Heuristic Analysis

Algorithm Description

As a part of this project, I implemented three different heuristic score evaluation functions for use by the Isolation game-playing agent. Their implementation was as follows:

(1) simple_score

The first evaluation function attempted to emulate the simple function described during the video lectures. The idea here is that a given player's score can be approximated by the number of legal moves they have left minus 2 times the number of legal moves left for the opponent. This encourages the AI player to "chase" the opponent around the board, limiting the number of spaces they have available to move.

A special case (game "utility") is checked to determine if we are at an end-game scenario. If the current player has no legal moves left, -infinity is returned to encourage avoiding this state. If the opponent has no legal moves, +infinity is returned to encourage this state.

(2) central_score

The second heuristic implemented is very similar to the first (simple_score) heuristic except that it also factors in how close the player is to the center of the board. The idea being that the closer a player is to the center of the board, the more potential moves they will have available now and in the future. This heuristic function encourages the player to stay more centralized in addition to maximizing their immediate next legal moves and limiting their opponent's legal moves.

(3) partition_score

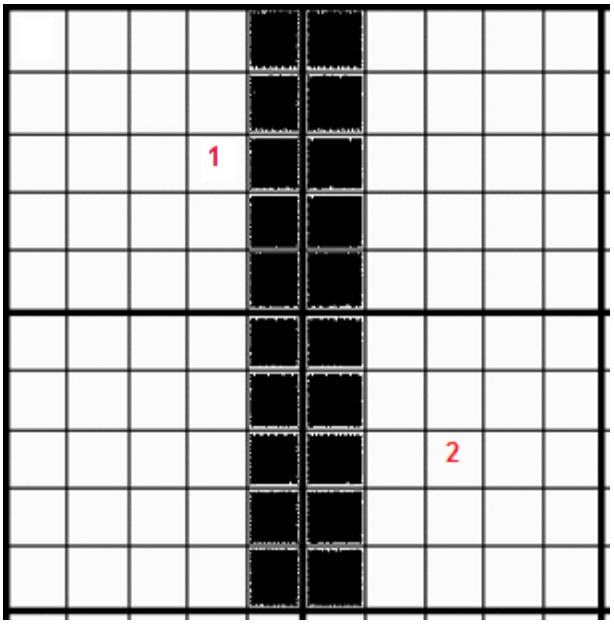
The final evaluation function implemented took a different approach. If at least 2 times the width or height of the game board number of moves have occurred in a game, it will search for "partitions" on the game board. If no partitions are found, then it will default to using the simple_score evaluation heuristic.

As implemented, a "partition" is considered to be either a double-line that spans the width or height of the game board or a single-line that spans the game board that also

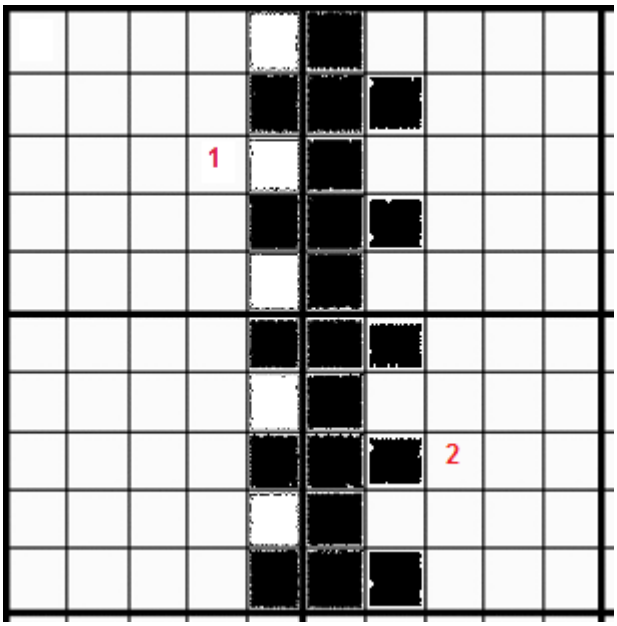
features “crosses” spaced at most two units across that intersect the solid line perpendicularly.

