Birgit Alitz (alitz2, 3-credit)
Kate Eaton (kkeaton2, 3-credt)
10/30/2017

MP2

Part 1: Flow Free

1. 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.

Variables

In this problem, every cell in the grid that does not have a predetermined (initial) value is represented by a variable.

Domains

The variables in the problem can take any value over the domain {R, G, B, Y, O, etc. }
The value '_' indicates a variable that has no value, and is not in the domain of the solution.

Constraints

The variables must form a "path" of one domain value between the two initially declared cells with the same domain value.
Of the 2 to 4 adjacent surrounding variables, the variable can only match one in color when being placed to avoid a 'zigzag' pattern. The final assignment of a color will match two -- the previously declared variable and the end variable which is part of the initial puzzle.

$x_{n,m} != x_{n-1,\,m}$

$x_{n,m} != x_{n,\,m+1}$

$x_{n,m} != x_{n,\,m-1}$

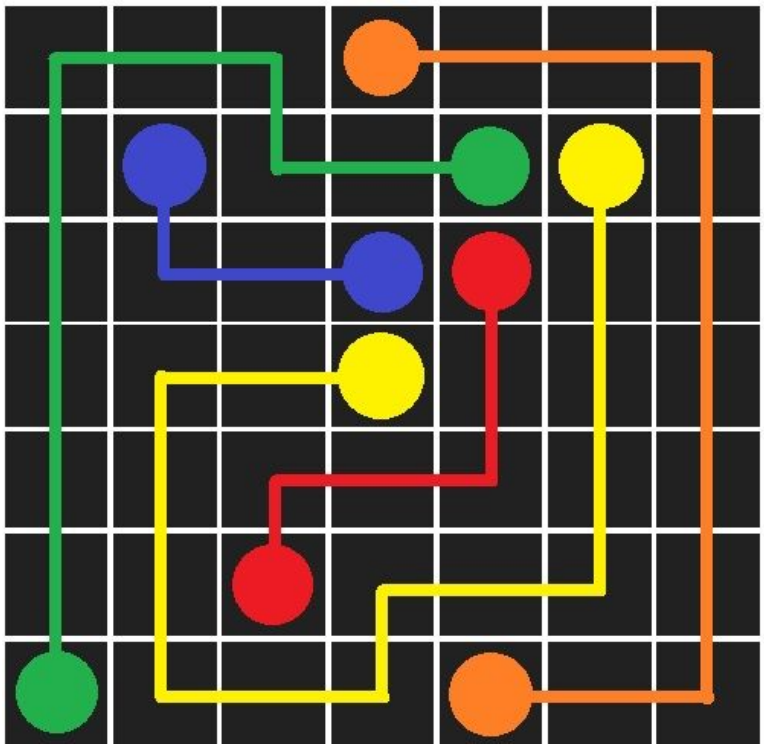
Our 'smart' implementation improved the inference technique. We use a priority queue and start at an initial value, and try to find the other initial value matching that value, which are ordered based on the manhattan distance between the current position and the final initial value. It also implements forward checking by making sure that no neighbors will become invalidated after the current value is assigned.

Results: (visualizations attached for extra credit)

7x7 Puzzle:

GGGOOOO
GBGGGYO
GBBBRYO
GYYYRYO

GYRRRYO
GYRYYYO
GYYYOOO



Dumb Implementation:

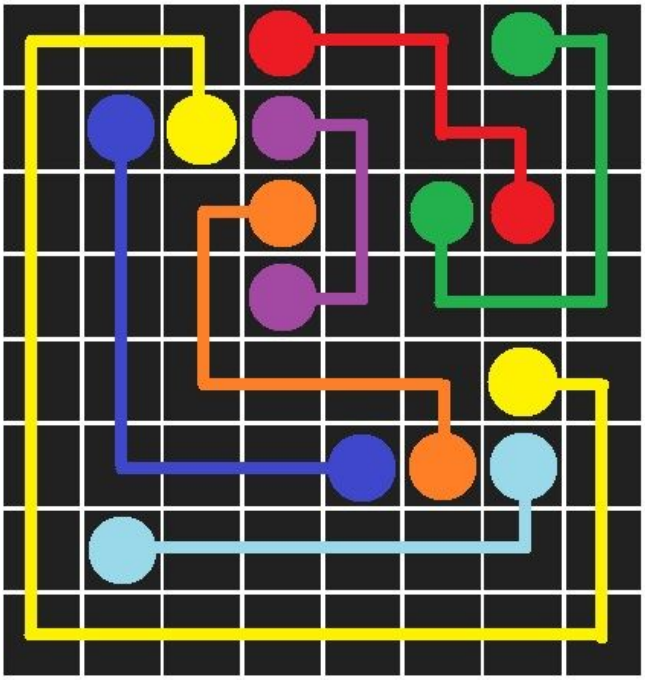
- Attempted Variable assignments: 383
- Runtime: 309 ms

