Comp 560 Fall 2015

Assignment 2: CSPs and Games

Due: October 8, 2015, 11:59pm

In this assignment, you will answer written questions and implement several algorithms related to CSPs and Games.

You are free to use any high-level programming language you are comfortable with and to work in groups of up to 3 people.

Part 1: Constraint Satisfaction Problems

Graduation Dinner (5 points)

You are helping to figure out the seating at your graduation dinner table. Your mom has already worked out seating on your side of the table, but wants you to decide where to put the relatives on the other side of the table. The relatives are your cousin Jasmine, your cousin Jason, your aunt Trudy, her husband Randy, and your aunt Misty. There are 5 seats, numbered 1-5 with 1 being the seat closest to the kitchen. You have the following constraints:

* No two relatives can sit in the same seat.
* Your cousins can't be seated next to each other because they will start a food fight.
* Jason as the eldest cousin gets the privilege of being seated closer to the kitchen than his sister Jasmine so he can get to the food first.
* Your aunt Misty shouldn't be seated next to Jason, Trudy or Randy because she invariably instigates an argument about politics and disrupts dinner.
* Additionally, Misty and Trudy have to be seated with at least 2 seats between them as a buffer.

Please answer the following written questions (**no implementation necessary**):

a) Formulate this problem as a binary CSP where the variables are relatives. In particular, state the domains and binary constraints on variables (e.g. different(A,B), notAdjacent(A,B), A<B, etc ...).

b) If you ran an arc consistency algorithm on this problem what would the domains be for each variable?

c) Which variable or variables would be assigned first according to MRV using the arc-consistent domains from part b?

d) If we assume that Randy is seated in the middle (seat 3) and run arc consistency, what will the remaining domains be?

e) List all solutions to this CSP with Randy in seat 3, or state that none exist.

Hide & Seek (8 points)

Local Search:

Emma, a freshman computer science student at UNC, has N friends. Her friends want to play "hide and seek" in a forest (a grid of size N x N) near campus. To start the game, each one of her N friends wants to stand in the forest (at some grid square) such that no one can see each other. All of her friends can see only in straight lines, in the 8 possible directions. The forest also contains trees. Friends cannot stand on tree grid squares or see past them. Your job is to help Emma write code to help her friends find places to hide – ie put each friend on a grid location such that no other friend can see them.

Let's define the CSP such that variables are grid locations, ie $\{A_{11},A_{12},A_{13},....\}$ with domains states of the grid, i.e. {friend, tree, empty}. Describe the constraints on the variables. Then, implement *local search (code)* to find an assignment that satisfies this problem (see below for a description of the problem input and format of the solution). You should initialize the locations of the friends randomly (one per column) and experiment with different methods for selecting which friend to move next. In your report, explain your approach and describe the selection methods you tried. Run your algorithm on different random initializations and several different input forests (for example experiment with differently sized grids or fewer/more trees). Include all of these in your report as well as the solutions found by your local search algorithm and the number of iterations executed by local search for each case. Note, sometimes local search can get stuck in local minima. Describe any inputs-initializations where this was the case.

Input: A text file input.txt where the first line of the input contains 2 integers where the first value corresponds to N (the number of friends) and the second value corresponds to K (the number of trees in the park). The next K lines contain the row and column locations of the trees in the park.

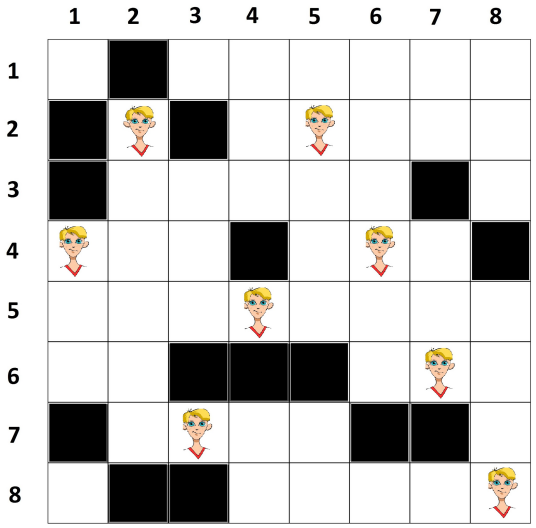
Output: Print out the row and column locations of each of the N friends.

