AIND Isolation – Heuristic Analysis

Analyzed the following custom score heuristics for Isolation projects. Please see Appendix for the implementation of the heuristic and results of the tournament.

1. **NumberOfOpponentVsMyMoves**:
    a. This score function returns the difference between number of moves available for self and opponent. A weighted factor is applied to the sum of self moves and opponent moves. Different weights of less than, greater than and around 0.5 were tried.
2. **KeepYourDistance**:
    a. This score function returns the difference between the number of moves available to self and opponent. Great distance between the two is rewarded with a higher score.
3. **KeepYourEnemiesCloser**:
    a. Keep your opponent close. Reward smaller difference in distance between the location vectors of minimizing and maximizing players with a higher score.
4. **ForecastAndLookAhead**:
    a. Forecast next moves for both maximizing and minimizing players and look ahead at the possible moves. Applied equal weight to maximizing player's current moves and future moves. Also tried testing with varying weights of less than 0.5 and more than 0.5 to emphazise current and future states. Did not affect the results by much, student and ID_improved both performed similarly.
5. **GetOutOfCorner**:
    a. Penalize corner moves for maximizing player and reward corner moves for minimizing player. If we are nearing the end of the game, the penalty/reward factor is greater. Student underperformed in testing.

Recommended Evaluation Function:

Of the 5 evaluation functions considered below, I would recommend the NumberOfOpponentMovesVsSelfMoves function. Apply a weight of around 0.5 to self moves and penalizing opponent for more moves outperformed all other functions. Notice ForecastAndLookAhead is a close competitor. However it does not execute in linear time.

Reasons for this recommendation include:
1. Performance: Data in above table shows this simple heuristic outperformed all others.
2. Complexity: This is of O(1) complexity and computes in linear time.
3. Readability and Maintainability: Code is easy to implement, read, debug and maintain.

Evaluation Function	**ID_Improved**	**Student**	**Student wins/ID_improved**
KeepYourDistance	67.86	63.57	0.94
KeepYourEnemiesCloser	65.71	60.71	0.92
ForecastAndLookAhead	72.14	72.83	1.01
GetOutOfCorner	70.06	65.71	0.94
NumberOfOpponentVsMyMoves(weight 0.45)	**71.43**	**74.57**	**1.04**
NumberOfOpponentVsMyMoves(weight 0.3)	68.57	69.29	1.01
NumberOfOpponentVsMyMoves(weight 0.7)	68.59	67.86	0.99

APPENDIX

```python
def keepYourDistance(game, player):
    """
        Keep your distance from opponent. Reward larger difference in distance
        between the location vectors of maximizing and minimizing players with higher score.
        """
    if game.is_loser(player):
        return float("-inf")
    if game.is_winner(player):
        return float("inf")
    oppLoc = game.get_player_location(game.get_opponent(player))
    if oppLoc == None:
        return float(0)
    myLoc = game.get_player_location(player)
    if myLoc == None:
        return float(0)
    return float(abs(sum(oppLoc) - sum(myLoc)))
```

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

Playing Matches:
----------
  Match 1: ID_Improved vs   Random    Result: 18 to 2
  Match 2: ID_Improved vs   MM_Null   Result: 16 to 4
  Match 3: ID_Improved vs   MM_Open   Result: 11 to 9
  Match 4: ID_Improved vs MM_Improved Result: 12 to 8
  Match 5: ID_Improved vs   AB_Null   Result: 13 to 7
  Match 6: ID_Improved vs   AB_Open   Result: 13 to 7
  Match 7: ID_Improved vs AB_Improved Result: 12 to 8


Results:
----------
ID_Improved         67.86%

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

Playing Matches:
----------
  Match 1:   Student   vs   Random    Result: 17 to 3
  Match 2:   Student   vs   MM_Null   Result: 9 to 11
  Match 3:   Student   vs   MM_Open   Result: 17 to 3
  Match 4:   Student   vs MM_Improved Result: 12 to 8
  Match 5:   Student   vs   AB_Null   Result: 15 to 5
  Match 6:   Student   vs   AB_Open   Result: 11 to 9
  Match 7:   Student   vs AB_Improved Result: 8 to 12