Smart Implementation:

- Attempted Variable assignments: 10010
- Total execution time: 275 ms

8x8 Puzzle:

YYYRRRGG
YBYPPRRG
YBOOPGRG
YBOPPGGG
YBOOOOYY
YBBBBOQY
YQQQQQQY
YYYYYYYY



Dumb Implementation:

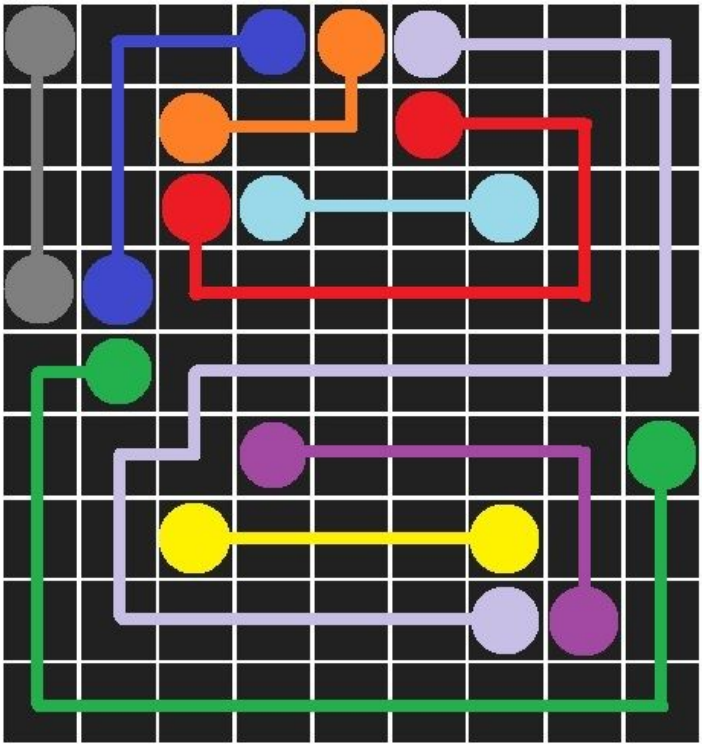
- Attempted Variable assignments: too many
- Run time: too long, fails to complete

Smart Implementation:

- Attempted Variable assignments: 49714
- Runtime: 896 ms

9 x 9 puzzle:

DBBBOKKKK
DBOOORRRK
DBRQQQQRK
DBRRRRRRK
GGKKKKKKK
GKKPPPPPG
GKYYYYYPG
GKKKKKKPG
GGGGGGGGG



Dumb Implementation:

- Attempted Variable assignments: too many
- Run time: too long, fails to complete

Smart Implementation:

- Attempted Variable assignments: 82589
- Runtime: 1441 ms

Part 2: Breakthrough

2. Your task is to implement agents to play the above game, one using **minimax search** and one using **alpha-beta search** as well as two evaluation 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).

Representation

Each node is represented by a complete game board (2-Dimensional ArrayList) with the positions and player (player 0 vs player 1) of each piece. Computations are done on ArrayListed tuples (x-coordinate, y-coordinate) associated with each player.

Hueristics

Offensive 2 created to beat Defensive 1

Offensive 2 plays like Defensive one, but places an additional constraint on distance. If a piece is far away from the goal, Offensive 2 will play defensively, prioritizing saving its own pieces to keep a strong wall against Defensive 1. However, once the wall was progressed to the second half of the map, pieces closer to the goal will take priority and attempt to dart through Defensive 1. The final equation for Offensive 2 is

```
2*(Number_MyPieces) + 4*(8-distance) + Math.random()
```

