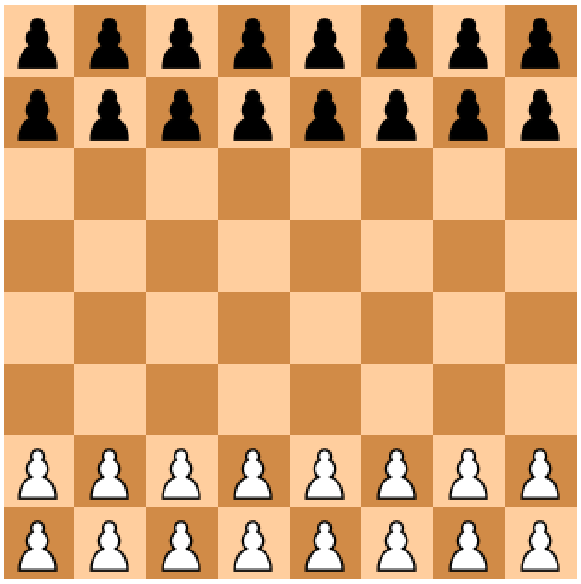


CS440/ECE448 Fall 2016

Assignment 2: Constraint Satisfaction Problems and Games

Deadline: Monday, October 24, 11:59:59PM

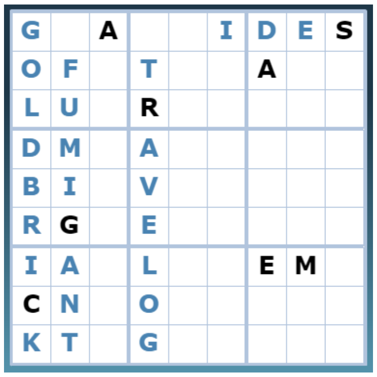
As on Assignment 1, you have the option of working in groups of up to three people. You are free to either stay with the same team or pick a new one, but as before, all the group members must be enrolled in the same section for the same number of credits (except for online students, who can work with Section Q students enrolled for the same numer of credits).

Contents

Part 1: CSP: Word Sudoku

Created by Daniel Calzada based on dkmGames version of the game

Sudoku is a puzzle that has gained much popularity since its first release in a US Newspaper in 2004. The object of the original Sudoku is to fill in a partially-completed 9x9 grid with numbers 1-9 such that each row, column, and the nine 3x3 sub-grids contain no duplicate numbers. In this assignment, you will implement a variation on Sudoku where, instead of filling the grid with numbers, you will fill the grid with words. You will be given a word bank, or a list of words that must be used exactly once (unless otherwise specified), and you must arrange them so that every cell in the grid contains a letter. Note that words will take up multiple cells in the grid and can be oriented either horizontally or vertically, and words are allowed to overlap. Following the rules of Sudoku, each row, column, and 3x3 cell cannot contain duplicate letters. To familiarize yourself with this game, we recommend playing it for yourself at http://dkmgames.com/WordSudoku.htm. The only difference between the online game and the game in this assignment is that in the online version, you are given the orientation of the words, but in this assignment, your code must be able to determine this for itself.



G		A			I	D	E	S
O	F		T			A		
L	U		R					
D	M		A					
B	I		V					
R	G		E					
I	A		L			E	M	
C	N		O					
K	T		G					

Word Sudoku game

The word bank will contain one word per line (example). The grid files will contain nine lines of nine characters each. If the character is an underscore "_", assume that the letter that goes in that cell is unknown. Otherwise, assume that the character belongs in the corresponding grid cell (example).

Your task is to implement backtracking search (see lecture) to solove the puzzles below. First, you need to determine your formulation of the CSP and include it in your report. What are the variables, domains, and constraints in your formulation? With these in mind, implement any ordering heuristics that make sense for your formulation to make your search efficient. Briefly describe your implementation.

For each of the inputs below, please include in your report:

- Your solution, the filled 9x9 grid that satisfies all the constraints (example). Please either include an image of the filled grid, or inline it in a monospace font (`Courier New`, for instance).
- In order, the sequence of assignments made, where each assignment contains the word, the coordinate of the top/left letter, and the orientation. Please follow the format in the example file, corresponding to the sample grid above. The lines are formatted as `O,R,C: WORD`, where O is the orientation (either H or V) and R and C are the row/column for the top-left coordinate of the first letter in the word. If you did not place this word on your grid (see Part 3), use `(N/A)` instead.
- The number of nodes expanded in the search tree. A state (either a partial or full set of assignments) is considered expanded when its children states, if any, are computed. This definition is equivalent to the definition of expanding a node in a search tree.
- The execution time of your search algorithm, in seconds or milliseconds.

