Tiling Puzzle Solver
University of Virginia

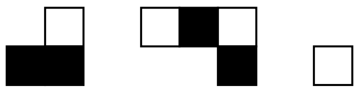
Gabriel Robins

"But what am I to do?" said Alice.

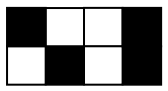
This term project entails implementing a program that will solve arbitrary tiling puzzles. The purpose of this project is to introduce a rich branch of discrete mathematics / combinatorics that involves tilings, symmetry, counting, solution space searching, branch-and-bound pruning, etc. Many theorems exist regarding tilings, and we will prove a few of these theorems in class.

Consider the following simple tiling puzzle (your program should of course be able to handle any similar but more complex puzzles):

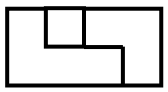
We are given N (N=3 in this example) two-dimensional multi-colored rectilinear tiles:



We are also given a colored target board configuration, such as:



Our goal is to completely tile the target configuration using the given tiles, without overlap. In this example, the solution to the tiling puzzle is:



Note that all shape and color constraints must be satisfied, and that in general there may be more than one solution (or no solutions at all). In the example above only two colors are used, but additional colors may occur in general.

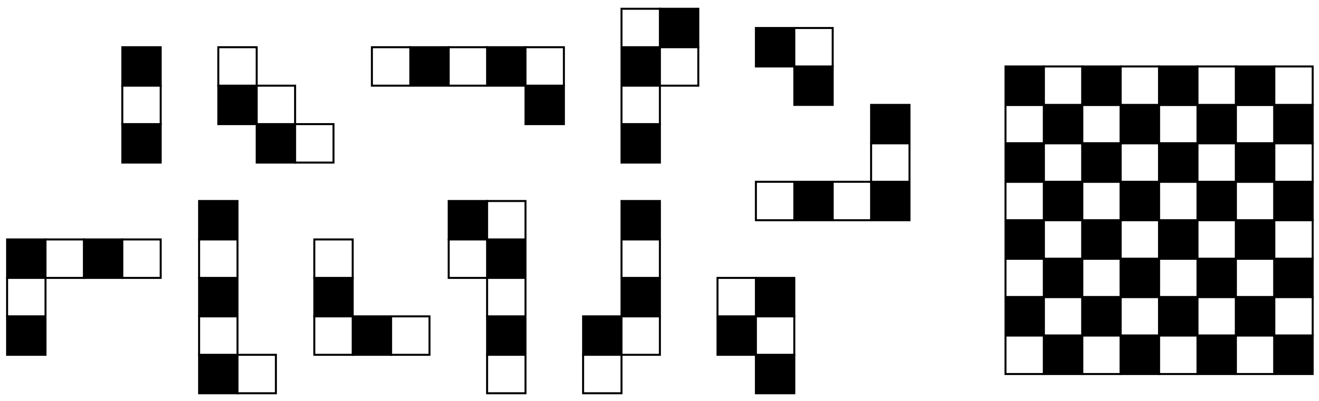
The puzzle is encoded in a single ASCII input file by using various characters to denote the shapes and colors of the tiles; tiles may be arbitrarily oriented/placed in the input file, and the target configuration is assumed to be the largest tile in that file. For example, the instance above may be encoded in the input file as:

```
......b...bab....abba
.....aa.....a....baba....b
```

(where the periods denote blank spaces, and the a's and b's denote the two colors of the tiles of this puzzle instance). An online input file containing the above instance may be found at:

www.cs.virginia.edu/~robins/puzzles/trivial

A less trivial instance of such a puzzle (tiles and target configuration/board) is as follows:



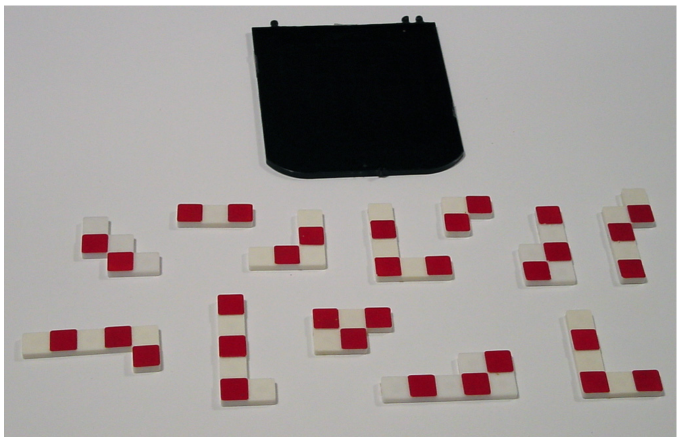
Note that there are color constraints in this puzzle as well as shape constraints. There are also several non-isomorphic solutions (i.e., not counting rotations and reflections of the entire configuration) to the puzzle instance above. This input file representing the puzzle instance above, contains the following data:

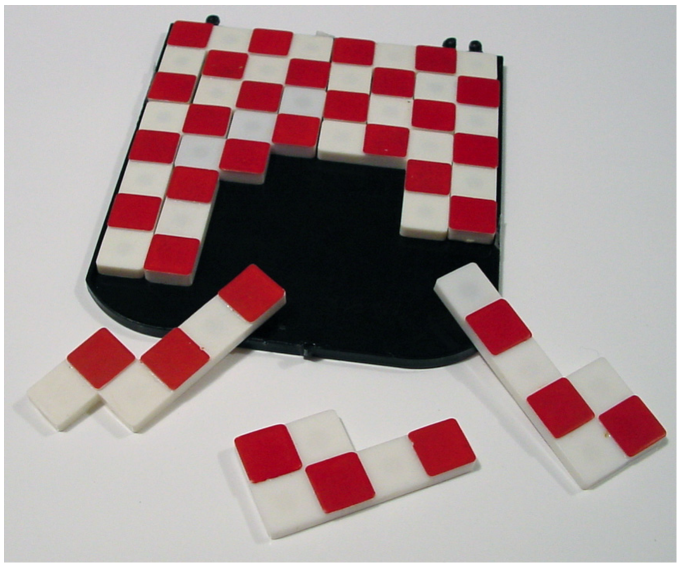
```
     O     OXOXO         OX
 X   XO        X  XO     XO          XOXOXOXO
 O    XO           X     O     X     OXOXOXOX
 X                       X     O     XOXOXOXO
       X   O     XO         OXOX     OXOXOXOX
XOXO   O   X     OX                  XOXOXOXO
O      X   OXO    O    X             OXOXOXOX
X      O          X    O    OX       XOXOXOXO
       XO         O    X    XO       OXOXOXOX
                      XO     X
                      O
```

