Project IV: Mine Shafted

Due Fri, Jul 30 by 9PM Pacific Time

To begin, use the following file: http://users.csc.calpoly.edu/~dkauffma/480/mineshafted.py

- To download locally, right-click link and choose "Save link as...".
- To download and unzip on a CSL server run:
  `wget http://users.csc.calpoly.edu/~dkauffma/480/mineshafted.py`
  `unzip mineshafted.py`

Description

Minesweeper is a single-player game consisting of a MxN board of spaces that each either conceal a mine or a clue about possible adjacent mines. The player must uncover the unknown contents of these spaces by selecting them one at a time, using the revealed clues to determine which spaces are safe to explore. The game is won if the player is able to explore every space except the ones with mines.

An example of a 4x3 game instance both in progress and solved are shown below. An instance is considered solved if all spaces but those containing mines have been explored.

0	1	
0	2	
0	2	
0	1	

A. In Progress

0	1	1
0	2	
0	2	
0	1	1

B. Solved

The above game instance only has one solution; that is, from Diagram A, there are no other places the mines could be that would satisfy the contraints provided by the clues.

CSP Formulation

When a search problem is formulated as a constraint satisfaction problem (CSP), the states in the search space are represented as a collection of variables, each with their own domain. When all variables in a state are assigned a value, the state is

Secure Shell (SSH)

Vim Editor

complete. When variables with assignments in a state are only assigned values in their domains, the state is **consistent**. A state that is both complete and consistent is a solution to the CSP.

In Minesweeper, it may initially seem natural to represent the state space using unexplored spaces as variables, each with a domain of `{MINE, NO_MINE}`. However, this formulation quickly becomes problematic because there is no simple means to associate clues to collections of spaces. For example, in Diagram A above, it would be difficult to infer that indices 2 and 11 (the top and bottom of the right-most column) are safe if all we know about each unexplored space is that it may or may not contain a mine.

Instead of unexplored spaces, the clues themselves should be the variables, and each such variable's domain represents all the possible mine placements around that clue. The following boards represent all the possible mine placements with respect to only the clue `2` at index 4. When taking all clues into account, only Assignment #3 is possible.

0	1	✹
0	2	✹
0	2	
0	1	

Assignment #1

0	1	✹
0	2	
0	2	✹
0	1	

Assignment #2

0	1	
0	2	✹
0	2	✹
0	1	

Assignment #3

Thus, to set up the CSP, all possible mine placements with respect to each clue should be generated, which serve as that clue's domain. These domains will then be reduced during the propagation step (see below) to determine which spaces are safe.

Constraint Propagation

In a CSP, a pair of variables that share a binary constraint (that is, some limitation placed on both variables together), is known as an **arc**. If all binary constraints are satisfied in a state (an assignment of variables), that state is said to be **arc consistent**. It is possible to represent any CSP in terms of only binary constraints, and doing so can make writing the propagation algorithm easier. (The conversion to binary constraints can sometimes be cumbersome, as in the case of the 9x9 puzzle Sudoku, in which the cells in every row, for example, have 9-ary constraints.)

For this problem, any two clues that share at least one unexplored space form an arc. Diagram A has the following arcs, where each number refers to the index of a clue.

```
(1, 4) (4, 1) (1, 7) (7, 1) (4, 7) (7, 4) (4, 10) (10, 4) (7, 10) (10, 7)
```

Notice that the ordering of each arc matters - `(1, 4)` is distinct from `(4, 1)` - which will be important for the propagation algorithm.

By establishing the arcs between variables, their domains may be reduced (known as constraint propagation) by determing which values in each of the pair's domain conflict with one another. For example, in arc `(4, 1)`, since the clue at index 1 allows for only one adjacent mine, Assignment #1 above (taken from the domain of the clue at index 4) is not possible.

Constraint propagation may be performed over a set of arcs using the AC-3 algorithm. This process uses a set that contains every arc in the state space. As each arc is removed from the set, the domain of one variable (say, the left-sided one in the arc) is made arc consistent with the other variable by eliminating any values in the left-sided variable's domain that are impossible in the other. If no values are removed from the variable's domain, the algorithm moves on to the next arc; otherwise, all arcs containing the left-sided variable **on the right side** are added to the set. For example, since making `(4, 1)` arc consistent resulted in a domain reduction, then `(7, 4)` would need to be re-added. This step is necessary since reductions to the variable's domains might allow reductions of other variable's domains with which it shares a binary constraint, which would only be found if these arcs were brought back into the set. The algorithm continues until the set is empty, at which point all variables have been made arc consistent. However, if at any point a variable's domain is reduced to nothing, then there exists no solution from the state at which constraint propagation was run.

Unlike some CSPs, in Minesweeper, trial and error is not an option, as uncovering a mine ends the game (an example of an environment that is **not safely explorable**). Thus, all boards used for this assignment will be solvable using inference via constraint propagation without any guessing.