For example:

(a) Double line partition



(b) Line with crosses partition



Such partitions prevent a player on one side of the structure from traversing to the other side, effectively creating “islands” that each player is stuck on. For each partition found, the

algorithm checks to see if the players are divided by it, and if so, it will calculate the number of contiguous squares present on each player's "island". If the current player has a greater number of contiguous squares, then they should win and this state is encouraged with a +infinity score. If the number of squares is tied or the opponent has greater contiguous squares, the opponent should win, so this state is discouraged with a -infinity score. I was able to unit test this algorithm to ensure that it was finding partitions and calculating scenarios correctly.

Algorithm Performance

In practice, each of the algorithms I implemented turned out to perform approximately the same, and similar to the performance of the provided "ID_Improved" algorithm included with tournament.py. Here was an example of one set of runs by tournament.py for each of the evaluation functions on my Lenovo W540 Thinkpad (Intel Core i7-4800 @2.70GHz, 32 GB RAM, Windows 7 64-bit, Python 3).

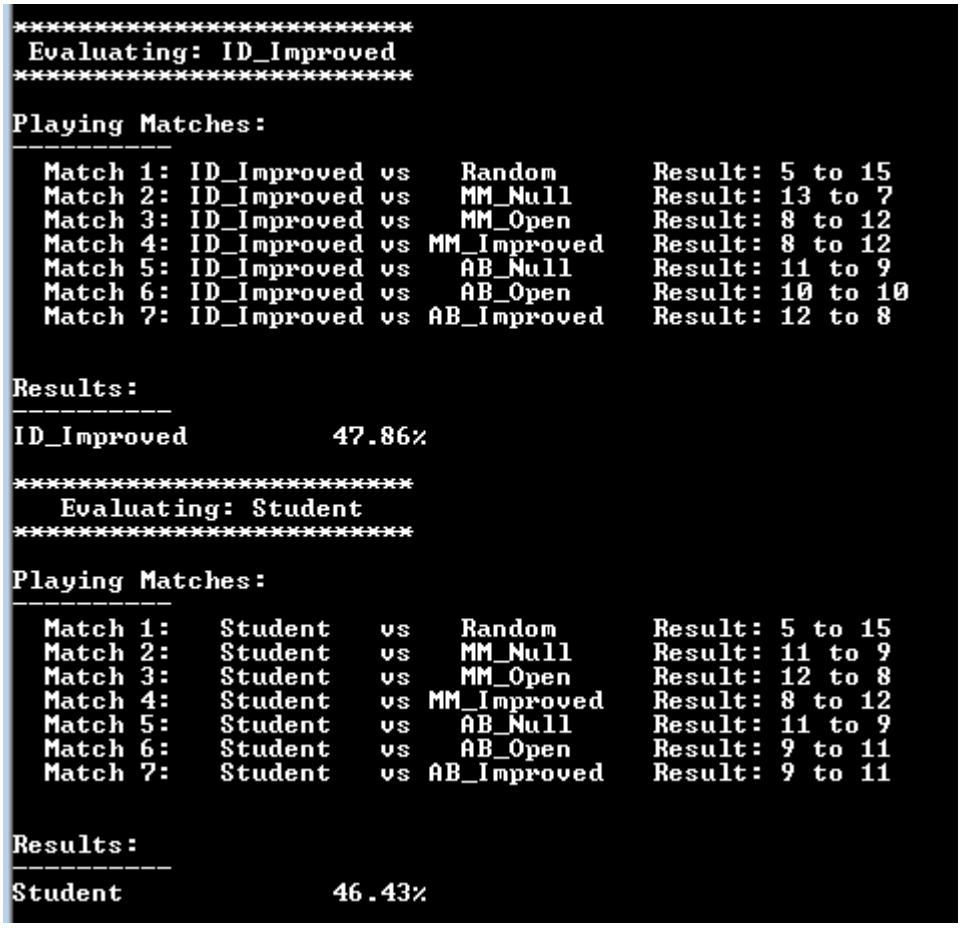
(1) simple_score

```
**************************
 Evaluating: ID_Improved
**************************

Playing Matches:
----------
  Match 1: ID_Improved vs    Random      Result: 1 to 19
  Match 2: ID_Improved vs    MM_Null     Result: 11 to 9
  Match 3: ID_Improved vs    MM_Open     Result: 9 to 11
  Match 4: ID_Improved vs MM_Improved    Result: 5 to 15
  Match 5: ID_Improved vs    AB_Null     Result: 10 to 10
  Match 6: ID_Improved vs    AB_Open     Result: 11 to 9
  Match 7: ID_Improved vs AB_Improved    Result: 8 to 12


Results:
----------
ID_Improved         39.29%

**************************
   Evaluating: Student
**************************

Playing Matches:
----------
  Match 1:   Student    vs    Random      Result: 2 to 18
  Match 2:   Student    vs    MM_Null     Result: 12 to 8
  Match 3:   Student    vs    MM_Open     Result: 10 to 10
  Match 4:   Student    vs MM_Improved    Result: 9 to 11
  Match 5:   Student    vs    AB_Null     Result: 13 to 7
  Match 6:   Student    vs    AB_Open     Result: 9 to 11
  Match 7:   Student    vs AB_Improved    Result: 10 to 10


Results:
----------
Student             46.43%
```

