Module 3

By: Andreas Lønes and Ole Halvor Dahl

Representations

Domains are the possible values a row or column can have given its input.
Constraints are represented as two lists of lists. One containing the numbers of colored squares per row, and one containing the numbers of colored squares per column.
Variables or Lines, (rows or columns), are represented as a class with an index, a field, is_row, that represents whether or not the line is a row, and a list of possible domains.
States are represented by two lists of lines, rows and columns, an assumption (the one that was made to generate that state), a parent state, and g- and h- values.

When the puzzle is loaded, all possible and legal domains are generated and added to the lines.

Heuristics

The heuristic used by A* to chose the next state is calculated by summing the length of the domains of rows and columns minus one. That means that the heuristic increases for each variable that has more than one available domain, by the number of domains minus one. Minus one is because the goal is to have all variables only have one domain.
This is the method inside the state class that calculate a states heuristic:

```
def calculate_h(self):
        h = 0
        for n in self.get_rows():
            h += len( n.get_domain() ) - 1
        for n in self.get_cols():
            h += len( n.get_domain() ) - 1
        return h
```

The heuristic used to create new children is to find the row with the fewest values in the domain list, but more than one value. and generate one child state for each value in the domai. This is also done on the column with the fewest available values in the domain list. The row or column is only chosen if it has more than one value in the domain list.

Making A* GAC handle nonograms

This implementation of A* GAC uses specific code to handle nonograms. The core is A*, with the filtering mechanics of GAC. However, the revise method is rewritten to target nonograms directly. Revise takes in C and a state. C[0] contains the row/column that was changed, and C[1] contains the row/column to be reduced. It then loops over the domains of c[0] to see if any domains can be removed from c[1]. This is specific because it needs to know whether or not it is dealing with a row or a column

Aside from revise, all the code is general, and reused from earlier modules.

Other design decisions

A decision that had to be made was the problem should be represented. We chose to generate all the legal values for each row and column as the domains. We also generate all possible constraints and run Filter on the start state before we enter the Agenda-loop of A*. On all of the problems we were given, the nonnogram is almost solved before the loop begins.

This line of code runs Filter on the start state before the Agenda-loop begins:

```
Filter(start_state, make_all_constraints(start_state))
```

The method make_all_constraints(start_state) returns a queue of all constraint pairs.