Heuristic 1: Score is the ratio of player moves to opponent moves; infinity if there are no opponent moves.
Intuition: This simple heuristic is based on the notion that the number of moves of a player provides more player options and improves chances of player win -- chances of winning directly proportional to player options. Also, chances of player winning is inversely proportional number of opponent's choices. This heuristic performs about the same as the improved ID heuristic on the average. This is likely because the overall intuition behind the score is the same, albeit the computation of the score is different.

Results of 10 runs of tournament.py against ID_improved: wins 8 out of 10 tournament

ID_improved	Student
86.43%	88.57%
88.57%	88.57%
88.57%	92.14%
88.57%	94.29%
90.71%	92.14%
94.29%	92.86%
92.14%	90.71%
92.86%	93.57%
81.43%	87.86%
86.43%	90.71%

Heuristic 2: Return the move that gives the maximum choices to player, and minimum choices to opponent, one move ahead. This is similar to the improved algorithm ID_improved, except it looks one move further ahead. The logic is that the further moves ahead that we examine the chances of winning, the higher the chances of winning. This assumes that the game is not pathological, in the sense that examining further into the future does not decrease the chances of winning, which seems to be true for isolation. The score computed one move ahead is the same as ID_improved, i.e., it picks the score that maximizes the difference between the choices for the player and the opponent.

As a first cut, the score calculated was as follows: for a given player move, score is the maximum of "number of choices for player in next move - number of choices of player for the opponent". This performed rather poorly compared to the improved_ID heuristic. The reason for this is that looking further ahead and ignoring immediate moves ignores the opponents next move, so we may be evaluating the score associated with a search tree of low probability.

So the value of the current set of moves available to the player was incorporated into the score as follows: "(number of current choices for player * number of choices for player in next move given an opponent move) - number of choices for opponent given an opponent move". This significantly improved the performance of the algorithm but was still considerably worse than ID_improved.

Results of 6 runs of tournament.py against ID_improved: wins 0 out of 6 tournaments

ID_Improved	Student
85.71%	83.57%
88.57%	85.00%
89.29%	83.57%
85.71%	84.29%
85.71%	83.71%
92.14%	79.29%

Heuristic 3: This heuristic is based on the technique described in "Game Tree search: Min/Max approximation" by Ronald Rivest. The key difference (which could be significant) is that this heuristic only expands the tree rooted at each examined node for a depth of 2, and then computes the penalties for that sub tree and picks the choice with the lowest penalty.

Pseudocode:

```
    Let position of player L be p and position of opponent Q be p' at Board B
    Use p for maximizing and minimizing players, as maxp and minp, with values 10 and 0.1 respectively
1.  M = B.legal_moves(L)
2. for m in M:
      B' = B.forecast_moves(m)
     M' = B'.legal_moves(P)
     compute score S and S' for players L and P, respectively, using heuristic 1
     determine the weight/penalty for edge using the negative log derivate as suggested
        in page 87,  w(p,p') = -log(len(M)) + (minp-1) * (log(S') - log(S))
     for  m' in M'
        B" = B'.forecast_move(m')  so that P is at position p"  after move M'
        compute score S" for P using heuristic 1
        determine weight(p',p") = -log(len(M')) + (maxp-1) * (log(S") - log(S'))
 3     determine w=min(weight(p,p')+weight(p'+p")) over all m->m'->m" sequences  and return 1.0/w  as score
```

The technique described in the paper propagates the penalties from the leaf nodes all the way back to the root of the subtree for arbitrary depths. However, this cannot be implemented for this project because it requires a different code structure not allowed by the project implementation framework. Specifically, to implement the algorithm described in the paper, we need to keep track of the weights, penalties and shortest subtree path to the leaf node for each node in the tree, and this would require changing the signature of the custom_score function among other differences.

Instead, the score calculated in this heuristic checks one move ahead for the player, i.e., player->opponent->player, and calculates the choices with the least penalty and returns the inverse of the penalty as the score. Note that the limitation here is that we do not look at all the global choices for expansion of a node as required by the algorithm in the paper, but only the choices from the current board state ignoring the history of the game upto the current board. A faithful implementation of the algorithm in the paper would require that the penalties be recalculated all the way up to the root of the tree, affecting the penalties at all non-terminal nodes yet to be expanded.

10 tournaments between ID_improved and this heuristic yielded the following results: wins 5 out 10 tournaments

ID_Improved	Student
92.14%	90.71%
92.14%	87.14%
87.14%	90.00%
90.00%	91.43%
87.86%	87.14%
89.29%	88.57%
88.57%	90.71%
87.86%	90.00%
92.86%	91.43%
87.86%	92.86%

Choice of Heuristic:

Heuristic 1 was chosen as custom score because it is simple to understand and performs consistently better than ID_Improved. Heuristic 3 may well perform better if it could be fully implemented, but the way it is implemented it performs a little worse than the heuristic1 score calculation on which it is based.

Heuristic 2 performs consistently worse than the other two heuristics because the score does not accurately reflect the effect of the immediate future move better than future moves, and this is less effective than heuristic 1 which does that pretty effectively.

Heuristic 3's performance is very similar to that of ID_Improved, even though it is based on heuristic1 – the reason for this is likely due to the fact that it does not do the following important steps mentioned in the paper: 1. Values are not recomputed all the way back to the root at every step, so we miss out potential choices on the rest of the tree 2. We are restricting ourselves to the subtrees rooted at the all the immediate choices for the player in the board, thus ignoring all the remaining leaf nodes.