(2) central_score

```
**************************
 Evaluating: ID_Improved
**************************

Playing Matches:
----------
  Match 1: ID_Improved vs   Random      Result: 3 to 17
  Match 2: ID_Improved vs   MM_Null     Result: 11 to 9
  Match 3: ID_Improved vs   MM_Open     Result: 12 to 8
  Match 4: ID_Improved vs MM_Improved   Result: 8 to 12
  Match 5: ID_Improved vs   AB_Null     Result: 10 to 10
  Match 6: ID_Improved vs   AB_Open     Result: 10 to 10
  Match 7: ID_Improved vs AB_Improved   Result: 11 to 9


Results:
----------
ID_Improved         46.43%

**************************
   Evaluating: Student
**************************

Playing Matches:
----------
  Match 1:   Student   vs   Random      Result: 4 to 16
  Match 2:   Student   vs   MM_Null     Result: 11 to 9
  Match 3:   Student   vs   MM_Open     Result: 12 to 8
  Match 4:   Student   vs MM_Improved   Result: 6 to 14
  Match 5:   Student   vs   AB_Null     Result: 9 to 11
  Match 6:   Student   vs   AB_Open     Result: 10 to 10
  Match 7:   Student   vs AB_Improved   Result: 11 to 9


Results:
----------
Student             45.00%
```

(3) partition_score


****
Evaluating: ID_Improved
****
Playing Matches:
Match 1: ID_Improved vs Random Result: 5 to 15
Match 2: ID_Improved vs MM_Null Result: 13 to 7
Match 3: ID_Improved vs MM_Open Result: 8 to 12
Match 4: ID_Improved vs MM_Improved Result: 8 to 12
Match 5: ID_Improved vs AB_Null Result: 11 to 9
Match 6: ID_Improved vs AB_Open Result: 10 to 10
Match 7: ID_Improved vs AB_Improved Result: 12 to 8
Results:
ID_Improved 47.86%
****
Evaluating: Student
****
Playing Matches:
Match 1: Student vs Random Result: 5 to 15
Match 2: Student vs MM_Null Result: 11 to 9
Match 3: Student vs MM_Open Result: 12 to 8
Match 4: Student vs MM_Improved Result: 8 to 12
Match 5: Student vs AB_Null Result: 11 to 9
Match 6: Student vs AB_Open Result: 9 to 11
Match 7: Student vs AB_Improved Result: 9 to 11
Results:
Student 46.43%