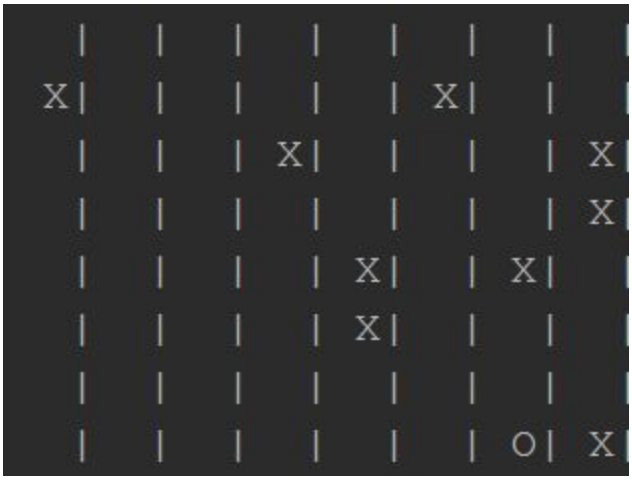
Defensive 2 created to beat Offensive 1

Defensive 2 plays like offensive one, but also places a priority on distance from the goal. If a piece is near the opposite side, it will instead charge to the end instead of attacking the enemy piece. Otherwise, pieces far from the goal remain for from the goal to guard against incoming enemy pieces.

```
-2*(Number_TheirPieces) + 4*(8-distance) + Math.random()
```

Results

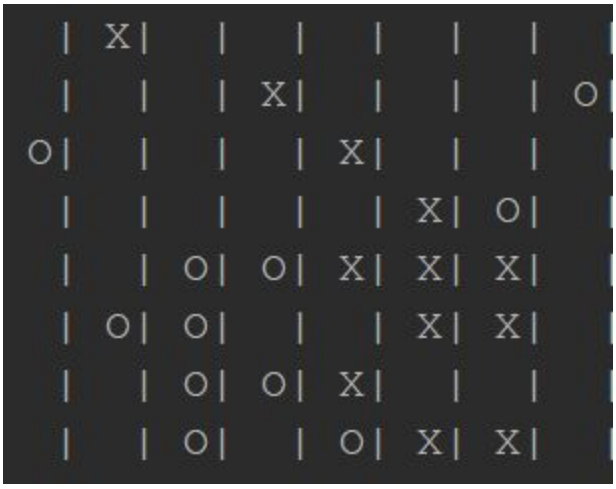
1. Minimax (Offensive Heuristic 1) vs Alpha-beta (Offensive Heuristic 1)
    a. Results
        i. Final Board State



        ii. Winner: Player 1 (X)
    b. Game Tree Nodes Expanded
        i. Player 0 (Minimax Off 1): 11279981
        ii. Player 1 (AlphaBeta Off 1): 2800564
    c. Averages
        i. Nodes Per Move
            1. Player 0 (Minimax Off 1): 268570
            2. Player 1 (AlphaBeta Off 1): 66680
        ii. Time Per Move
            1. Player 0 (Minimax Off 1): 1164 milliseconds
            2. Player 1 (AlphaBeta Off 1): 281 milliseconds
    d. Winning Stats
        i. Players Captured
            1. Player 0 (Minimax Off 1): 7
            2. Player 1 (AlphaBeta Off 1): 15
        ii. Total Moves
            1. Player 0 (Minimax Off 1): 42
            2. Player 1 (AlphaBeta Off 1): 42

Trends: The AlphaBeta search won more often despite using the same algorithm because it searched to a lower depth.

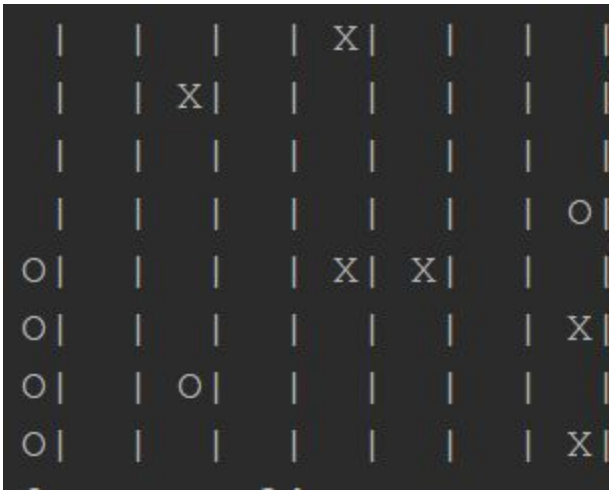
2. Alpha-beta (Offensive Heuristic 2) vs Alpha-beta (Defensive Heuristic 1)
    a. Results
        i. Final Board State