Sample Input (from http://www.tamaraberg.com/teaching/Fall_15/HW/HW2/input.txt):
8 15
7 7
3 1
7 1
4 8
8 3
6 5
4 4
2 1
8 2
6 3
3 7
7 6
2 3
1 2
6 4

Sample Output:
2 2
2 5
4 1
4 6
5 4
6 7
7 3
8 8

Notes: In above explanation, indices start from 1, Output can be in any order.

Here is a possible solution for an 8x8 board (black cells represent trees):



Part 2: Minimax

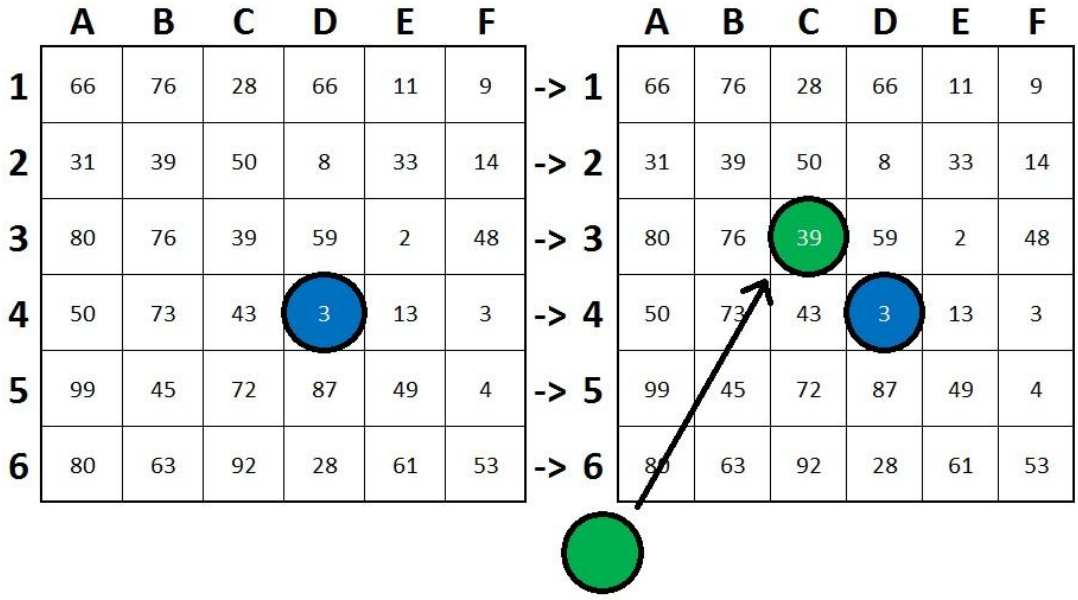
Candy game (12 points)

The goal of the game is to implement an agent to play a simple “candy game.”

Rules of the game:

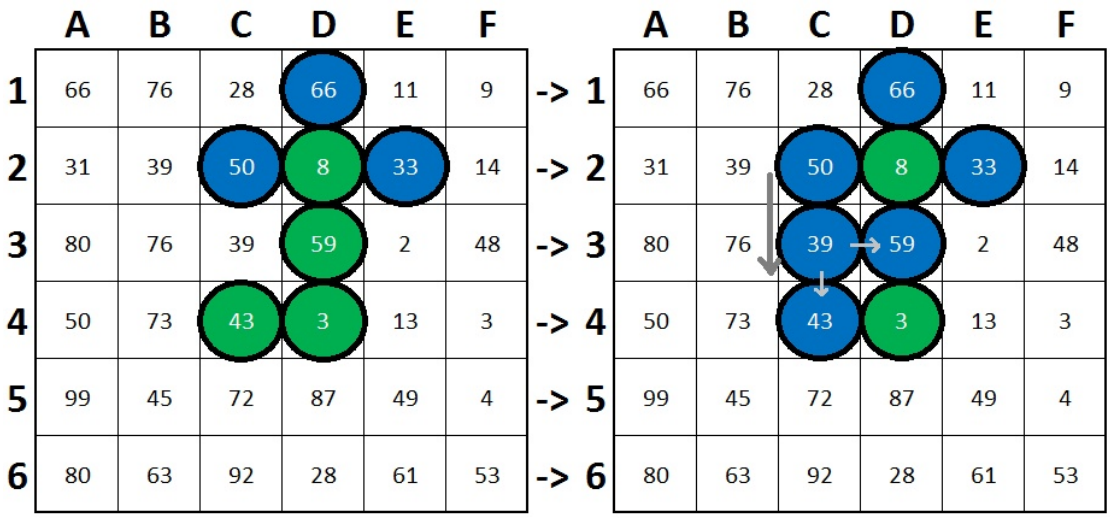
1. The game is played on a board of size 6x6.
2. Each cell of the board has a specified positive **value** (1 to 99).
3. There are two players, Blue and Green. Blue takes the first turn and Green takes the second and so on.
4. Each time Blue/Green “own” a cell, they place a colored candy piece in the cell to represent their ownership of the cell.
5. Blue/Green never run out of candy!
6. The objective of the game is to own cells with maximum total **value.**
7. The game ends when all the cells are occupied.
8. The values of the cells are variables that can be changed for each game, but stay constant during a game.
9. On a turn, a player can make one “**Drop and own”** move as follows: If any cell is unoccupied, Blue or Green can own it by placing a candy.
10. **Candy capture:** If a “Drop and own” move places a candy horizontally or vertically adjacent to other candies of the same color, then enemy candies that are horizontally or vertically adjacent to the placed candy are converted. Diagonal candies are not affected.

