CSC466 Project Task 8

Task Explanation

Abstract

This task was supposed to be the implementation and testing of a constraint satisfaction rule to improve the heuristic player. Because of time constraints and poor planning on my part, I was unable to fully implement the rule. Instead I wrote up this document which explains how the rule would have worked and the core concepts behind a constraint satisfaction problem.

Constraint Satisfaction Problems

In essence, a constraint satisfaction problem is a mathematical representation of a problem consisting of states whose solutions must satisfy a set of constraints. There are three components to a CPS problem:

- A set of variables
- A set of domains for each variable
- A set of constraints

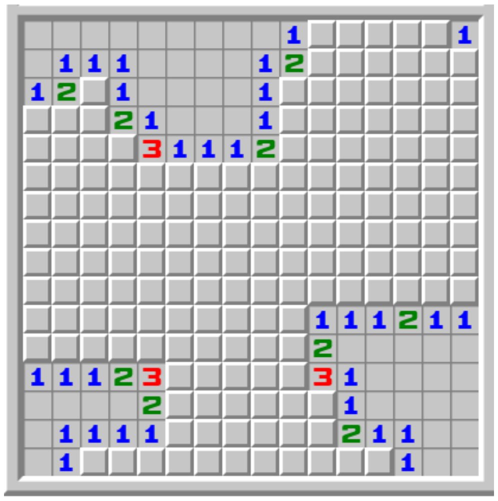
A constraint is a set of variables and a relation that defines the available values the variable can take. The goal of a CSP is to find a value for every variable so that no constraint is violated.

Minesweeper as a CSP

Minesweeper works well as a CSP. The tiles represent the variables. The domain of each tile is 0 or 1, where 0 represents no mine, 1 represents a mine. And the constraints consist of the set of neighboring tiles and the number of neighboring tiles which are mines. The algorithm would then look for constraints which could be combined and reduced to find tiles that can either be safely revealed or confidently flagged.

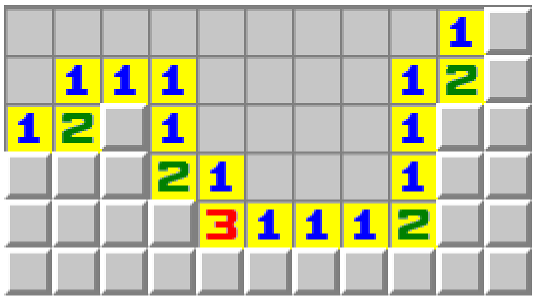
My Ideas

My first thought was that I needed to separate my edge tiles into multiple lists in case there was more than one open area. The image below shows and example of this, there are 3 different open sections, each would have to be its own list. One problem with this is how my original list of edge tiles that I wrote in task 7 added tiles in the order there are saved to the board which happens to be from left to right starting from the top left corner. This means that I would have needed a different function which worked based off of the tile's neighbors instead of the list.

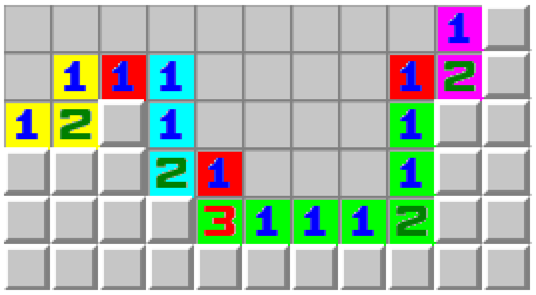


Next I would have made a constraint for each tile in each list which would contain a list of its neighbors and the number of unflagged neighboring mines. I would then need every subset of constraints in which each element is a neighbor of the previous element. I would then compare each subset to see if the combination of any two subsets could tile(s) which could be revealed or flagged.

A problem with this is that it would be inefficient to check every combination of subsets. For example, from the picture above this section contains 18 tiles, which is not the biggest number of tiles which could be in a list. The number of subsets which could be made from this list is high, since each subset would have to be compared to every other subset until one works.



A potential solution to this would be to separate this list above into smaller lists. Outside corners can be used to separate the list, as tiles on each side cannot help the other come to a solution. The list above could separated into 7 lists, 4 if the corners themselves are not counted as it only has 1 element.



With this knowledge and more time I could have implemented a working rule.