        ii. Winner: Player 0
    b. Game Tree Nodes Expanded
        1. Player 0 (AlphaBeta Off 2): 2439996
        2. Player 1 (AlphaBeta Def 1): 2424666
    c. Averages
        i. Nodes Per Move
            1. Player 0 (AlphaBeta Off 2): 62564
            2. Player 1 (AlphaBeta Def 1): 63807
        ii. Time Per Move
            1. Player 0 (AlphaBeta Off 2): 346 ms
            2. Player 1 (AlphaBeta Def 1): 350 ms
    d. Winning Stats
        i. Players Captured
            1. Player 0 (AlphaBeta Off 2): 4
            2. Player 1 (AlphaBeta Def 1): 6
        ii. Total Moves
            1. Player 0 (AlphaBeta Off 2): 39
            2. Player 1 (AlphaBeta Def 1): 38

This heuristic was particularly hard to design since a Defensive algorithm could always react to an Offensive move that preceded it. In the end, this heuristic was still not very efficient.

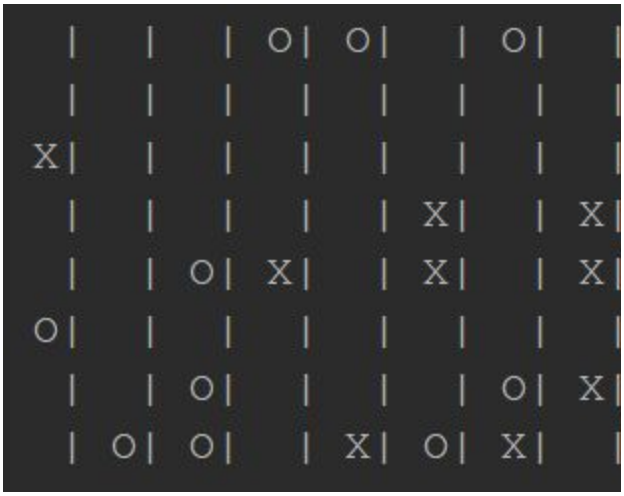
3. Alpha-beta (Defensive Heuristic 2) vs Alpha-beta (Offensive Heuristic 1)
    a. Results
        i. Final Board State



        ii. Winner: Player 0
    b. Game Tree Nodes Expanded
        1. Player 0 (AlphaBeta Def 2): 2645552
        2. Player 1 (AlphaBeta Off 1): 234779
    c. Averages
        i. Nodes Per Move
            1. Player 0 (AlphaBeta Def 2): 77810
            2. Player 1 (AlphaBeta Off 1): 69111
        ii. Time Per Move
            1. Player 0 (AlphaBeta Def 2): 380 ms
            2. Player 1 (AlphaBeta Off 1): 331 ms
    d. Winning Stats
        i. Players Captured
            1. Player 0 (AlphaBeta Def 2): 15
            2. Player 1 (AlphaBeta Off 1): 9
        ii. Total Moves
            1. Player 0 (AlphaBeta Def 2): 34
            2. Player 1 (AlphaBeta Off 1): 33

This heuristic was far easier to design since Offensive 1 would capture pieces, but ignore the goal. This was much more effective.

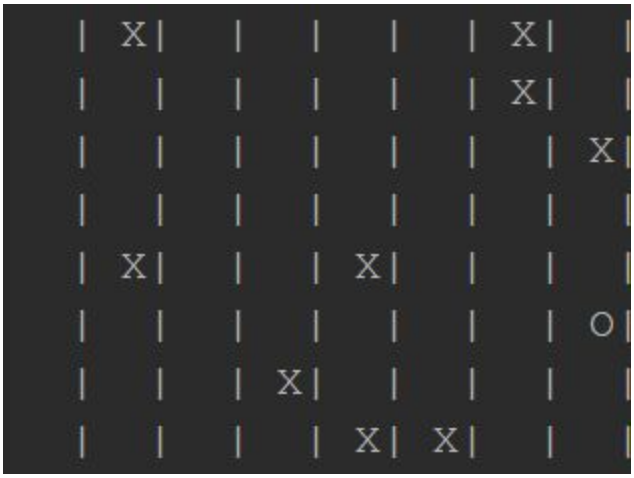
4. Alpha-beta (Offensive Heuristic 2) vs Alpha-beta (Offensive Heuristic 1)
    a. Results
        i. Final Board State



        ii. Winner: Player 1
    b. Game Tree Nodes Expanded
        1. Player 0 (AlphaBeta Off 2) 3875647
        2. Player 1 (AlphaBeta Off 1) 3989834
    c. Averages
        i. Nodes Per Move
            1. Player 0 (AlphaBeta Off 2): 94527
            2. Player 1 (AlphaBeta Off 1): 97313
        ii. Time Per Move
            1. Player 0 (AlphaBeta Off 2): 451 ms
            2. Player 1 (AlphaBeta Off 1): 467 ms
    d. Winning Stats
        i. Players Captured
            1. Player 0 (AlphaBeta Off 2): 7
            2. Player 1 (AlphaBeta Off 1): 7
        ii. Total Moves
            1. Player 0 (AlphaBeta Off 2): 41
            2. Player 1 (AlphaBeta Off 1): 41