Results:
----------
Student             63.57%
```

```python
def keepYourEnemiesCloser(game, player):
    """
    Keep your opponent close. Reward smaller difference in distance
    between the location vectors of minimizing and maximizing players
    with a higher score.
    """
    if game.is_loser(player):
        return float("-inf")
    if game.is_winner(player):
        return float("inf")
    oppLoc = game.get_player_location(game.get_opponent(player))
    if oppLoc == None:
        return 0.
    myLoc = game.get_player_location(player)
    if myLoc == None:
        return 0.
    return float(-abs(sum(oppLoc) - sum(myLoc)))
```

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

Playing Matches:
----------
  Match 1: ID_Improved vs   Random    Result: 18 to 2
  Match 2: ID_Improved vs   MM_Null   Result: 13 to 7
  Match 3: ID_Improved vs   MM_Open   Result: 10 to 10
  Match 4: ID_Improved vs MM_Improved Result: 14 to 6
  Match 5: ID_Improved vs   AB_Null   Result: 16 to 4
  Match 6: ID_Improved vs   AB_Open   Result: 11 to 9
  Match 7: ID_Improved vs AB_Improved Result: 10 to 10


Results:
----------
ID_Improved         65.71%

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

Playing Matches:
----------
  Match 1:   Student   vs   Random    Result: 16 to 4
  Match 2:   Student   vs   MM_Null   Result: 14 to 6
  Match 3:   Student   vs   MM_Open   Result: 10 to 10
  Match 4:   Student   vs MM_Improved Result: 10 to 10
  Match 5:   Student   vs   AB_Null   Result: 13 to 7
  Match 6:   Student   vs   AB_Open   Result: 13 to 7
  Match 7:   Student   vs AB_Improved Result: 9 to 11


Results:
----------
```

```
Student              60.71%
```

```python
def numberOfOpponentVsMyMoves(game, player): # Weight 0.45
    """
    This score function returns the difference
    between the number of moves available for self and the opponent player.
    Add a weighted factor to the sum of own moves and the opponents moves.
    """

    if game.is_loser(player):
        return float("-inf")

    if game.is_winner(player):
        return float("inf")

    weight = 0.45

    ownMoves = len(game.get_legal_moves(player))
    oppMoves = len(game.get_legal_moves(game.get_opponent(player)))
    return float(weight * ownMoves + (1 - weight) * (-oppMoves))
```

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

Playing Matches:
----------
  Match 1: ID_Improved vs   Random    Result: 18 to 2
  Match 2: ID_Improved vs   MM_Null   Result: 18 to 2
  Match 3: ID_Improved vs   MM_Open   Result: 12 to 8
  Match 4: ID_Improved vs MM_Improved Result: 14 to 6
  Match 5: ID_Improved vs   AB_Null   Result: 17 to 3
  Match 6: ID_Improved vs   AB_Open   Result: 12 to 8
  Match 7: ID_Improved vs AB_Improved Result: 9 to 11


Results:
----------
ID_Improved          71.43%

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

Playing Matches:
----------
  Match 1:   Student   vs   Random    Result: 18 to 2
  Match 2:   Student   vs   MM_Null   Result: 18 to 2
  Match 3:   Student   vs   MM_Open   Result: 13 to 7
  Match 4:   Student   vs MM_Improved Result: 12 to 8
  Match 5:   Student   vs   AB_Null   Result: 17 to 3
  Match 6:   Student   vs   AB_Open   Result: 13 to 7
  Match 7:   Student   vs AB_Improved Result: 12 to 8