Note that the tiles and board for each puzzle are located in the same file, with the largest "tile" assumed to be the board itself.

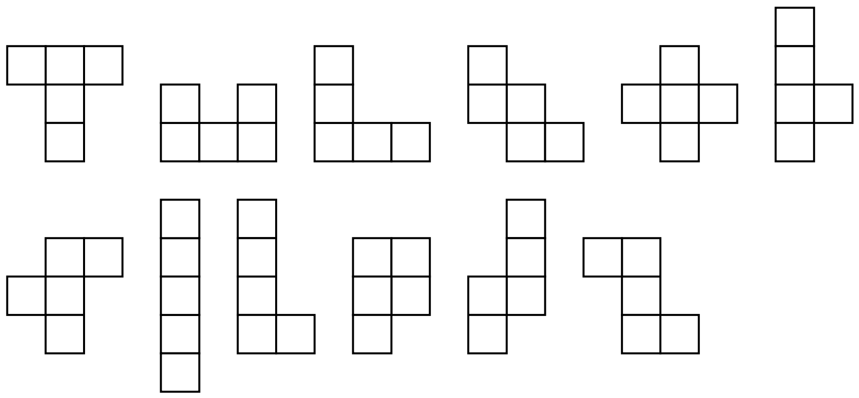
An online input file containing the above puzzle instance can be found at:

www.cs.virginia.edu/~robins/puzzles/checkerboard

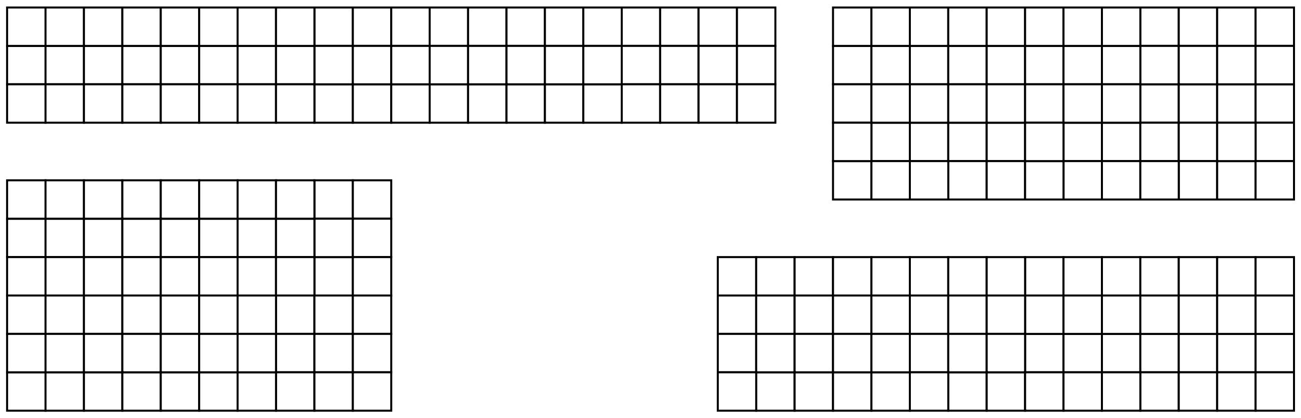




An interesting set of puzzles consists of pieces corresponding to the twelve "pentominoes" (i.e., all of the 12 non-isomorphic combinations of five connected unit squares).

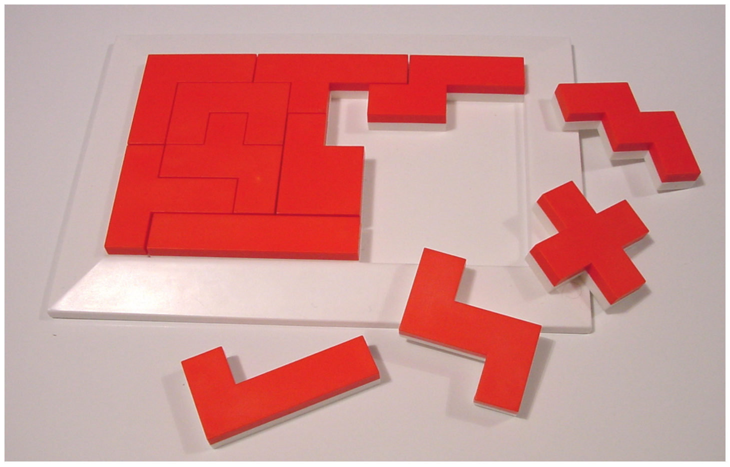


The above 12 pentominoes (collectively having a total area of 12*5=60 square units) can be fitted into a 60-square-unit rectangle of dimensions (a) 3x20, (b) 4x15, (c) 5x12, or (d) 6x10:

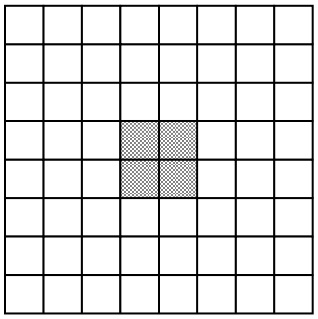
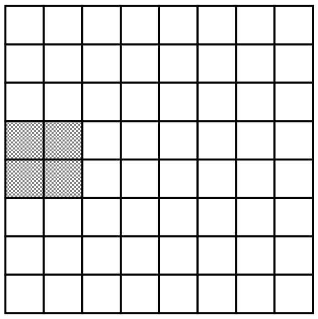
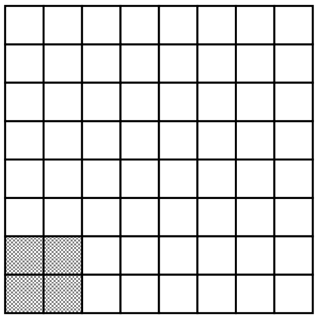
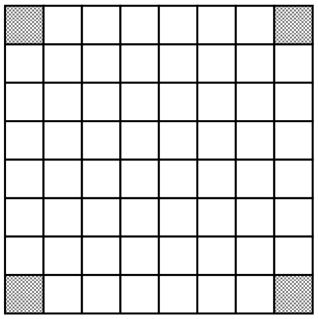
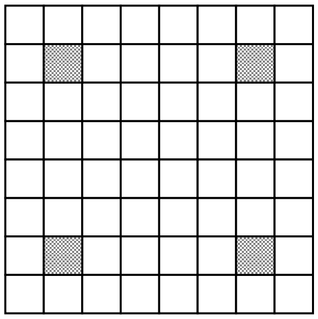
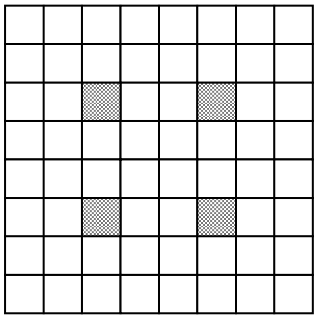
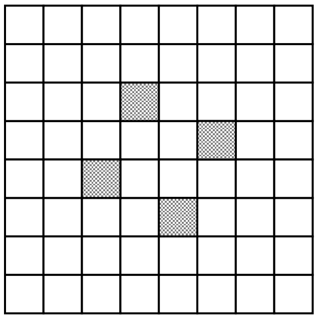
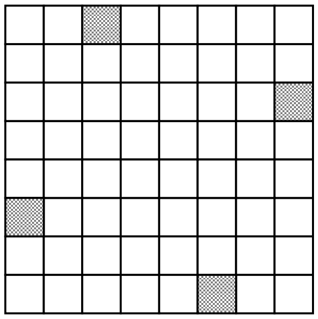
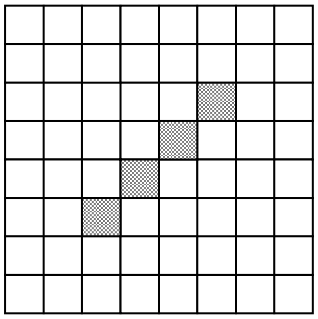


Online input files containing the above four puzzle instances can be found at:

www.cs.virginia.edu/~robins/puzzles/pentominoes3x20
www.cs.virginia.edu/~robins/puzzles/pentominoes4x15
www.cs.virginia.edu/~robins/puzzles/pentominoes5x12
www.cs.virginia.edu/~robins/puzzles/pentominoes6x10



The 12 pentominoes can also be fitted into each of the following nine 8x8 boards with 4 squares missing from each, as indicated:

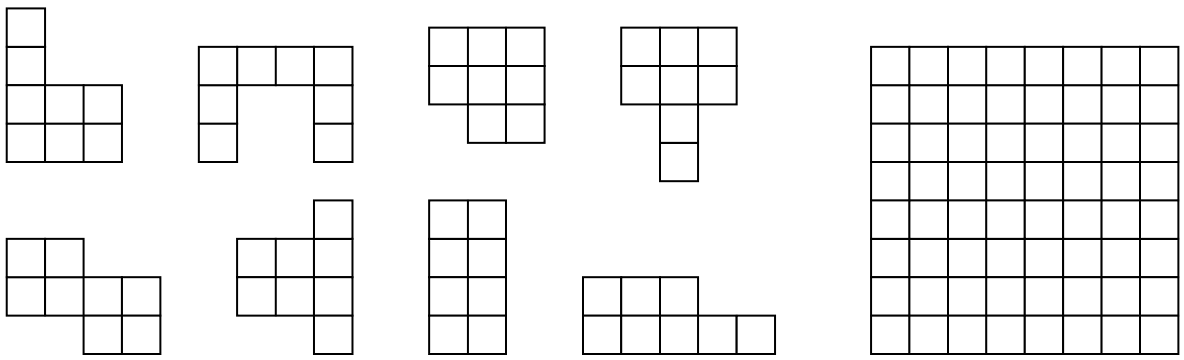










Online input files containing the above puzzle instances can be found at:

www.cs.virginia.edu/~robins/puzzles/pentominoes8x8_middle_missing
www.cs.virginia.edu/~robins/puzzles/pentominoes8x8_side_missing
www.cs.virginia.edu/~robins/puzzles/pentominoes8x8_corner_missing
www.cs.virginia.edu/~robins/puzzles/pentominoes8x8_four_missing_corners
www.cs.virginia.edu/~robins/puzzles/pentominoes8x8_four_missing_near_corners
www.cs.virginia.edu/~robins/puzzles/pentominoes8x8_four_missing_near_middle
www.cs.virginia.edu/~robins/puzzles/pentominoes8x8_four_missing_offset_near_middle
www.cs.virginia.edu/~robins/puzzles/pentominoes8x8_four_missing_offset_near_corners
www.cs.virginia.edu/~robins/puzzles/pentominoes8x8_four_missing_diagonal

All known combinations of four holes in an 8x8 board can be tiled with the twelve pentominoes (it is in fact unknown whether there exists any four-holed 8x8 board that is not thus tilable).

