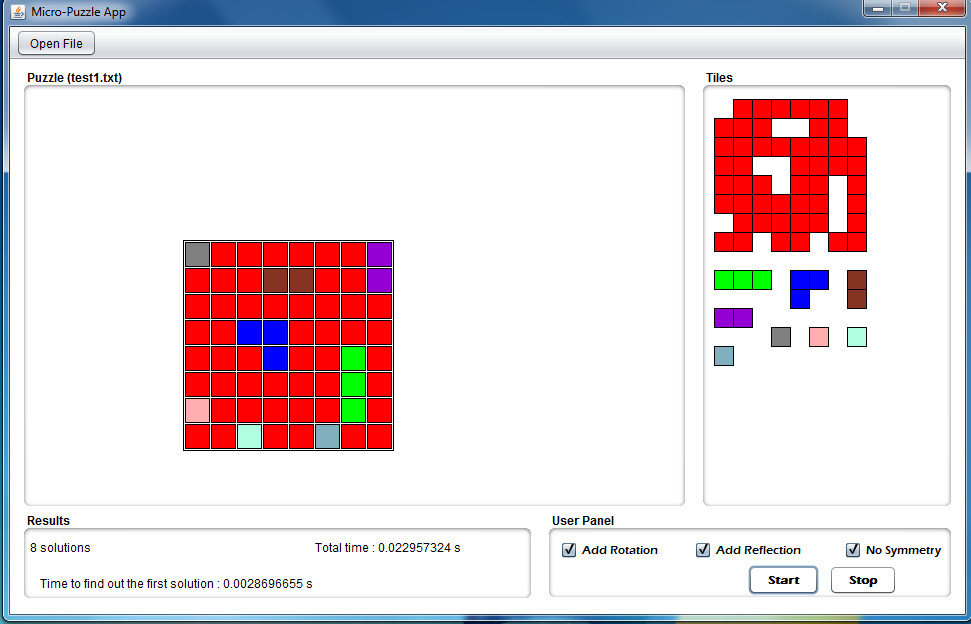
1. Pentomines8x8\_ side\_missing



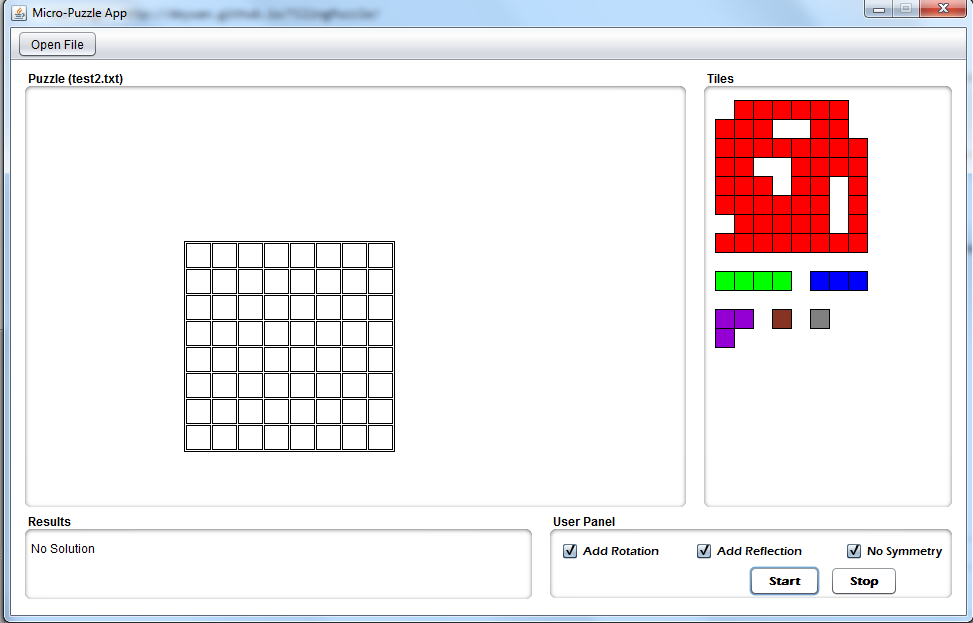
1. Pentomines8x8\_ side\_offset\_missing



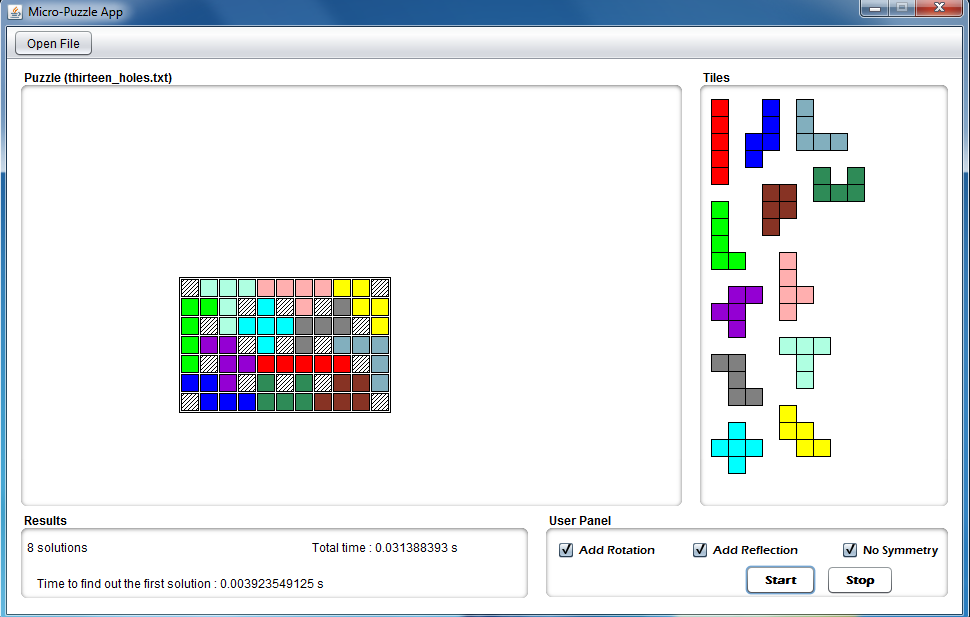
1. Test1



1. Test2



1. Thirteen\_holes



1. Trivial

