ITSA Programming Challenge 2005

Joseph Cordina, Gordon Pace, Sandro Spina

July 2005

Abstract

This document outlines the problem that has to be solved in the programming competition held by the Department of Computer Science and A.I. The competition will open on Friday, 22nd July 2005 at 20:00 and closes on Sunday, 24th July 2005 at 20:00 ZST[1]. This document also specifies the criteria that will be used to judge the entries. The full rules of the Programming Challenge can be found on the website `http://www.cs.um.edu.mt/~itsapc05`.

1 Judging Panel

The judging panel is made up of the following three members:

- Joseph Cordina <`joseph.cordina@um.edu.mt`>
- Gordon Pace <`gordon.pace@um.edu.mt`>
- Sandro Spina <`sandro.spina@um.edu.mt`>

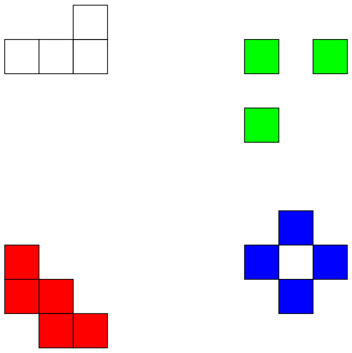
Any queries to the panel have to be directed to all three members making use of the mailing list address <`itsapc05support@cs.um.edu.mt`>. The role of the panel is to judge all submissions made to this competition and declare the winners. The judging panel's decision is final.

[1]Zeus Standard Time — the time as set on the UNIX server zeus. The time can be seen on the Programming Challenge website.

2 The Problem

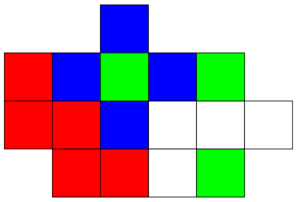
2.1 Problem Definition

Your task is to write a program which given a collection of tetris-like pieces made up of unit (1×1) squares, packs them into as small an area as it can without overlapping pieces. The pieces can be rotated (by 90, 180 or 270 degrees) and moved around. Below is an example — a collection of four pieces:



Pieces may *not* be flipped (mirror image) to pack in a smaller area. To complicate things, pieces may consist of disjoint squares (the green piece made up of three squares is one such piece).

One possible packing of the pieces given above is:



Note that pieces can only be moved around an exact number of units (eg you cannot move a piece half a square to the right).

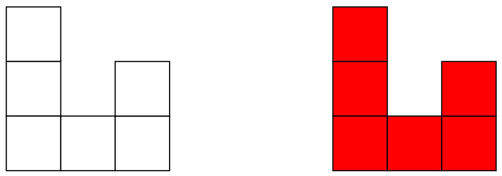
What makes a good packing, or layout? The measure that will be used to measure the quality of a layout is the area of the smallest rectangle (with sides parallel to the x and y axes) than encloses the pieces as laid out. In the above example, the given solution can be enclosed in a 6×4 rectangle. The programs will be tested on a number of collections of pieces and the ones packing the pieces into the smallest area will get more points. We have secondary measures to resolve a tie, but more about that later.

The problem is known to be a computationally intensive one, so you will have to use heuristics and rules of thumb to ensure that the program works in reasonable time. Programs not returning a result in 5 minutes will be considered to have failed to produce a valid layout.

2.2 Input and Output File Syntax

Your program will be given a file containing a number of named, pieces each made up of unit squares at given cartesian coordinates. For example the following input file consists of two identical pieces:

```
APiece:(0,0)(1,0)(2,0)(0,1)(2,1)(0,2)
AnotherPiece:(0,0)(1,0)(2,0)(0,1)(2,1)(0,2)
```

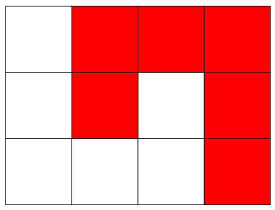


Note that each line is made up of an identifier (you can assume that it is a sequence of alphabetic characters possibly followed by a number of digits), followed by a colon (:), followed by a sequence of integer[2] coordinates. Each piece in the input file will start at coordinate $(0, 0)$ and will have a unique identifier. Note that piece identifiers are case sensitive. You can assume that the input file will always have at least one piece, and that each piece will consist of at least one square.

Your program should be able to parse such a file and produce another file with similar syntax, but in which the pieces will be moved and rotated so as not to overlap. An effective way of putting the above pieces together fitting into a 6×4 rectangle is as follows:

```
AnotherPiece:(1,1)(1,2)(2,2)(3,2)(3,1)(3,0)
APiece:(0,0)(1,0)(2,0)(0,1)(2,1)(0,2)
```

[2]You can assume that all given integers will fit in a normal signed 16 bit word. Your placement coordinates must also satisfy this constraint.



