CS 440 Artificial Intelligence

Adarsh Ramchandran (adarshr2) - Section R
Jessica Gomes (jsgomes2) - Section Q

2.24.19

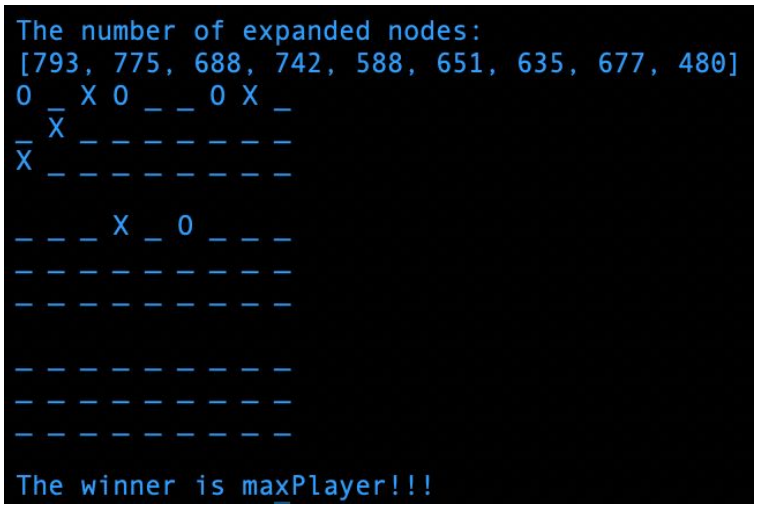
Section I : CSP

Our representation of the problem consisted of two dictionaries, constraints and choices, that represent a bipartite graph. The constraints dictionary maps coordinates on the grid that had a '1', represented as tuples (x, y) , to pentominos, represented as a tuple formatted as (pent_idx, orient_idx). The choices dictionary maps pentominos represented as a tuple, (pent_idx, orient_idx) to each top-left coordinate the pentomino can fit in. As we place pentominos, we remove from the constraints and choices library accordingly. We use the LRV (Least Remaining Value) heuristic to choose which pentomino to place next, specifically by looking at the length of the value array in the choices dictionary. If the length of the value of any of the pentominos in the choices library is 0, that means we have run into an invalid solution and must backtrack and try something else. If all the constraints are satisfied, then we know we have a solution for the puzzle.

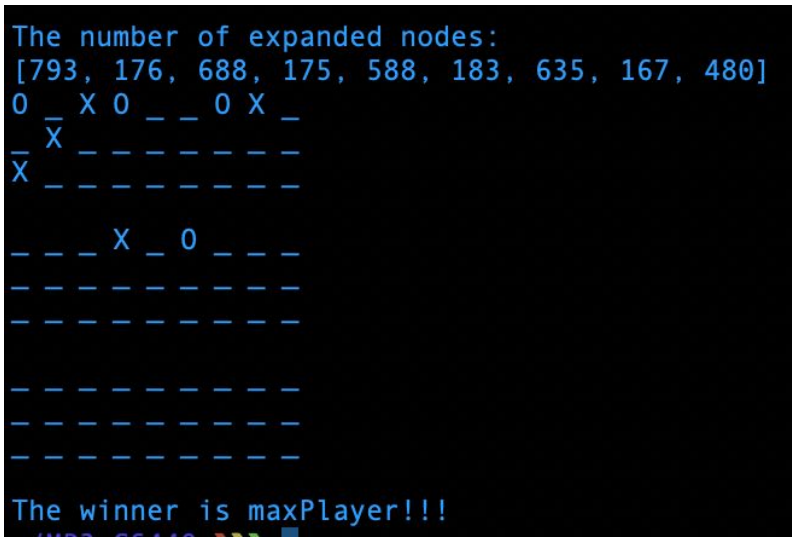
We implemented a backtracking algorithm to solve the problem. The heuristic we used relied on the two aforementioned dictionaries, constraints and choices. It is a Least Remaining Value heuristic, and it chooses to first try and place pentomino with the least number of valid placement choices. Every time a pentomino is placed, we make sure to remove all rotations/flips of that pentomino from the choices dictionary and remove the filled coordinates from constraints. From there, it recurses and continually chooses the pentomino with the lowest number of valid placement options until either the board is filled properly or it is invalid and we need to try another board placement. In this case, we undo our move, undo all the changes *that specific move* made to both dictionaries, and attempt another placement.