Since Offensive 2 was designed somewhat to be a better version of the Defensive heuristic, it tended to lose against Offensive 1, as it would wait too long to start moving pieces that were further along toward the goal.

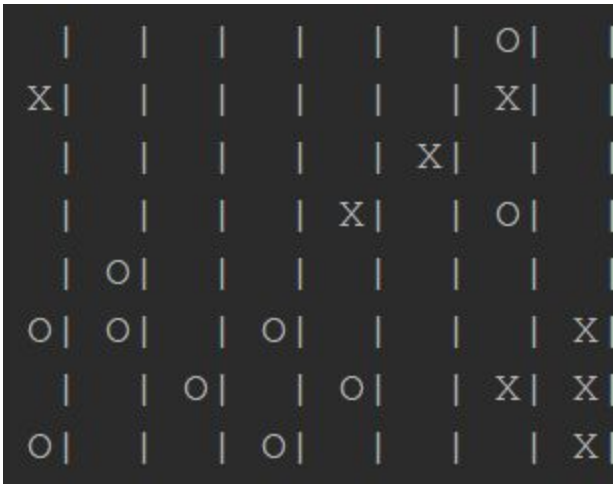
5. Alpha-beta (Defensive Heuristic 2) vs Alpha-beta (Defensive Heuristic 1)
    a. Results
        i. Final Board State



        ii. Winner: Player 0
    b. Game Tree Nodes Expanded
        1. Player 0 (AlphaBeta Def 2) 4031640
        2. Player 1 (AlphaBeta Def 1) 3263281
    c. Averages
        i. Nodes Per Move
            1. Player 0 (AlphaBeta Def 2): 85779
            2. Player 1 (AlphaBeta Def 1): 69431
        ii. Time Per Move
            1. Player 0 (AlphaBeta Def 2): 371 ms
            2. Player 1 (AlphaBeta Def 1): 297 ms
    d. Winning Stats
        i. Players Captured
            1. Player 0 (AlphaBeta Def 2): 7
            2. Player 1 (AlphaBeta Def 1): 15
        ii. Total Moves
            1. Player 0 (AlphaBeta Def 2): 47
            2. Player 1 (AlphaBeta Def 1): 46

Running these resulted regularly in a single Defensive 2 piece reaching the opposite side, as the Defensive 1 pieces would routinely move out of the way and destroy the wall created by Defensive 2.

6. Alpha-beta (Offensive Heuristic 2) vs Alpha-beta (Defensive Heuristic 2)
    a. Results
        i. Final Board State



        ii. Winner: Player 1
    b. Game Tree Nodes Expanded
        1. Player 0 (AlphaBeta Off 2) 2093445
        2. Player 1 (AlphaBeta Def 2) 2302538
    c. Averages
        i. Nodes Per Move
            1. Player 0 (AlphaBeta Off 2): 56579
            2. Player 1 (AlphaBeta Def 2): 62230
        ii. Time Per Move
            1. Player 0 (AlphaBeta Off 2): 511 ms
            2. Player 1 (AlphaBeta Def 2): 550 ms
    d. Winning Stats
        i. Players Captured
            1. Player 0 (AlphaBeta Off 2): 8
            2. Player 1 (AlphaBeta Def 2): 6
        ii. Total Moves
            1. Player 0 (AlphaBeta Off 2): 37
            2. Player 1 (AlphaBeta Def 2): 37

This simulation was won almost equally by each heuristic in multiple tests, and tended to take either more turns than usual or fewer (with one test taking as few as 25 turns). This may be because of the "rush vs stall" function built into each heuristic.

Statement of Individual Contribution:
Kate all of part 1, wrote that part of this report, and aided Birgit in designing heuristics for part 2. Birgit did all of part 2, wrote that part of this report, and aided Kate in designing the heuristics for part 1.