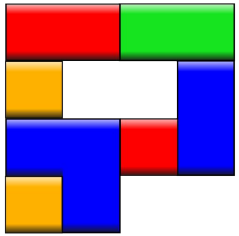


Cohesion

An AI perspective

Válter Castro up201706546@fe.up.pt
Vasco Garcia up201805255@fe.up.pt

Summary

Similar to 15 puzzle

Move pieces vertically or horizontally

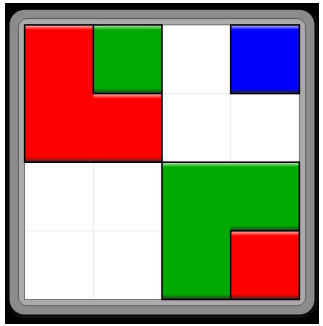
When pieces of the same color touch they become a single piece

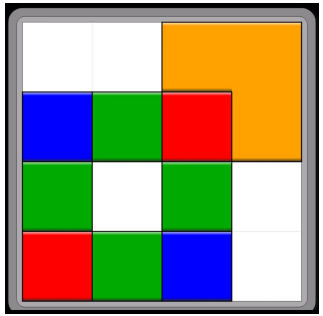
Join all the pieces of the same color to win

Made by NeatWits, 2015

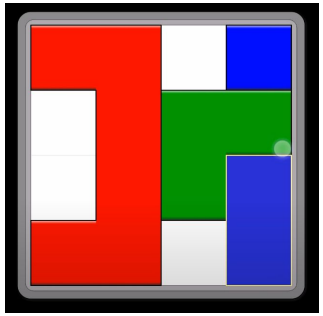
Summary

Difficulty varies





Easy to get stuck



Unitary moves

We can't 'skip' over a merge

The pieces we form are called polyominoes, more specifically fixed polyominoes since we cannot rotate or flip them.

https://en.wikipedia.org/wiki/Polyomino

cohesion demo video

Solving it

Treat it like a search problem

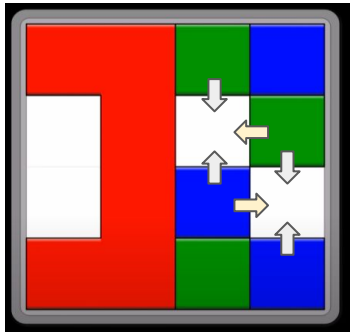
Every state has a finite number of possible moves

Tree-like searching algorithms where the nodes represent a game state

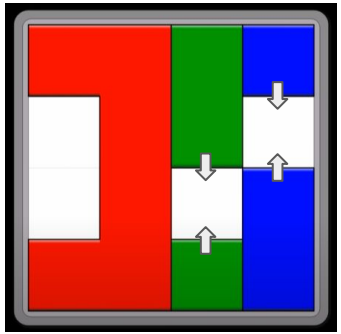
There can be multiple win/objective states

We can test for a objective state by checking if the number of colours is equal to the number of current shapes

We can easily derive a heuristic from that



6 child states



4 child states

Let h(s) be the difference between the number of colors and the number of shapes in the given state

$h(s) = 0 \Rightarrow$ s is a win state

Stuart J. Russell and Peter Norvig. (2020). *Artificial Intelligence: A Modern Approach 4th ed.* Pearson Education.

Implementation

Game logic

BoardState(width, height, pieces: set[Piece])

is_win() Checks if the number of pieces is equal to the number of different colors in the board

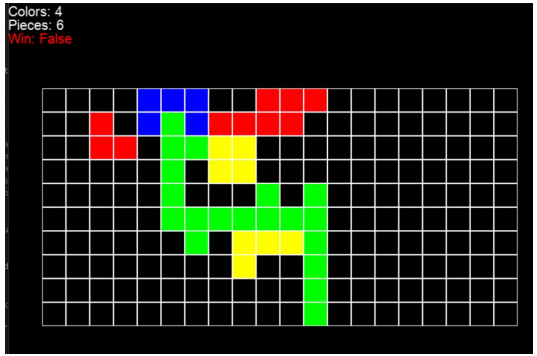
move(piece, direction) Used to move pieces around the board, used to generate children states

Piece(color, positions: set[(int,int)]) Represents a piece

Implemented algorithms

BFS, DFS, Greedy, Beam, A* , Weighted A*.

Basic graphics using *pygame*.



Search Performance

BFS performs very poorly

This is expected since the branching factor can get very large depending on the size of the board and the number of movable pieces.

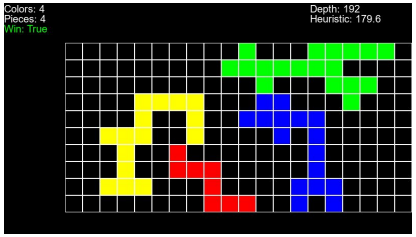
DFS and Greedy are better than expected

They can find solutions in small/medium size boards in an acceptable time.

However, this heavily depends on the ‘fullness’ of the board in the initial state, if it is high there is a very high probability that they will make one or many ‘bad’ moves, meaning they can venture down a whole search space that will not lead to a solvable state.

Such moves are critical in the early game since the branching factor is still very high and because the algorithms don’t have the intuition or lookahead capabilities.

Search Performance



A* / Weighted A*

A good balance between speed and depth(moves), its performance varies depending on the used heuristic(s) and weights.

Performance varies a lot depending on the weights used, not just the general heuristic weight (Weighted A*) but also the weight of each individual heuristic when we combine multiple of them.