Yet another 8x8 tiling puzzle is as follows:

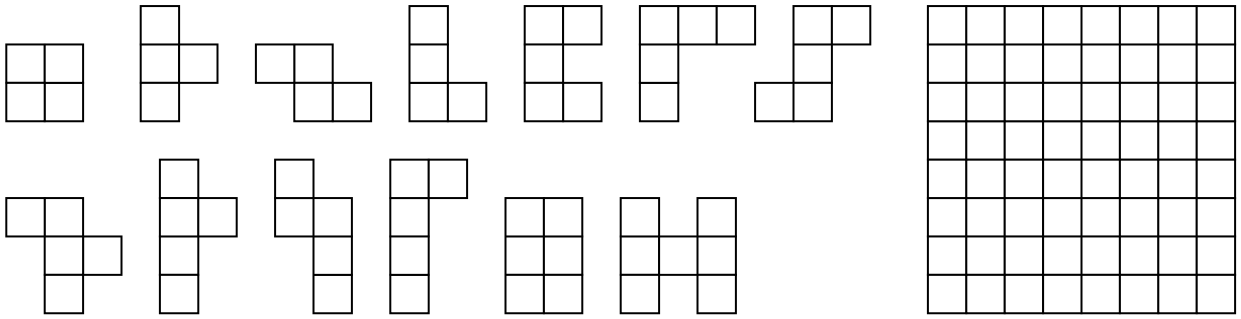


An online input file containing the above instance can be found at:

www.cs.virginia.edu/~robins/puzzles/IQ_creator

I.Q. CREATOR I
Place all 8 pieces into the box without causing any gaps. (See instructions)
Placez les 8 morceaux dans la boîte en ne laissant aucune ouverture.(Voir les instructions)

The following tiling puzzle is marketed by www.bitsandpieces.com under the name "Lucky Thirteen":

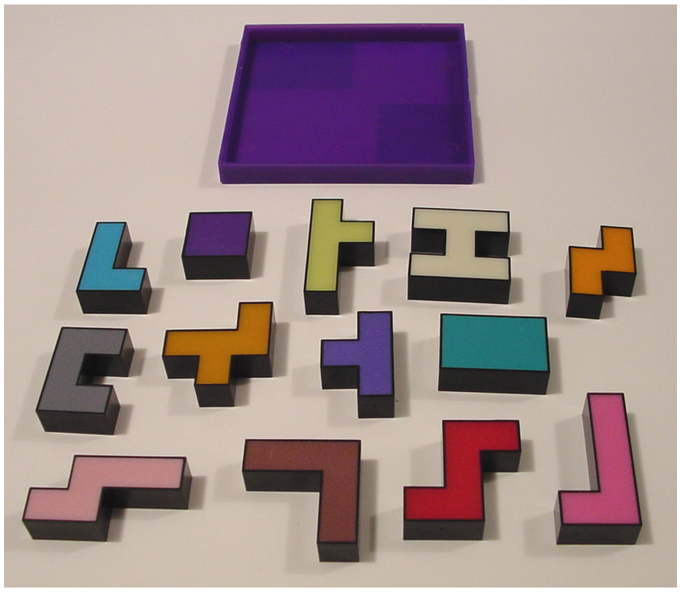


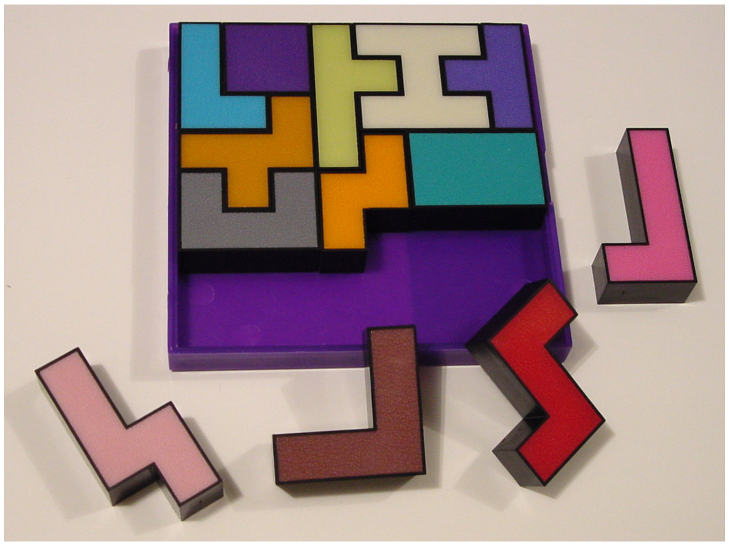
An online input file containing the above instance can be found at:

www.cs.virginia.edu/~robins/puzzles/lucky13

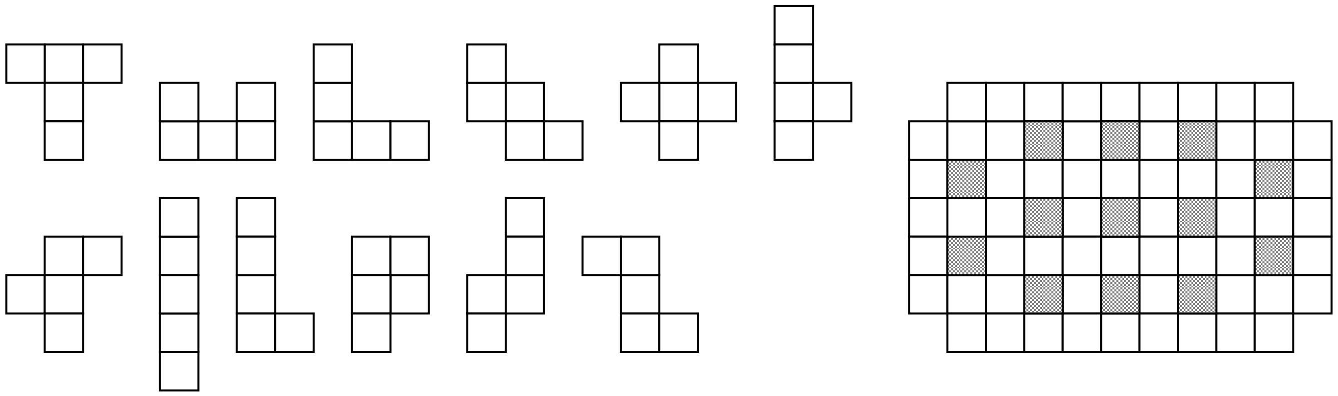
(This is one of the actual plastic puzzles that I passed around in class.)