Exploring Safely

A Minesweeper solver must alternate between domain reduction (via AC-3) and inferring which spaces are safe in these reduced domains. Again using Diagram A above, the clues adjacent to an unexplored space have the following domains. In this notation, each integer represents a board index, with a positive integer representing a mine at that index and a negative integer representing that index as safe. Integers within parentheses represent one possible assignment in the domain.

When comparing two domains for a contradiction, only compare the indices they have in common. For example, when comparing the domains of Clues 1 and 4, ignore the `8`. Doing so shows that `(2, 5, -8)` is not possible because `(2, 5)` is not in Clue 1's domain.

Clue Index	Clue Value (Mine Count)	Domain
1	1	`(2, -5)` `(-2, 5)`
4	2	`(2, 5, -8)` `(2, -5, 8)` `(-2, 5, 8)`
7	2	`(5, 8, -11)` `(5, -8, 11)` `(-5, 8, 11)`
10	1	`(8, -11)` `(-8, 11)`

To start a domain reduction simulation, let's remove `(2, 5, -8)` since `2` and `5` cannot both contain a mine; `(-5, 8, 11)` can be removed for a similar reason. The updated domains are shown below. Note that AC-3 has not completed at this point, as more reductions can still be made.

Clue Index	Clue Value (Mine Count)	Domain	Note
1	1	`(2, -5)` `(-2, 5)`	
4	2	`(2, -5, 8)` `(-2, 5, 8)`	`8` must contain a mine
7	2	`(5, 8, -11)` `(5, -8, 11)`	`5` must contain a mine
10	1	`(8, -11)` `(-8, 11)`	

Since we now know that `5` and `8` contain mines, we can reduce the domains of Clues 1 and 10. Further reductions would then be made to Clues 4 and 7 (not shown).

Clue Index	Clue Value (Mine Count)	Domain	Note
1	1	`(-2, 5)`	`2` is safe

4	2	`(2, -5, 8)` `(-2, 5, 8)`	
7	2	`(5, 8, -11)` `(5, -8, 11)`	
10	1	`(8, -11)`	`11` is safe

Once the AC-3 algorithm has completed, the domains of one or more variables will have been sufficiently reduced so that a guaranteed-safe selection is possible (again, these problems assume that the board is solvable by pure logic). Recall that a variable's domain represents all possible assignments to that variable. Thus, if there exists an index in any domain that is always safe (i.e. every possible assignment in a domain has no mine at an index), that index must be safe. In this example, both indices `2` and `11` are always mine-free in at least one domain; thus, both of these spaces are safely explorable and either may be selected for the next move.

Implementation

Allowed Modules: `itertools` (for combinations and permutations)

For this assignment, Minesweeper boards will be represented as 2D lists of integers, in which each inner list represents one row of the board; within each row, a non-negative integer indicates the number of mines adjacent to the space at that position and `-1` represents a mine. The top-left index will always have a clue of `0`, indicating that its surrounding spaces are safe to explore - every search should start here. A sample 4x3 board is shown below.

```
[[ 0,  1,  1],
 [ 0,  2, -1],
 [ 0,  2, -1],
 [ 0,  1,  1]]
```

See the `mineshafted.py` starter file (linked above) for the initial structure of this program, including the provided `BoardManager` class, which provides an interface to select spaces to explore and receive clues about the number of mines surrounding that space.

Sample boards may be generated manually or using an external source such as World of Minesweeper (select "No guessing mode" but note that this site is not perfect at generating such boards).

```
sweep_mines(bm: BoardManager) -> List[List[int]]
```

Given a BoardManager (bm) instance, return a solved board (represented as a 2D list) by repeatedly calling `bm.move(index)` until all safe indices have been explored.

If at any time a move is attempted on a non-safe index, the BoardManager will raise an error; this error signifies the end of the game and cannot be caught.

```
board = [[0, 1, 1], [0, 2, -1], [0, 2, -1], [0, 1, 1]]
bm = BoardManager(board)
sweep_mines(bm) -> [[0, 1, 1], [0, 2, -1], [0, 2, -1], [0, 1, 1]]
```

Note that the `BoardManager` hides the contents of the board; to recreate it, this function will need to explore safe spaces until the locations of all mines can be inferred.

Scoring Rubric

The score you receive on this assignment will be based on which achievements the program satisfies. The achievements are listed in the order recommended to complete them.

	Achievement	Credit
1	Sweeps 1 Mine	10%
2	Sweeps 2 Mines	15%
3	Sweeps 3 Mines	20%
4	Sweeps 4 Mines	25%
5	Sweeps 5+ Mines	30%

Submission

On a CSL server with `mineshafted.py` in your current directory:

Instructor	Command
Daniel Kauffman	`/home/dkauffma/casey 480 mineshafted`