CSCI 3202 – Introduction to Artificial Intelligence
Instructor: Hoenigman
Assignment 3
Due: Friday, October 10, 3pm.

Can a Political Election be Rigged?

The drawing of political district boundaries is a contentious issue. It's been demonstrated repeatedly that boundaries can be drawn to give favor to one party, and that the same population divided up differently can result in a different outcome in an election. We can view a political election as a zero-sum game, where two parties (players), court the votes of individuals in swing districts or states, and there can be only one winner in the election. Both parties know which districts are contentious (they have perfect information), but they don't know how independent voters will ultimately vote, or what voter turnout will be on election day.

In this assignment, we're creating a game out of political districting. One political party is the MAX party and one party is the MIN party. Each party is provided with a map of MAX and MIN households in a neighborhood, and needs to select which households it wants in its districts. The party with the most majority districts at the end of the game is the winner of the game (and the election).

Input Data

There are two files on Moodle, one is called *smallNeighborhood* and the other is called *largeNeighborhood*. These files are the simulated census data that your program needs to use to play your game, i.e. you need to run your Minimax search on this data. The files each contain an *n x m* matrix of the predominant voting party in each block in the neighborhood. For example, the top left block of *smallNeighborhood* has a value of "R", signifying that voters in that cell vote for the Rabbit party. Your program will need to work on both files.

For *smallNeighborhood*, your game should divide the neighborhood into four districts of four blocks each. For *largeNeighborhood*, your game should divide the neighborhood into eight districts of eight blocks each. For each neighborhood, you are determining the outcome of the election if both players play the perfect game.

Game Rules

You are implementing Minimax search to find the optimal move for MAX and MIN to play the perfect game. Both MAX and MIN are searching for the district alignment that is in its own best interest. Your rules need to follow the Minimax pseudocode found on page 166 of your textbook. At each step, the MAX party selects the move that maximizes its utility, and MIN selects the move that minimizes its utility. In this game, we can think of MAX as the favored party, and MIN is playing the spoiler, preventing MAX from just rigging the election for its own benefit.

You will not be able to calculate the entire Minimax tree all the way to the terminal nodes for the *largeNeighborhood* data set. You will need to implement a depth-limited search or iterative-deepening search algorithm instead of calculating the entire tree. These algorithms are described on pages 87-90 of your textbook.

Rules:
The specific rules for how players select blocks is left as a design decision to the game designer. A few possible options have been selected in lecture. One option is to select *N* blocks at each turn, where *N* is the district size. This potentially leads to infeasible solutions that you will need to address in your program. Another option is to select *N/2* blocks the first few turns, and then reduce the number of blocks in each round to deal with blocks that have been isolated. In this implementation, selected blocks can be added to existing districts that are not yet the maximum size.

The rules are flexible, but there are some constraints that need to be met in any implementation you choose.

1. At each step, MAX and MIN select *X* blocks from the neighborhood of available blocks.
2. Blocks in a district need to be contiguous.
3. The district needs to have a maximum block size.
4. MAX and MIN evaluate selections using a heuristic evaluation function that captures relevant details of the state space
5. Winner is the player with the most majority districts after all blocks selected.

Data structures and algorithms

Your program needs to store the neighborhood in a data structure of your choosing. The data structure you choose will influence the rules of your game, but you will ultimately be evaluated on the outcome of your game, not necessarily your implementation decisions.

Your code needs to include a comment block that describes your game algorithm, i.e. your approach for each player selecting blocks for districts. The comments need to be specific enough that the TA and Instructor can understand what you did.

Running your program and program output:

This assignment is more free-form than the first two, and assignments like this can be difficult for the grader to grade.

The main entry point to your program needs to be called gerrymander.py and take one command-line argument, the data file that the program will use for playing the game. We will run your program with:

```
>>python gerrymander.py smallNeighborhood.txt
>>python gerrymander.py largeNeighborhood.txt
```

Display:
When the grader runs your program, it should produce a display of the relevant information.

Output the label in the .txt files for who is playing MAX and who is playing MIN.

```
************************************
MAX=<R or D>
MIN=<R or D>
************************************
```

After the algorithm runs and districts are assigned, output the cells assigned to each district. For example,

```
************************************
District 1: (1,1), (1,2), (1,3)
District 2: (3,1), (3,2), (3,3)
************************************
```

Also, output the districts awarded to each player.

```
************************************
District 1: R
District 2: D
************************************
```

Finally, output who won the most districts, and therefore, won the election.

```
************************************
Election outcome: <R or D> wins
************************************
```

Evaluation function

You need to implement an evaluation function for scoring each state. The evaluation function needs to be the same for MAX and MIN, i.e. a state that is desirable for MAX should have a highly positive score and a state that is desirable for MIN should have a highly negative score.

During class, there were a few heuristics suggested for evaluating a state. One potential approach is to assign values that group MIN and create a slim majority in each district for MAX.

For example, if there are N cells assigned to a district, then the evaluation function for each state could be:

MAX	MIN	Eval(s)
0	N	N+1
1	N-1	N
2	N-2	N-1
…		
N-1	1	2
N	0	1

You will likely need to experiment with different weights for different state configurations to explore different election outcomes. Different weights will likely result in different districts being drawn, and ultimately, a different election outcome.

Handling infeasible states:

Your program will need to handle infeasible states (states that violate one of the constraints) by either predicting that a path will end in an infeasible state and pruning that branch in the tree, or recursing back to a feasible state when an infeasible state is encountered and then following a different branch in the tree.

As an example of an infeasible state, consider the following 9-block neighborhood that is labeled as Player-Block Index.

R-1	D-2	R-3
D-4	D-5	R-6
D-7	R-8	R-9

A feasible state of three districts is (1, 2, 3), (4, 5, 7), (6, 8, 9). All districts contain contiguous blocks and the same number of blocks. An infeasible state is (2, 4, 5), (1, 7, 8), (3, 6, 9) because the (1, 7, 8) district is not contiguous.

Extra credit:

You can choose to implement neither, one, or both of these extra credit options.

Alpha-beta search

Implement alpha-beta search on the Minimax search tree. Your program needs to output the state of the root node of the pruned branch as the game is being played. For example, using the 9-block neighborhood from above, your code could output:

****
Pruned branch: (1, 2, 3), (4, 5, 6), (7, 8, 9)
****

if all children of that node in the tree could not possibly contain the optimal solution.

Voter turnout or independent affiliation

So far, we have assumed that voter turnout will be 100% and all blocks were equivalent in regards to number of voters. However, in the real world this is not the case. Voter turnout is often influenced by voter registration campaigns and whether an election will be a close race. In many districts, parties will target the small number of registered independent voters to gain a slim majority over their opponent.

Design an evaluation function that considers voter turnout in evaluating a game state. Your function should use conditional probabilities, e.g. the probability of MAX voter turnout in a block given that most blocks in the district are assigned to MIN. You may want to do some research on voter turnout and factors that influence the probability of individuals voting.

Output two election outcomes using different voter turnout probabilities. Use the same output format as listed above, where you display the blocks assigned to each district and the districts assigned to each party and the final election winner. You also need to describe how you've included probability in your evaluation function in the comments in your code.