Results:
```

```
----------
Student             74.57%
```

```
def numberOfOpponentVsMyMoves(game, player): # Weight 0.3
    """
    This score function returns the difference
    between the number of moves available for self and the opponent player.
    Add a weighted factor to the sum of own moves and the opponents moves.
    """

    if game.is_loser(player):
        return float("-inf")

    if game.is_winner(player):
        return float("inf")

    weight = 0.3

    ownMoves = len(game.get_legal_moves(player))
    oppMoves = len(game.get_legal_moves(game.get_opponent(player)))
    return float(weight * ownMoves + (1 - weight) * (-oppMoves))
```

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

Playing Matches:
----------
  Match 1: ID_Improved vs   Random    Result: 15 to 5
  Match 2: ID_Improved vs   MM_Null   Result: 16 to 4
  Match 3: ID_Improved vs   MM_Open   Result: 13 to 7
  Match 4: ID_Improved vs MM_Improved Result: 12 to 8
  Match 5: ID_Improved vs   AB_Null   Result: 16 to 4
  Match 6: ID_Improved vs   AB_Open   Result: 12 to 8
  Match 7: ID_Improved vs AB_Improved Result: 12 to 8


Results:
----------
ID_Improved         68.57%

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

Playing Matches:
----------
  Match 1:   Student   vs   Random    Result: 18 to 2
  Match 2:   Student   vs   MM_Null   Result: 16 to 4
  Match 3:   Student   vs   MM_Open   Result: 14 to 6
  Match 4:   Student   vs MM_Improved Result: 12 to 8
  Match 5:   Student   vs   AB_Null   Result: 15 to 5
  Match 6:   Student   vs   AB_Open   Result: 11 to 9
  Match 7:   Student   vs AB_Improved Result: 11 to 9
```

```
Results:
----------
Student              69.29%
```

```python
def numberOfOpponentVsMyMoves(game, player): # Weight 0.7
    """
    This score function returns the difference
    between the number of moves available for self and the opponent player.
    Add a weighted factor to the sum of own moves and the opponents moves.
    """

    if game.is_loser(player):
        return float("-inf")

    if game.is_winner(player):
        return float("inf")

    weight = 0.7

    ownMoves = len(game.get_legal_moves(player))
    oppMoves = len(game.get_legal_moves(game.get_opponent(player)))
    return float(weight * ownMoves + (1 - weight) * (-oppMoves))
```

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

Playing Matches:
----------
  Match 1: ID_Improved vs   Random    Result: 15 to 5
  Match 2: ID_Improved vs   MM_Null   Result: 16 to 4
  Match 3: ID_Improved vs   MM_Open   Result: 12 to 8
  Match 4: ID_Improved vs MM_Improved Result: 12 to 8
tournament.py:100: UserWarning: One or more agents lost a match this round
due to timeout. The get_move() function must return before time_left()
reaches 0 ms. You will need to leave some time for the function to return,
and may need to increase this margin to avoid timeouts during  tournament
play.
  warnings.warn(TIMEOUT_WARNING)
  Match 5: ID_Improved vs   AB_Null   Result: 10 to 10
  Match 6: ID_Improved vs   AB_Open   Result: 16 to 4
  Match 7: ID_Improved vs AB_Improved Result: 15 to 5


Results:
----------
ID_Improved          68.59%

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

Playing Matches:
----------
  Match 1:   Student   vs   Random    Result: 19 to 1
  Match 2:   Student   vs   MM_Null   Result: 14 to 6
  Match 3:   Student   vs   MM_Open   Result: 11 to 9
  Match 4:   Student   vs MM_Improved Result: 10 to 10
  Match 5:   Student   vs   AB_Null   Result: 15 to 5
  Match 6:   Student   vs   AB_Open   Result: 11 to 9
  Match 7:   Student   vs AB_Improved Result: 15 to 5