Here is a picture showing some example moves:



The figure above shows two “drop and own” moves. On the left, Blue makes a move, dropping its candy on [D,4] and getting 3 points. Then, as shown on the right, Green drops its candy on [C,3] and gets 39 points.

The following figure shows what happens when Blue drops its candy onto [C,3]. Because this is horizontally adjacent to the blue candy at [C,2], the green candies at [D,3] and [C,4] are “Candy Captured” and converted to blue, as shown on the right. Notice that in its next move, Green will not be able to “Candy Capture” any of Blue's candies.



Your task is to implement different agents to play this game, one using **minimax search** and one using **alpha-beta search**. Your program should use depth-limited search with an evaluation function -- which you need to design and explain in the report. Try to determine the maximum depth to which it is feasible for you to do the search (for alpha-beta pruning, this depth should be larger than for minimax). The worst-case number of leaf nodes for a tree with a depth of three in this game is roughly 42,840. Thus, you should at least be able to do minimax search to a depth of three.

Run the following matchups on the five game boards in (with blue, indicated on left, moving first): http://www.tamaraberg.com/teaching/Fall_15/HW/HW2/game_boards.zip

1. Minimax vs minimax;
2. Alpha-beta vs alpha-beta;
3. Alpha-beta vs minimax;
4. Minimax vs alpha-beta;

Carefully mention in the report:

1. The sequence of moves (e.g. “Blue: drop A3, Green: drop C1, etc. ”), the final state of the board (who owns each square) and the total score of each player.
2. The total number of game tree nodes expanded by each player.

3. The average number of nodes expanded per move and the average amount of time to make a move.

For bonus points:

1. Design an interface for the game that would allow you to play against the computer. How well do you do compared to the AI? Does it depend on the depth of search, evaluation function, etc.?
2. Design your own game boards and show results on them. Try to play the game on a larger board. How large can you go?
3. Describe and implement any advanced techniques from class or your own reading to try to improve efficiency or quality of gameplay.

Submission Instructions

HW should be uploaded to the classroom.cs.unc.edu server (one submission per group uploaded into the directory of one of the group members). Email Ric (poirson@cs.unc.edu) with the location of your HW submission, your group member names, and your PDF write-up. All submitted code must run on the classroom server for full credit. **Note**, if you edit any portion of your HW after the submission date that time-stamp will become your submission time and count into your late days.

Submissions should include:

1. A report in PDF format called HW2.pdf with the names of all group members indicated on top. The report should describe your implemented solution and fully answer the questions for every part of the assignment. Your description should highlight the most "interesting" aspects of your solution, i.e., any non-obvious implementation choices and parameter settings, and what you have found to be especially important for getting good performance. Feel free to include pseudo-code or figures if they are needed to clarify your approach. Your report should be self-contained and it should (ideally) make it possible for us to understand your solution without having to run your source code.

All group reports need to include a brief statement of individual contribution, i.e., which group member(s) was/were responsible for which parts of the solution and submitted material.

2. Your source code and a README file indicating how to run your code. Code should be well commented and it should be easy to see the correspondence between what's in the code and what's in the report.

Late policy: Assignments are due at 11:59pm on the assigned date. Students have 5 free late days to use over the course of the semester (e.g. an assignment submitted at 2am after the deadline will count as 1 day late). After free late days are used, assignments will be accepted up to 1 week late at a penalty of 10% per day.

Academic integrity: All code and written responses for assignments should be original within your group. To protect the integrity of the course, we will be actively checking for code plagiarism (both from current classmates and the internet).