Project #3 – Hexomino 3-Color Covering

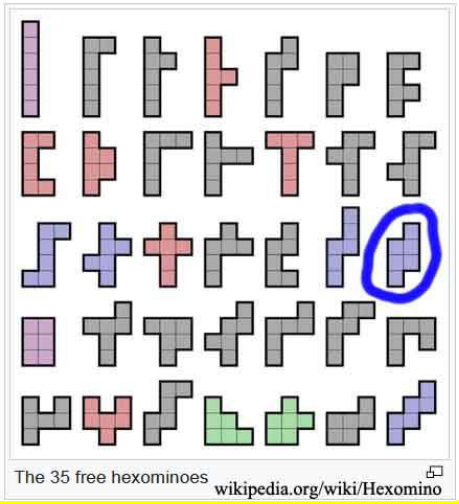
Introduction

This project is show the progress in attempting to find a pattern to fit the 35 hexomino tiles (plus one duplicated tile) into a 15x15 grid "board" (with a 3x3 center hole cut out) where they don't overlap, and they form a 3-colored map. The tiles can't overlap, or protrude beyond the boundary or into the center hole. (You'll likely want to do State-Space search, or DFS – maybe not BFS so much.)

Hexominoes

Hexominoes are like dominoes, but where a domino is two squares joined on a side (and hence there is only 1 unlabeled domino shape), a hexomino is six squares joined by their sides (and joined along "grid" lines). For example: a 1x6 tile, a 2x3 file, a 2x2 block with two "ear" squares joined at a corner. CF the web for more info on hexominoes (eg, Wikipedia).

(You need to use 2 of the tile circled in blue.)



The 35 free hexominoes wikipedia.org/wiki/Hexomino

(Try to start in upper-left corner & say run DFS and build out from the first tile you put there. Orientation (& flip), to place just the first tile. You can't leave unfillable holes, so that helps finding an early deadend. Can't overlap cells of two different tiles. Build-out = a solid growing tiled area border, and you might want to grow that border with each next tile placement.)

The hexominoes can be flipped (2 ways) and rotated (4 ways) as needed to fit onto the board. (Per DFS or SS-Search, this means from a given board position (SS-node), a single tile might contribute up to 8 possible kid positions, at just one spot along the tile-area boarder.) The one duplicate hexomino is made up of a 2x2 block with "ear" squares on opposite sides of the block, one high and one low. It is circled at the right. (It is sometimes called the “short N” tile.)

You may use any algorithm you wish. Your project will include a write-up (description) of your algorithm, and a brief analysis of its running time (basic operations and Big-O class). Because this

involves both fitting and coloring, you can choose whether to combine these or to do them separately.

Board

As indicated, the tiles must fit into a 15x15 board with a 3x3 hole missing in the center of the board. This gives 216 grid cells, each the same as one square of a hexomino. The 36 (35 plus a duplicate) hexominoes also have 216 squares in total, so they will just fit snuggly. (You don't have to provide visible "grid lines", but to make the algorithm's progress clear, either use grid lines (which you will have to repaint) or pattern/texture/outline each tile so it is easy to follow which tile is being placed.

Coloring

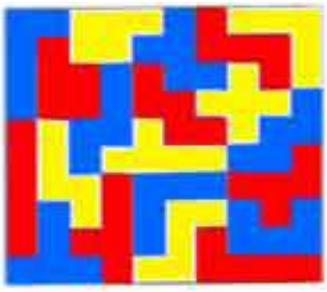
You can either pick (and change) colors for the tiles as you go, or you can do the coloring search after you have found a fit. (Note that we don't know if all coverings allow a 3-coloring.)

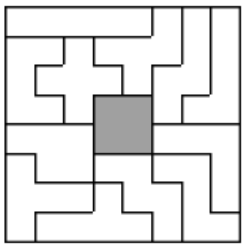
Every tile will receive one of only 3 colors, and that color will be used for all six of the tile's squares. (Pick bright colors and only one of Red,Green,Brown to make life easy for our crippled eyes.) A 3-coloring is one where no square of tile borders another square of a different tile where both tiles have the same color.

Tiles that only meet at a corner but don't share a square edge border can be colored the same -- the usual map-coloring constraint.

(We have in hand one complete 3-colored till covering as a proof of concept, and believe there are many more. YMMV.)

(You could create a graph of nodes (one per tile placed) and edges to the other tiles that border this one.)





8x8 Pent cover.

Example ***Pentomino*** 10x10 3-color covered board (w dup'd tiles). (20 tiles give a 20-node graph with the upper-left node connedted to 3 others (2 red tiles and one yellow tile).)

Tile Overlaps

You should be able to detect when one tile's placement overlaps another tile that is already placed. This means that you will need to know each tile pattern, and how it can be oriented (flipped, or rotated from some "canonical form" by 90,180,or 270 degrees).

You should also be able to determine which tiles border a give tile. You may or may not find it convenient to maintain a graph of which tiles border which. As an alternative, you could merely check the cross-border cells for their color/pattern/texture/whatnot.

(Sample Heuristic (could be poor): how many cell edges does the placed tile share with the growing border.)

Animation

Your program will show each tile being tried successfully. (You don't have to show any unsuccessful placements, if you don't want to.) Depending on how fast your algorithm seems, you may want to run it at full speed (eg, 24 fps).

Solution

If your program finds a solution, take a screen shot and submit it. If not, throw out a tile (reduce to 35 tiles) and try to find a 3-coloring of that. Continue reducing until you find a 3-color covering. Submit the best result you can find. (And by the way, if you are concerned, you can begin by trying to find something like a 30-tile 3-color solution first and build up to the full 36-tile 3-color solution.)

Team

The team size is the same as before, but you can change team members from the previous project if you wish (but do it soon).

Technical Debt

We emphasize working S/W (Rule #0). However, getting to working S/W fast often leaves technical debt – ugly code that is both hard to understand and complicates future modifications. Technical debt will rapidly turn into a Bad Smell if left too long to fester. Therefore, as this is the last project delivery, **the technical debt must be paid**, approximately in full; which is to say that your team's source code should be reasonably clean and well-documented (including reasonable code comments). (use pseudo-code (English-ish) sketch as code commentary – comments usually tell a) why you do it, or b) give you the bigger picture as to what is being done.) As mentioned in the first lecture, a handy way to get reasonably good comments is to write your functions/methods in natural (eg, English) language and then add this pseudo-code as comments to your actual code.

(Hex obj: id #, name, location (of its key cell, registration cell in the hexomino itself, orientation(rot & h-flip).)

Academic Rules

Correctly and properly attribute all third party material and references, lest points be taken off.

Project Reports, Submission & Readme, Grading

Same as before.