Section II : Ultimate Tic Tac Toe Predefined Agents

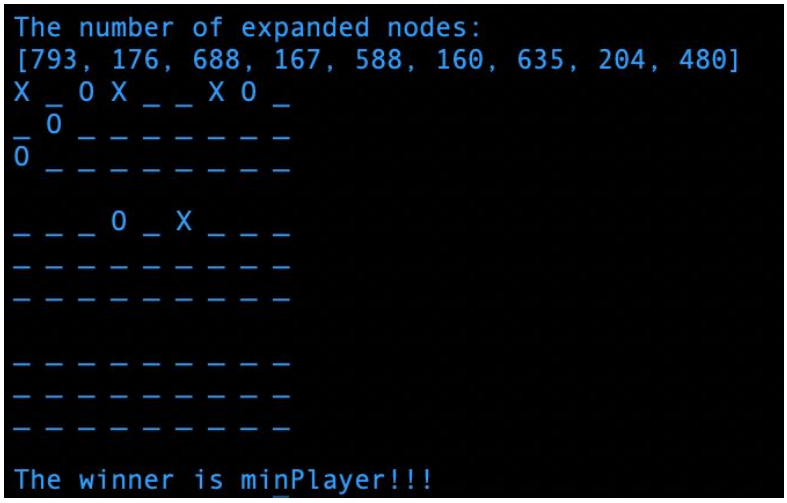
offensive(minimax) vs defensive(minimax), maxPlayer first



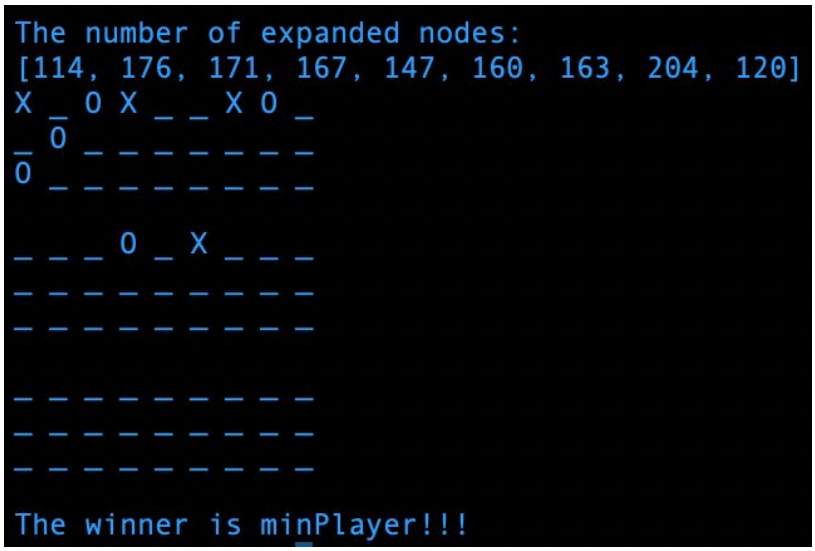
offensive(minimax) vs defensive(alpha-beta), maxPlayer first



offensive(alpha-beta) vs defensive(minimax) Min player first



offensive(alpha-beta) vs defensive(alpha-beta) Min player first

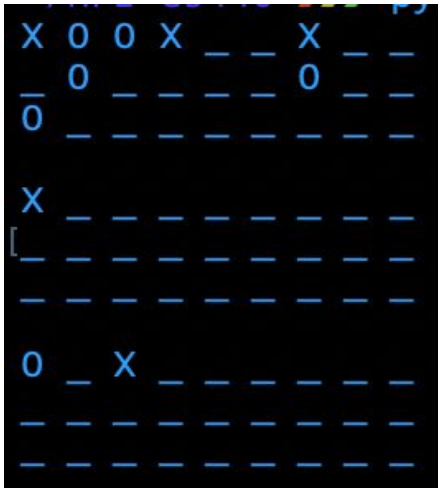


Section III : Ultimate Tic Tac Toe Your Agent

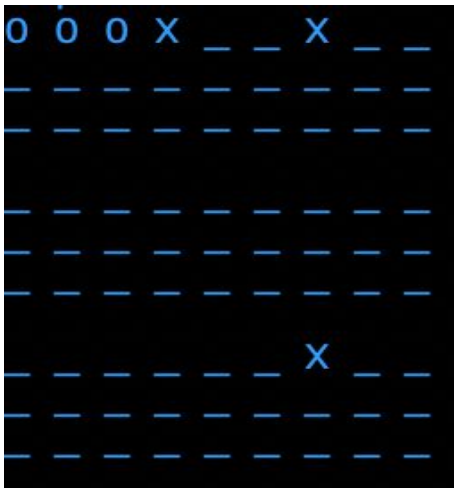
Our initial optimizing idea was to add a few tiebreaker scenarios for tied scores between several possible best moves in our algorithm function. However, since we were limited to only changing our score evaluation (evaluatePredefined) for our own agent, our options in optimizing the defensive player to consistently defeat the offensive agent were reduced to just manipulating how we calculate the score. We first experimented with assigning different point values for the rules with varying degrees of success, but nothing made a noticeable impact. The biggest issue we found with the predefined agents is that they had a singular mindset of only finding their own score on the board after a certain move. To rectify this issue, our agent returns a net positive or negative score that does not rely on whose turn it is to move. It calculates all the points for the max player, adds the cumulative negative score for the min player, and returns that result. For example, if a defensive move at depth 0 would result in the offensive agent winning the game at depth 1, the score returned would be 10000, and in our alphabeta/minimax algorithm, that move would not be chosen by the min player.