Interestingly, "Lucky Thirteen" is also a three-dimensional puzzle, with each solid piece having a height of one unit, and all thirteen pieces can be assembled into a three-dimensional 4x4x4 cube. In addition, six of the thirteen pieces can be assembled into a three-dimensional 3x3x3 cube.





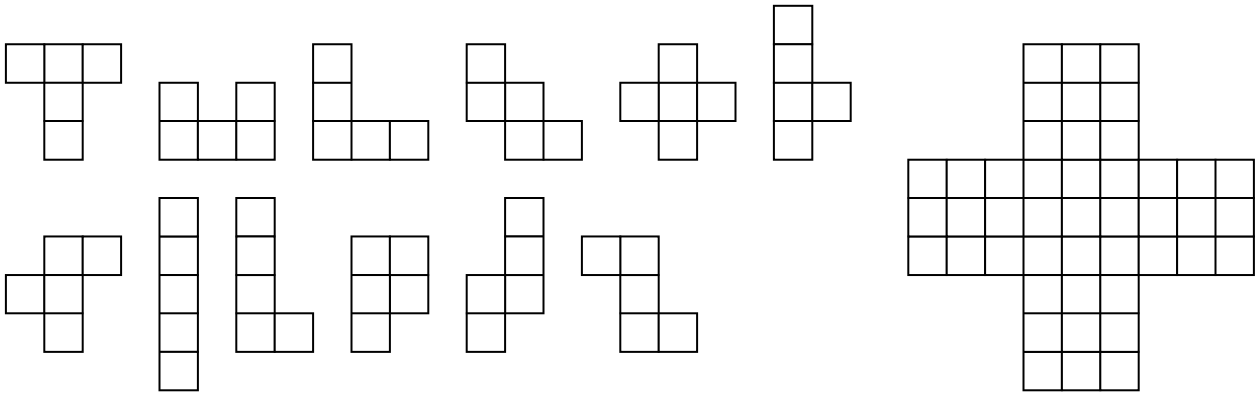
The 12 pentominoes can be fitted into the following irregular board with 13 holes in it.



An online input file containing the above instance can be found at:

www.cs.virginia.edu/~robins/puzzles/thirteen_holes

Sometimes not all the tiles are needed for a solution, as is the case here when using the set of 12 pentominoes to cover a cross-shaped board (only 9 of the 12 pentominoes suffice to cover the board in this example):



In cases such as this, your program should print a warning that not all the tiles are used, and proceed to solve the puzzle. An online input file containing the above instance can be found at:

www.cs.virginia.edu/~robins/puzzles/partial_cross

This instance is a special case of the "pentominoe triplication" problem, where for each pentominoe, we can use 9 of the <u>other</u> pentominoes to create a three-times-normal-size image of the original pentominoe.

Instructions

- This project is due before the final exam. However, please start on this project immediately. **<u>Warning: you will not be able to complete this project if you procrastinate on it until the last week of the semester!</u>**

- You may work either by yourself or in teams of two people (but <u>not</u> more than two). Both team members should contribute equally to the overall project effort.

- Document / comment your code well (each major function should be explained, etc.)

- Your program should be able to solve arbitrary puzzles (in particular, it should be able to solve all the puzzle instances given above). Please make up additional puzzles of similar style to exercise your program (and exchange puzzles with your classmates - see if your program can solve other people's puzzles without you being given their puzzle solutions).

- If no solution exists for an input puzzle, your program should report this.

- Please implement a reasonably friendly, graphical-based user interface, and display the solutions graphically, with lines & boxes on the screen, use different colors to denote different tiles, etc.

- Your program must be able to directly read the input ASCII file, and parse its contents in order to determine the puzzle specification (i.e., search the file for individual tiles, with the target/board configuration implicitly being the largest "tile" found; all tiles in the input file will be separated from each other by blanks).

- Make sure that your code finds and reports <u>all</u> the non-isomorphic solutions of a given instance (instead of finding only one of the solutions). Note that rotated / reflected versions of the same solution do not count as distinct from one another.

- Be prepared to demo your code in real-time, on puzzles that you have never seen before (we'll provide you with several such puzzles during your demo, each in its own file in the same format as the puzzles above). Your program should <u>never</u> crash, regardless of what the input is (if the input file is not a valid puzzle, your program should report this). The more robust your program is, the more credit it is worth, and conversely for programs that crash, give wrong solutions, fail to find existing solutions, difficult to use, etc.

- Do not copy code from any source (e.g., other people, the Web, etc.) - all code and algorithms should be your own. This project is pledged under the UVa honor code.

- You may use whatever programming language you choose (e.g., C, C++, Java, Visual Basic, Perl, Python, etc.), under your favorite combinations of OS (e.g., MS Windows, Unix, Linux, Solaris, etc.) and hardware platform (e.g., PC, Sun, Apple, SGI, etc.)

- Allow the user to optionally select a Boolean flag that allows all puzzle pieces to be reflected (i.e., flipped over) as well as rotated; note that activating this option may potentially increase the number of feasible solutions to any puzzle instance.