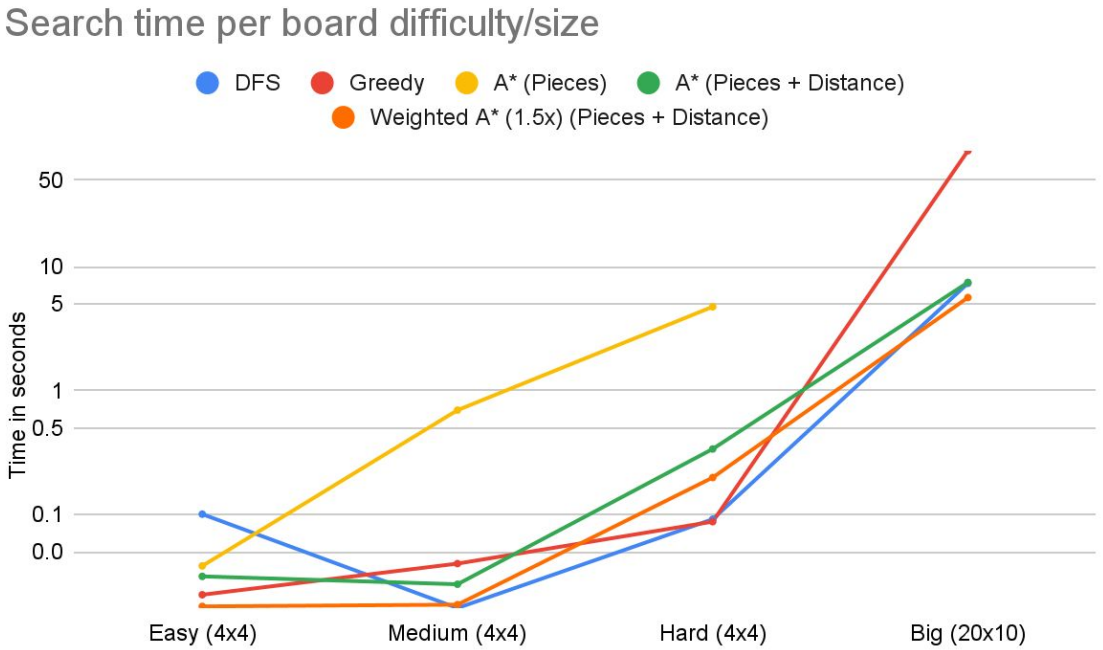
Beam Search

Can be pretty good in large boards but like other algorithms it may make bad moves and depending on the beam size it may run out of children without backtracking and thus stop without finding a solution

Large boards

All algorithms struggle because of critical moves, specially true on more dense boards as shown in the next slide. Using Weighted A* wields decent results, depending on tweaking the weights of multiple heuristics. We managed to get it to solve 10x10 boards with almost 50% of filled cells. Of course at this point we are happy to reach a solution even if it is not the optimal one thus some of the heuristics are not admissible.

Search Performance

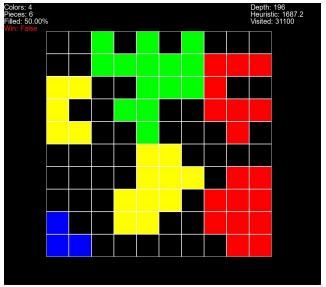


Large board sizes start becoming a problem really fast (notice the logarithm scale).

In this comparison the big board severely depletes the algorithms performances, some don't even finish in a reasonable amount of time.

Furthermore, if we were to opine on the difficulty of the big board it would probably be 'easy' since it was a very sparse board with just 14.5% of filled cells!

Optimizations



Heuristics

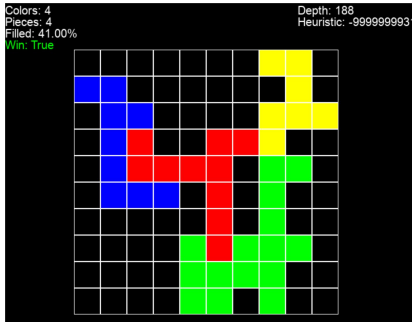
Manhattan distance - Pretty straightforward, it helps the algorithms join pieces by reducing the distance between pieces of the same color, however it also has some potential drawbacks.

Piece ‘uniformity’ - This comes from playing experience, where more neatly packed pieces are easier to navigate than pieces who branch off in “weird” forms, because they are less likely to collide with others and thus have more possible moves/children. It is however not always the best strategy because there could be situations where the optimal play involves forming “weird” pieces. On large boards it is highly beneficial to try to make compact pieces so we can move them more easily in the late game, because of this, the weight given to this heuristic must be chosen with care, otherwise the algorithms may ‘refuse’ joining pieces even if there is a solution close by.

Cut completed colors children

This is good for larger boards where pieces that have already been completed can wander off and that doesn’t bring us closer to a solution. But we have to be careful to not cut potentially useful moves.

Conclusions



Impossible/stuck detection is harder than it looks

Mutual locking rarely makes it unsolvable.

Pieces can have possible moves and yet be isolated and completely enclosed, making the puzzle unsolvable from there.

Flood fill algorithms (**expensive!**) can rule out some states however there are a lot of edge cases.

To make progress you often need to make it ‘worse’

This is a pattern prominent in larger boards where is is necessary to rearrange pieces in a way that temporarily worsens some heuristics. Eg: going around some other piece, moving pieces just to unblock another…

Erik D. Demaine and Robert A. Hearn. *Playing Games with Algorithms: Algorithmic Combinatorial Game Theory*