Over 20 games, our defensive agent won quickly if it started the game, with the expanded nodes being around 750. When the offensive agent first, we still won each game, but they were far lengthier and it was clear how our agent was taking advantage of the predefined agent's ignorance of its opponent. The expanded nodes in these cases were around 1000-1600 depending on the initial starting board and starting player. Our agent won 100% of the games in our simulation.

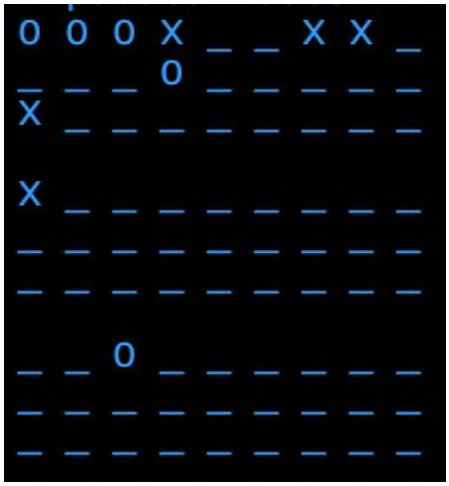
Final Game Board 1:
This game shows our agent's advantage, as you can see that the defensive agent is clearly taking into account the offensive agent's possible next moves. In the top right local board, X played initially in the top corner of that board. Later on, O was taken to that same board, and under the predefined agent, it should have picked a corner for the maximum magnitude score. However, it chooses to pick the left middle cell and make X play on a new board.



Final Game Board 2:
Here, the defensive agent went first and started on the top left board. It predicted that the offensive agent would take the top corner as the most valued cell in each local board it was sent to and won the top left local board this way.



Final Game Board 3:
Here, the defensive agent gets the top left corner and top right corner of local board 0, and is eventually faced with a move to make at the top middle local board. It could choose the right corner or one of the bottom corners to maximize its point value, but it instead takes the left middle cell, knowing that in an empty local board, the offensive agent always takes the top left cell. That would take the defensive agent back to board 0, in which it already has an unblocked two-in-a-row. This results in a defensive win.