Suggestions

- First, don't be intimidated by this project - it is definitely doable by CS undergraduates!

- Approach this project with a stepwise refinement mindset: implement and debug simple self-contained modules separately (e.g., data file parser, graphical user interface, main search routine, symmetrical-equivalence detector, solutions cataloguer, display routines, etc.) Problems do get easier if you break them down into small, manageable pieces!

- When debugging, test your code with small puzzles, so you can easily track your code's behavior and fix any bugs. A graphical user interface will also be helpful in debugging, since you can then easily and quickly observe what the code is doing. So it would be helpful to debugging if you create the graphical user interface first.

- Although a basic version (i.e., a brute-force no-frills version) of this project can be implemented in only a few days, for some people this may take several weeks of effort, depending on their programming skills, and especially if they choose to implement some of the extra-credit suggestions (which I would highly recommend!). Above all else, please do not procrastinate on this project - you will not be able to complete it during the last week of the semester with all your other time pressures and school responsibilities piled on (and in order to receive credit for this project, your code must work, i.e., it must be able to actually solve puzzles).

- It will be instructive for you to obtain (or construct) some of these puzzles yourself, and play with them in order to develop intuitions on improved search strategies, etc. (trying to solve these puzzles manually also vividly illustrates how much faster a computer is than a human… ☺)

Ideas for Extra Credit

Extra credit will be given for each of the following implementation options (among others - be creative and impress me!):

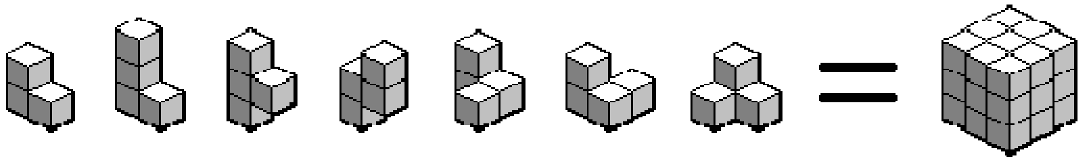
- Turn in your project early (for one extra point per each day earlier than the final deadline).

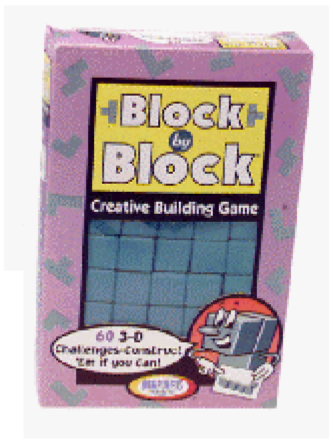
- Allow the user to set a flag which will graphically display the partial solution that is currently being examined (this option, when selected, may slow down the search speed considerably, but it will aid in debugging the code, and it will provide a visual status of the search progress).

- Allow the user to set a flag which will cause the code to "single-step" through the search space, indicating graphically which puzzle tile is currently being tried at what board position.

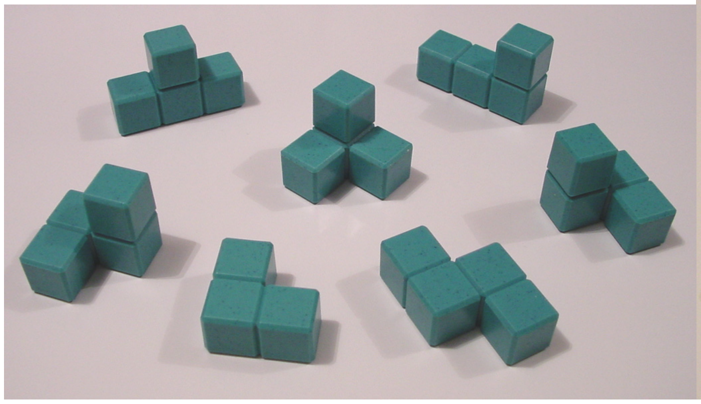
- Implement a very efficient search, which avoids dead-ends and non-feasible choices, using branch-and-bound techniques. A highly optimized program should be able to find a solution of each of the puzzles above within a few CPU seconds, and all solutions within a few minutes.

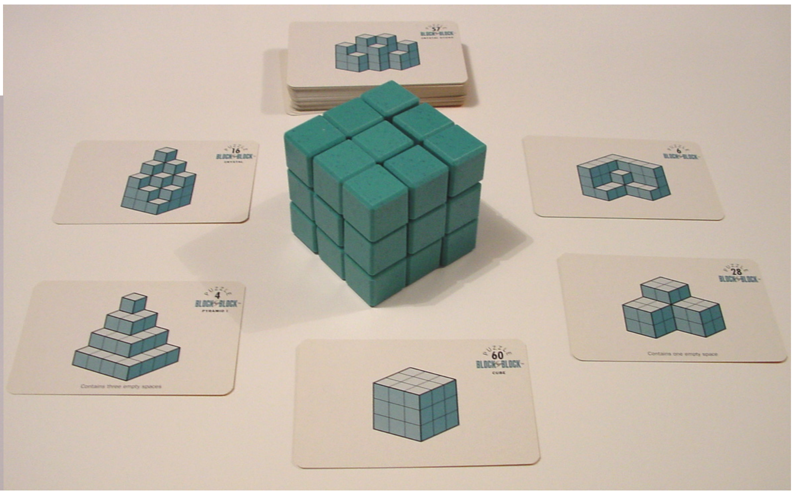
- Generalize your program and use interface to handle three-dimensional polyominoe puzzles, such as the Soma Cube (below), and 3D Pentominoes (see next page)



