Module 2

By: Andreas Lønes and Ole Halvor Dahl

Generality of the A* algorithm

The A* algorithm used in this module was based on the one from module 1. In conjunction with this, we implemented code regarding CSP and GAC in their own methods. This allows A* to operate the same way regardless of the problem it is fed to.
The filtering method is also a stand-alone function that can be reused when facing new CSP problems. However, the revise method is custom tailored to this specific problem in order to make the algorithm to run faster.

Generality of the A*-GAC algorithm

The A*-GAC algorithm is general in a way that new problems only need to formulate the constraints and domain of the variables. However the Revise() method is custom-fit to this specific problem. Revise holds the code that reduce the domains using the constraints. The code in revise takes a pair of nodes and reduce the domain of one the node according to the domain of the other node and the constraint that the nodes can not have the same color.

Separation of CNET and VIs and CIs

The constraints are stored in a separate multidimensional list. When a set of constraints are fed to the revise method, these are copies from the original constraints, and with pointers to the nodes. Each VI contains the domain of the variable, and a pointer to the CNET.

Explanation of the code

The code of module 2 does not use code chunks, or a hand-build interpreter. Revise contains code specifically written for this problem.

```
def revice(state, c):
    if len(state.nodes[c[0]].domain) < 2:
        return len(state.nodes[c[0]].domain)
    elif c[0] == c[1][0] and len(state.nodes[c[1][1]].domain) == 1:
        check_node = state.nodes[c[1][1]].domain[0]
    elif c[0] == c[1][1] and len(state.nodes[c[1][0]].domain) == 1:
        check_node = state.nodes[c[1][0]].domain[0]
    else:
        return len(state.nodes[c[0]].domain)
    for change_node in state.nodes[c[0]].domain:
        if change_node == check_node:
            state.nodes[c[0]].domain.remove(change_node)
            #print "changed node: ", state.nodes[c[0]].domain, "new len",
len(state.nodes[c[0]].domain)
            return len(state.nodes[c[0]].domain)
    return len(state.nodes[c[0]].domain)
```

Revise takes in the current state, and a pair of nodes (c). c looks like this: [object_1, [object_1, object_2]. c[0] is the node with a domain that will tried to be reduced. The two following is the constraint pair from the constraint list. Revice then tries to reduce the domain by removing possible colors that the other node can be based on the first node. The method returns the length of the new domain. If it has been reduced, Filter adds new constraints to the queue.