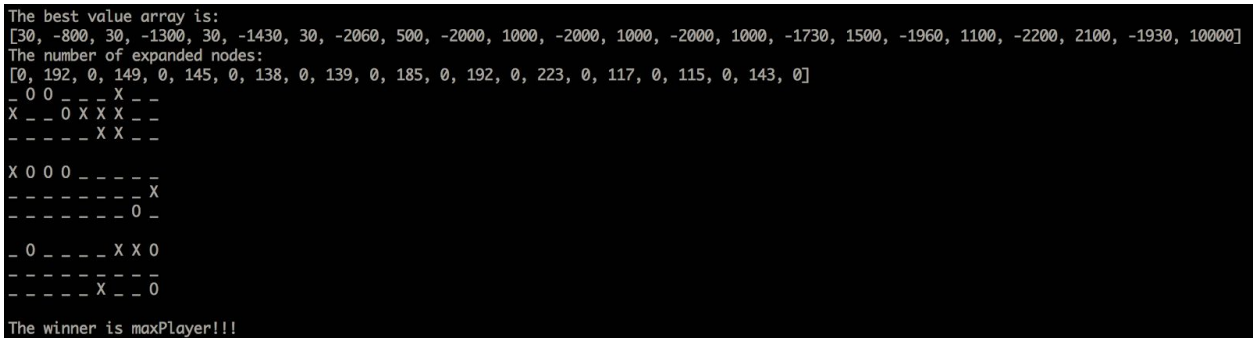


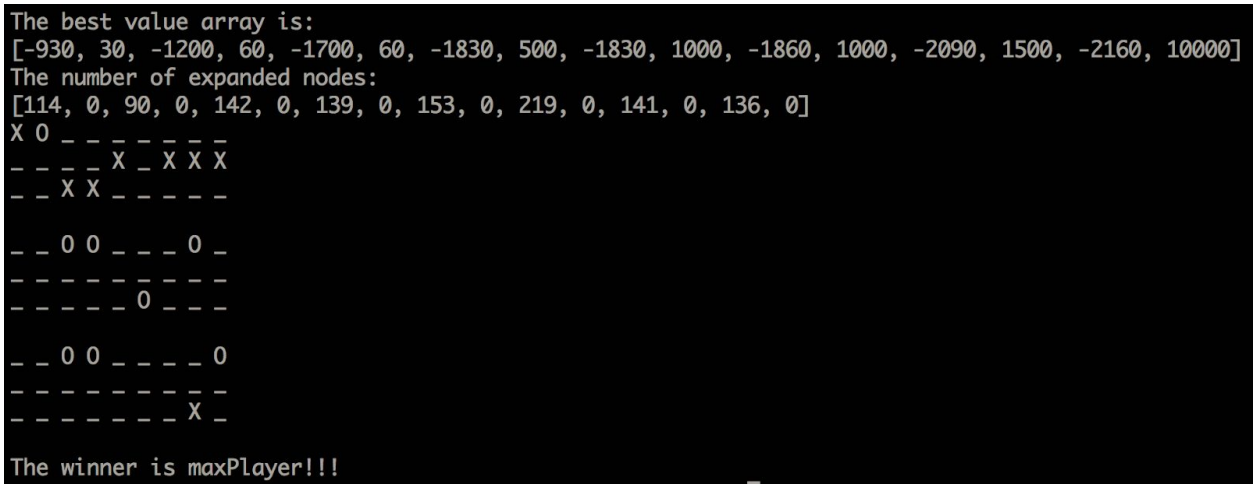
Section IV : Ultimate Tic-Tac-Toe Human vs. Your Agent

For 10 games of human vs our agent, discuss observations and % winning time. Talk abt advantages/disadvantages of our function. Show 3-5 final games boards showing advantage/disadvantage

Our agent was quite smart and strategic while playing against humans. Most games were quite lengthy, as our agent had a tendency to strategically send the human player to more empty boards to reduce the chances of an opposing player getting an unblocked 2-in-a-row. This would often work in its favor, as the human player had a tendency to make more mistakes as the game went on.

Our agent was more successful near the beginning of the game. As the game progressed, and more local boards had two X's in a row, O allowed X to play in a board where there were already two X's in a row allowing X to win. Most occurrences of X winning resulted in that situation. In addition, our agent would have been stronger had we been able to modify our alpha beta function. We would have probably increased our search depth by 1 even though we realize it would take much longer and would not be as fast of an algorithm.





```
The best value array is:
[-930, 0, -1460, 0, -1960, 30, -2090, 500, -2660, 100, -2800, 100, -2730, 600, -3430, 700, -3090, 1700, -2990, 1700, -2720, 2700, -2320, 2700,
-2490, 2800, -3090, 2800, -3860, 2400, -3790, 2400, -2890, 2500, -3860, 2100, -4390, 2200, -4360, 2200, -10000, 2800, -10000]
The number of expanded nodes:
[114, 0, 152, 0, 155, 0, 145, 0, 122, 0, 204, 0, 111, 0, 164, 0, 101, 0, 164, 0, 194, 0, 181, 0, 137, 0, 132, 0, 115, 0, 143, 0, 139, 0, 82, 0,
 100, 0, 92, 0, 57, 0, 51]
X _ X O _ _ _ _ X
O X _ _ X O _ X X
O _ O _ _ X O O O

O X X X X O _ O _
O O _ O O X X O _
_ _ X X _ _ _ _ _

O O _ _ _ O X _ O
X _ _ _ O _ X _ _
_ X _ _ X _ O _ _

The winner is minPlayer!!!
```

```
Your turn! Enter a valid move on board 4
Enter x coordinate (0-8): 2
Enter y coordinate (0-8): 2
Move made!
The best value array is:
[30, -800, 30, -1300, 30, -1430, 60, -1430, 60, -1560, 500, -2060, 100, -2160, 600, -2260, 700, -2760, 1200, -2490, 2200, -2990, 2300, -3960, 2
500, -10000, 10000]
The number of expanded nodes:
[0, 192, 0, 149, 0, 145, 0, 170, 0, 133, 0, 139, 0, 134, 0, 212, 0, 137, 0, 190, 0, 140, 0, 132, 0, 134, 0]
X _ O _ _ _ _ _ X
_ O _ X _ X _ X _
_ _ X _ _ _ O _ X

O O _ _ O O O _ O
_ _ _ _ X _ _ _ _
_ _ _ X X X _ _ _

_ _ _ _ _ _ X _ _
X O X _ O _ _ O _
_ _ _ _ _ _ O _ _

The winner is maxPlayer!!!
```

Statement of Contribution :

Both members of the group thought and worked through both parts of the MP. Jessica and Adarsh coded and debugged part 1 together. On part 2, Jessica wrote the playGamePredefined to actually implement the game and Adarsh detailed the heuristic (evaluatePredefined) as well as several helpers to assist us later on. Jessica coded out the minimax/alpha-beta functions and Adarsh and Jessica wrote the heuristic for our own agent together. Both wrote the report.