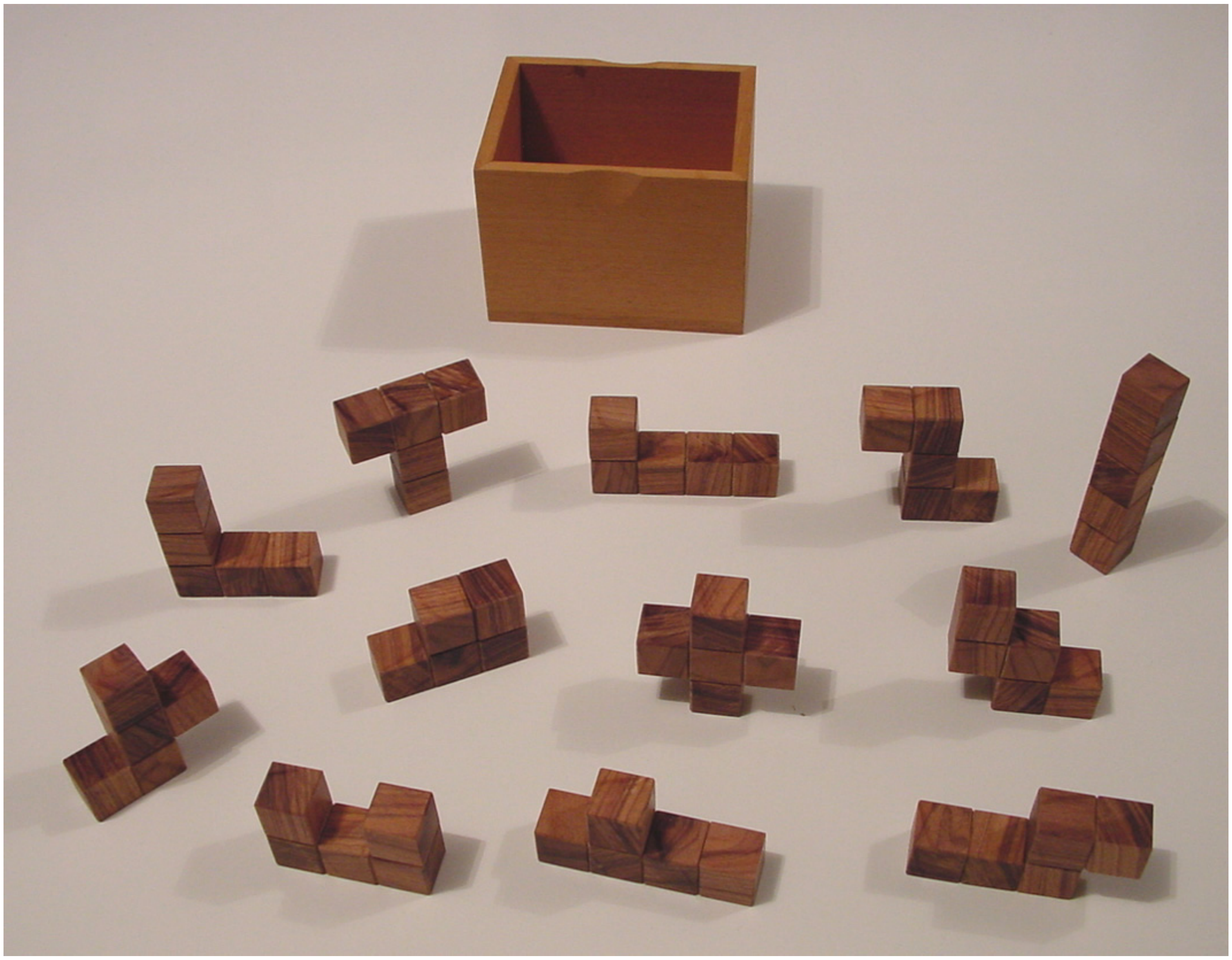


The Soma Cube was marketed by Think Fun (www.thinkfun.com) under the name "Block by Block", and available through Amazon.com: www.amazon.com/exec/obidos/ASIN/B00000IRTH/qid%3D999309229/104-5976693-6618339





The three dimensional (i.e., one unit high) version of the 12 pentominoes, can be packed tightly into a 3x4x5 box.



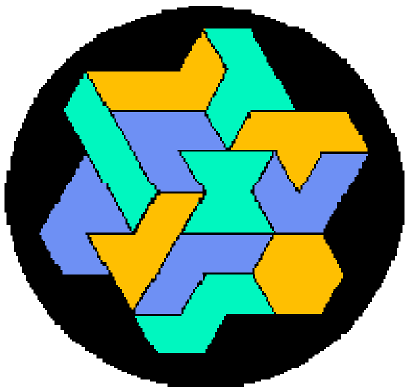
- Implement a puzzle-editor, i.e., allow the user to interactively specify a puzzle using the mouse and click/point/drag operations, and allow the user to save such newly-created puzzles into files for future use.

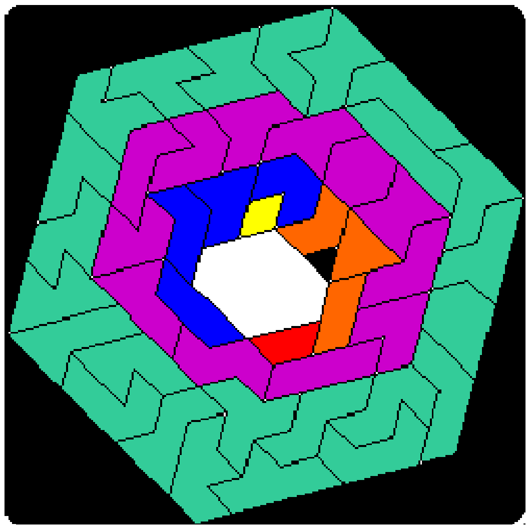
- Implement a random puzzle generator, which given a user-specified MxN rectangle, and a number of tiles K (specified though say, type-in data fields in the graphical user interface), generates a random feasible (i.e., solvable with at least one solution) MxN rectangular puzzle with K tiles. The user should then be able to ask the program to find and report all solutions.