Input 1 (for everybody): Start Grid with Hints
In this input, some of the cells in the grid will already be filled in for you.
Grid File | Word Bank

Input 2 (for everybody): Empty Start Grid
Solve another puzzle this time, except that unlike last time, you will not be given hints in the starting grid.
Grid File (empty) | Word Bank

Input 3 (for four credits only): Decoy Words
In this puzzle, you will be given a word bank and a non-empty start grid. The challenge here is that some of the words in the word bank (you don't know which ones) will not be used in the final solution. You will need to find which words should be placed and which ones shouldn't. Your solution should place **as few** words as possible. This requires traversing the entire search tree and generating all possible solutions, so think carefully about how you want to implement this in order to achieve a solution in a decent amount of time. In your report, briefly describe any modifications to your implementation you needed to make, and as part of your solution, also include in your sequence of assignments the words that were not placed, according to the notation above.
Grid File | Word Bank

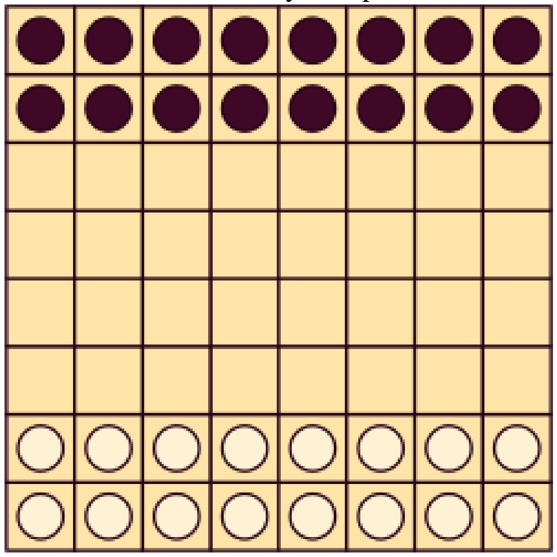
Part 2: Game of Breakthrough

Created by Shreya Rajpal based on this game

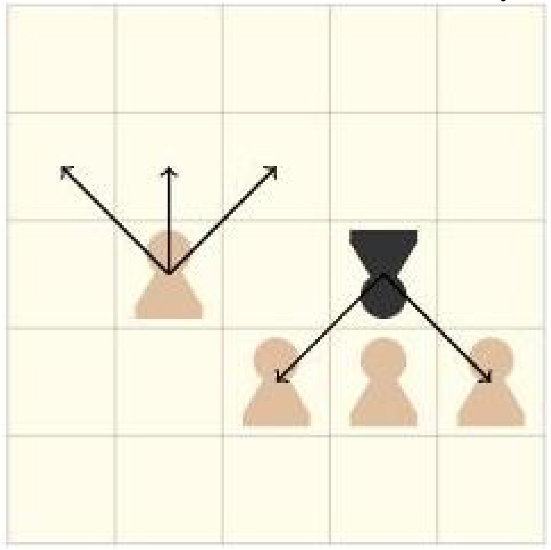
The goal of this part of the assignment is to implement an agent to play a simple 2-player zero-sum game called 'Breakthrough'.

Rules of the game

- The 8x8 board is initially set up as shown below. Each player has 16 workers in their team.



- Players play alternating turns, and can only move one piece (of their own workers) at a time.
- In each turn, a worker can only move *one square* in the forward or diagonally-forward directions, as shown below. Moreover, a worker can 'capture' workers of the enemy team if they are placed diagonally forward from it, as shown in the illustration below. Note that if enemy workers are directly in front of the player, then a capture isn't possible.



- The game finishes when (a) a worker reaches the enemy team's home base (the last row); or (b) when all workers of the enemy team are captured.

For more information, check out Breakthrough's Wikipedia page.

2.1 Minimax and alpha-beta agents (for everybody)

Your task is to implement agents to play the above game, one using **minimax search** and one using **alpha-beta search** (see this lecture) as well as two heuristic functions - one which is more **offensive** (i.e., more focused on moving forward and capturing enemy pieces), while the other which is more **defensive** (i.e., more focused on preventing the enemy from moving into your territory or capturing your pieces). Your program should use depth-limited search with an evaluation function -- which you, of course, need to design yourself 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 110,592, but in practice is usually between 25,000 - 35,000. Thus, you should at least be able to do minimax search to a depth of three.

You are to run four games for each of the following match-ups:

A. Minimax vs. minimax;
B. Alpha-beta vs. alpha-beta;
C. Minimax vs. alpha-beta (minimax goes first);
D. Alpha-beta vs. minimax (alpha-beta goes first).

The first game should have the **offensive** agent going first, the second game should have the **defensive** agent going first, the third should be **offensive vs. offensive**, and the fourth should be **defensive vs. defensive**. For each matchup, report the following:

1. The final state of the board (who owns each square) and the winning player;
2. The total number of game tree nodes expanded by each player in the course of the game;
3. The average number of nodes expanded per move and the average amount of time to make a move.
4. The number of opponent workers captured by each player, as well as the total number of moves required till the win.

Finally, you should summarize any general trends or conclusions that you have observed. How do search depth and type of evaluation function (offensive vs. defensive) affect the outcome? For players of equal strength (i.e., equal depth of search), what can give an advantage (going first, offensive vs. defensive evaluation function)?