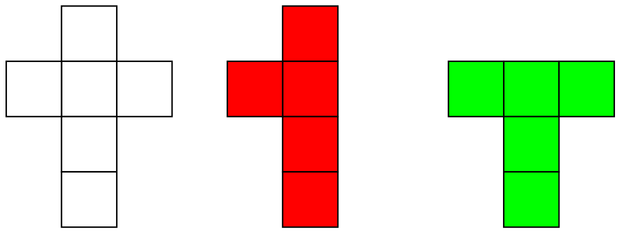
A valid output file satisfies the following specification:

- All the pieces appearing in the input file appear exactly once in the output file.
- Pieces can only be moved as a whole (keeping integer coordinates) and/or rotated (by 90, 180 or 270 degrees) from how they appear in the input file.
- No pieces in the output file overlap.

Note that the order in which the pieces appear in the output file does not need to be the same as the order of the pieces in the input file. Similarly, any ordering of the coordinates of an individual piece is acceptable.

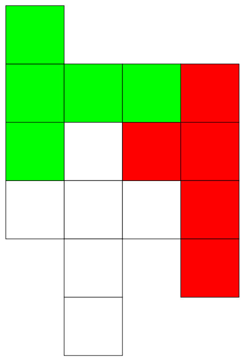
2.3 Yet Another Example

Below is yet another example with three pieces:



```
Cross:(0,0)(-1,0)(1,0)(0,1)(0,-1)(0,-2)
Hook:(0,0)(0,1)(0,2)(-1,2)(0,3)
Tee:(0,0)(1,0)(2,0)(1,-1)(1,-2)
```

One possible layout for these pieces with an area of 24 (6×4) is the following:



```
Hook:(1,0)(2,0)(2,1)(2,-1)(2,-2)
Cross:(0,0)(0,-1)(0,-2)(0,-3)(-1,-1)(1,-1)
Tee:(-1,0)(-1,1)(0,1)(1,1)(-1,2)
```

3 Submission Rules

Any submissions have to be Microsoft Windows executables. The executable has to be packaged inside a zip file. The executable has to be called *tiles.exe* and will take two parameters. It will be executed with the following command line parameters:

```
tiles <input file> <output file>
```

The program will be executed on an Intel-based PC running at 2GHz and having 1GB of RAM. The operating system will be Microsoft Windows XP and it will also have the .NET Framework 1.1 installed. Any submissions that fail to execute on the mentioned environment or any submission that take longer than 5 minutes to output a result will not be considered by the judging team.

Each registered team is allowed to submit as many solutions as it wishes. The last submission on Saturday, 23rd of July at 8:00pm will be considered for the *lightning* prize. The last submission on Sunday, 24th July at 8:00pm will be considered for the *standard submission* prize. After these times no corrections or re-submissions will be allowed.

4 Judging Criteria

The judging panel will take all submissions and run them on a number of pre-defined test problems. For each problem, the programs will be ranked according to the size of the smallest bounding rectangle (the smaller the better) and given a score according to the ranking: best (smallest) 50 points, second 30 points, third 20 points, fourth 10 points and fifth 5 points. The rest will not be awarded any points. In the case of ties on an individual problem, all submissions with the same area will get the same ranking, and hence score.

The winner will be the team that obtains the largest total number of points. In the case of an overall tie, teams will be ranked and scored according to the size of the smallest size of the rectangle (again, the smallest wins) and given the points as explained.

In the unlikely case of unresolved ties, the ranking will be done according to the execution time of the program (fastest program wins) after which the judging panel reserves the right to resolve any further tie in as fair a way as possible using a tie breaker.

5 Sample Problems and Other Resources Provided

Testing programs: Two programs will be provided to help you test out your programs. The first takes an input pieces file and an output layout file and verify their correctness. The program also calculates the size of the smallest rectangle that can contain all the pieces given. It can be executed on any Windows operating system using the following command line:

```
tiler <input file> <solution file>
```

The program assumes that the input file is a valid one with all pieces starting at position $(0, 0)$. Any problems with the files' syntax and layout (overlapping pieces, missing pieces) will also be output.

Another program will be provided that is able to generate a Postscript file from the solution file for visualization. This can be executed as follows:

```
vis <solution file> <output postscript file>
```

Note that `vis` checks for a valid layout, and will thus not work on input files which always include overlaps (since all pieces in the input file start at $(0, 0)$).

Typical problems: The judging team will be providing on the website a number of sample problems and possible layouts. The aim of these problems is to enable you to try your submission on typical problems.

The judging suite: To ensure fairness, the problems that will be used for judging the submissions will also be made available when the challenge opens. These will be encrypted and at the end of the competition, the key will be given to ensure that the files were unchanged.

6 Clarifications

During the period between Friday, 22nd July 2005 at 20:00 to Sunday, 24th July 2005 at 20:00 all the teams can make queries for clarification to the judging panel on the mailing address `itsapc05support@cs.um.edu.mt>`. If the judging panel deems the query to be valid, the answer will be posted on the website and an e-mail will be sent to all participating teams. The judging panel will do its best to answer these queries in as short a time as possible.

Good luck!