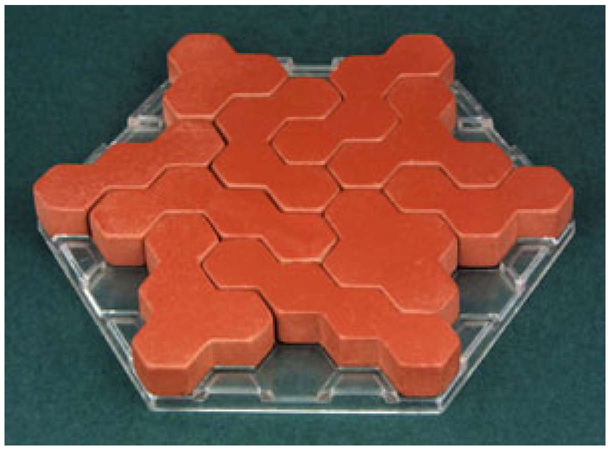
- Implement an "interactive manual mode" which will allow the user to solve a puzzle manually by dragging and dropping pieces onto a board. The program will enforce the puzzle constraints (i.e., that every user move is valid, without pieces overlapping, etc.) and time how long the user took to solve the puzzle (both in terms of elapsed time and the number of "moves" or tries). This "interactive manual mode" will work well with the random puzzle generator!

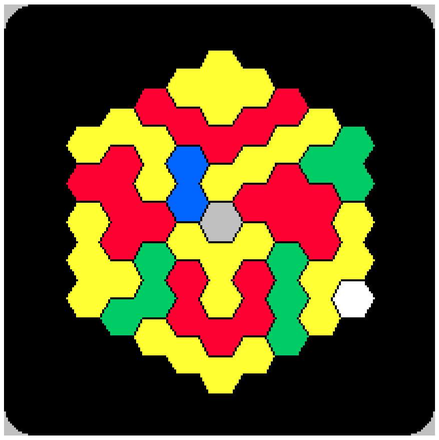
- Generalize your program to handle triangle and hexagon -based polyominoes (the main difficulty here is resolving the representation mismatch between the non-rectilinear tile/board vs. the rectilinear arrays). For example of non-rectilinear puzzles on the market, please see:

  www.gamepuzzles.com/esspoly.htm









What to Submit

Please turn in the following materials:

1. A discussion of what was implemented and how it works (i.e., an explanation of your algorithms, solution scheme, search strategies, optimizations, etc.);

2. Some screen-shots, showing snapshots of your program's execution;

3. A complete hardcopy listing of your code and documentation;

4. A pointer to the on-line ready-to-run code (or a diskette/CD with same);

5. Output from runs on the input puzzles given above, and their solution(s). In particular, your program should at a minimum be able to solve all the puzzles at:

   www.cs.virginia.edu/~robins/puzzles

   You should also make up new non-trivial puzzles for your program to solve. Be creative!

   For each puzzle, if there are a small number of solutions (i.e. dozens or less), please report them all; if there are a large number of solutions (hundreds or more), just list the first few dozens (but your code should still find all of them and report how many there are).

6. For each puzzle, please report the CPU run times that your code required to find the <u>first</u> solution, as well as the CPU time your program took to find <u>all</u> solutions; you can put all this information in one table, as follows:

Puzzle name	Number of solutions	CPU time for 1 solution	CPU time for all solution

7. An actual real-time demonstration of your project running and solving puzzles. Please arrange with the TA an appointment to demo your project.

"Is that all?" Alice timidly asked.

Notes and Pointers to the World of Puzzles

Physical plastic/wood versions of puzzles like the ones above may be purchased on the Web; examples include:

- The authoritative book on tilings is "Polyominoes" by Solomon Golomb, the inventor of pentominoes in the early 1950's; this book contains numerous tiling problems, solutions, and tiling-related theorems; it is available at www.amazon.com at:

  www.amazon.com/exec/obidos/ASIN/0691024448/qid%3D999160313/sr%3D2-1/104-5976693-6618339

- "Lucky Thirteen" puzzle, was marketed by www.bitsandpieces.com

- Quintachex, Poly-5, Quintillions, and Sextillions marketed by Kadon Enterprises (http://www.gamepuzzles.com/) at:

  www.gamepuzzles.com/polycub3.htm#QX
  www.gamepuzzles.com/polycub2.htm#P5
  www.gamepuzzles.com/polycub2.htm#SX

- The Soma Cube is marketed by Think Fun (www.thinkfun.com) under the name "Block by Block", and is available for sale through Amazon.com:

  www.amazon.com/exec/obidos/ASIN/B00000IRTH/qid%3D999309229/104-5976693-6618339

- A comprehensive encyclopedia on puzzles may be found at:

  www.johnrausch.com/PuzzlingWorld/default.htm

- Pentominoes and other tilings can be used as the basis of sliding-type puzzles. For examples of interactive Java-applet polyominoe-type sliding puzzles, please see:

  www.johnrausch.com/SlidingBlockPuzzles

- There are international puzzle-design competitions, such as:

  www.johnrausch.com/DesignCompetition/default.htm

- There are some puzzle museums and exhibits around the world, e.g.:

  www.tutka.net/~linkola/exhibition.html

- Additional good Web-based puzzles and games vendors include the following:

  www.puzzles.cleversoul.com

  www.seriouspuzzles.com

  www.bitsandpieces.com

  www.gamepuzzles.com

  www.mefferts.com

  www.custompuzzlecraft.com

  www.mindwareonline.com

  www.thinkfun.com

  www.spilsbury.com

  www.puzzles.ca

  www.turnoffthetv.com

  www.puzzlesdownunder.com.au

  www.areyougame.com

  www.unclesgames.com

  www.mrpuzzle.com.au

  www.puzzle-factory.com

  www.spacecubes.com

  www.constructiontoys.com

  www.blueorangegames.com

  www.chessusa.com