Tips

- Pseudocode for alpha-beta pruning is given in Figure 5.7, p. 170, in the 3rd edition.
- For alpha-beta pruning, try to come up with a move ordering to increase the amount of pruning. Discuss any interesting choices in your report.
- For offensive vs. defensive evaluation functions, you may want to define the evaluation function as having two components: your score and your opponent's score. You would then weight these two components differently or combine them in different ways to get a more offensive or more defensive strategy. You can also read the Breakthrough Wikipedia page for possible features of a position to look for and prioritize for offensive vs. defensive play. Better yet, you should try playing the game yourself to get a better sense of what features or formations can occur.

2.2 Extended rules (for four-credit students)

Modify your implementation and evaluation functions to support the rule changes below and run several matchups for alpha-beta agents only with the maximum depth you can manage. You can choose any combination of offensive or defensive agents. Report outcomes of 2-4 representative matchups (or averages over several matchups of the same type), and summarize any interesting trends and differences from part 2.1.

1. **3 Workers to Base**: To win, 3 workers of a player's team must reach the opponent's base (as opposed to 1 in the original game). A player automatically loses when less than three of their pieces are left on the board. Therefore, the two ways to win the game would be to (a) move 3 pieces to the enemy's home base; (b) capture n-2 of the enemy's pieces, where n is the total number of players the enemy has at the beginning of the game.
2. **Long Rectangular Board**: For this variation, stick to the original, 1-worker-to-enemy-base win criterion. Instead, change the board shape to an oblong rectangle of dimensions 5x10.

For bonus points

- 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.?
- Implement any advanced techniques from class lectures or your own reading to try to improve efficiency or quality of gameplay.
- Implement a 1-depth greedy heuristic bot, and describe the differences between the gameplays of the greedy bot and the minimax bot.

Report Checklist

Your report should briefly describe your implemented solution and fully answer the questions for every part of the assignment. Your description should focus on 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 pseudocode 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. For full credit, your report should include the following.

Part 1:

1. For everybody: Give your CSP formulation (variables, domains, and constraints) and briefly describe your backtracking search implementation. Include the solution grid, sequence of assignments, number of nodes expanded, and execution time for Input 1 and Input 2.
2. For four-credit students: Include your solution to Input 3, briefly describing any modifications to your implementation.

Part 2:

1. For everybody: Describe your implementation, especially your choice of offensive and defensive evaluation functions. For matchups A-D (four each), report items 1-4.
2. For four-credit students: For two types of extended rules, describe your modified implementation (and especially evaluation functions) and report items 1-4 as in Part 1 for 2-4 matchups of any combination of maximum-depth alpha-beta agents (alternatively, report averages over multiple matchups of the same type). Discuss any interesting trends that you observe.

Extra credit:

- We reserve the right to give **bonus points** for any advanced exploration or especially challenging or creative solutions that you implement. Three-unit students always get extra credit for submitting solutions to four-unit problems. **If you submit any work for bonus points, be sure it is clearly indicated in your report.**

Statement of individual contribution:

- All group reports need to include a brief summary of which group member was responsible for which parts of the solution and submitted material. We reserve the right to contact group members individually to verify this information.

WARNING: You will not get credit for any solutions that you have obtained, but not included in your report! For example, if your code prints out path cost and number of nodes expanded on each input, but you do not put down the actual numbers in your report, or if you include pictures/files of your output solutions in the zip file but not in your PDF. The only exception is animated paths (videos or animated gifs).

Submission Instructions

As for Assignment 1, **one designated person from the group** will need to submit on **Compass 2g** by the deadline. Three-unit students must upload under **Assignment 2 (three credits)** and four-unit students must upload under **Assignment 2 (four credits)**. Each submission must consist of the following two attachments:

1. A **report** in **PDF format**. As for Assignment 1, the report should briefly describe your implemented solution and fully answer all the questions posed above. **Remember: you will not get credit for any solutions you have obtained, but not included in the report.**

   As before, all group reports need to include a brief **statement of individual contribution**, i.e., which group member was responsible for which parts of the solution and submitted material.

   The name of the report file should be **lastname_firstname_assignment2.pdf**. Don't forget to include the names of all group members and the number of credit units at the top of the report.

2. Your **source code** compressed to a **single ZIP file**. The 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. You don't need to include executables or various supporting files (e.g., utility libraries) whose content is irrelevant to the assignment. If we find it necessary to run your code in order to evaluate your solution, we will get in touch with you.

   The name of the code archive should be **lastname_firstname_assignment2.zip**.

Multiple attempts will be allowed but only your last submission before the deadline will be graded. **We reserve the right to take off points for not following directions.**

Late policy: For every day that your assignment is late, your score gets multiplied by 0.75. The penalty gets saturated after four days, that is, you can still get up to about 32% of the original points by turning in the assignment at all. If you have a compelling reason for not being able to submit the assignment on time and would like to make a special arrangement, you must send me email **at least a week before the due date** (any genuine emergency situations will be handled on an individual basis).

Be sure to also refer to course policies on academic integrity, etc.