Results:
----------
Student              67.86%
```

```python
def forecastAndLookAhead(game, player):
    """ Look ahead moves for both maximizing and minimizing players.
    """
    if game.is_loser(player):
        return float("-inf")

    if game.is_winner(player):
        return float("inf")

    ownMoves = game.get_legal_moves(player)
    oppMoves = game.get_legal_moves(game.get_opponent(player))

    weight = 0.5

    if len(ownMoves) == 0:
        return float("-inf")

    lookAheadMyMoves = 0.0
    lookaheadOppMoves = 0.0

    for move in ownMoves:
        # Look ahead self moves and add it to existing move
        lookAheadMyMoves += len(game.forecast_move(move).get_legal_moves(player))
        # Look ahead opp moves at the next level.
        lookaheadOppMoves += 
len(game.forecast_move(move).get_legal_moves(game.get_opponent(player)))
    # Average moves self player has as a % of my moves
    lookAheadMyMoves = lookAheadMyMoves/len(ownMoves)
    # Average moves opponent has as a % of my moves
    lookaheadOppMoves = lookaheadOppMoves/len(ownMoves)

    if lookAheadMyMoves == 0:
        pass

    score = weight * (len(ownMoves) - len(oppMoves)) + (1 - weight) * (lookAheadMyMoves - 
lookaheadOppMoves)
    return float(score)
```

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

Playing Matches:
----------
  Match 1: ID_Improved vs   Random    Result: 19 to 1
  Match 2: ID_Improved vs   MM_Null   Result: 14 to 6
  Match 3: ID_Improved vs   MM_Open   Result: 15 to 5
  Match 4: ID_Improved vs MM_Improved Result: 12 to 8
  Match 5: ID_Improved vs   AB_Null   Result: 10 to 10
  Match 6: ID_Improved vs   AB_Open   Result: 15 to 5
  Match 7: ID_Improved vs AB_Improved Result: 16 to 4


Results:
----------
ID_Improved         72.14%

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

Playing Matches:
----------
  Match 1:   Student   vs   Random    Result: 18 to 2
  Match 2:   Student   vs   MM_Null   Result: 19 to 1
  Match 3:   Student   vs   MM_Open   Result: 12 to 8
  Match 4:   Student   vs MM_Improved Result: 13 to 7
  Match 5:   Student   vs   AB_Null   Result: 14 to 6
  Match 6:   Student   vs   AB_Open   Result: 12 to 8
  Match 7:   Student   vs AB_Improved Result: 13 to 7


Results:
----------
Student             72.83%
```

```python
def getOutOfCorner(game, player):
    """
    For the remaining moves, penalize corner moves for self and
    reward corner moves for opponent. If the number of remaining blank
    space is about 20% of the board increase the penalty/reward factor.
    """
    if game.is_loser(player):
        return float("-inf")
    if game.is_winner(player):
        return float("inf")

    penaltyRewardFactor = 1
    # If you are in the corner as the game is closer to the end penalize heavily
    if len(game.get_blank_spaces()) < (game.width * game.height / 5.):
        penaltyRewardFactor = 5

    # corners
    corners = [(0, 0),(0, (game.width - 1)),
               ((game.height - 1), 0),((game.height - 1), (game.width - 1))]

    ownMoves = game.get_legal_moves(player)
    selfInCorner = [move for move in ownMoves if move in corners]
    oppMoves = game.get_legal_moves(game.get_opponent(player))
    oppInCorner = [move for move in oppMoves if move in corners]

    #Penalize self for being in the corner, reward opponent for being in the corner
    return float(len(ownMoves) - (penaltyRewardFactor * len(selfInCorner))
                 - len(oppMoves) + (penaltyRewardFactor * len(oppInCorner)))
```

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

Playing Matches:
----------
  Match 1: ID_Improved vs   Random    Result: 17 to 3
  Match 2: ID_Improved vs   MM_Null   Result: 14 to 6
  Match 3: ID_Improved vs   MM_Open   Result: 15 to 5
  Match 4: ID_Improved vs MM_Improved Result: 12 to 8
  Match 5: ID_Improved vs   AB_Null   Result: 15 to 5
  Match 6: ID_Improved vs   AB_Open   Result: 10 to 10
  Match 7: ID_Improved vs AB_Improved Result: 15 to 5


Results:
----------
ID_Improved         70.06%

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

Playing Matches:
----------
  Match 1:   Student   vs   Random    Result: 15 to 5
  Match 2:   Student   vs   MM_Null   Result: 13 to 7
  Match 3:   Student   vs   MM_Open   Result: 14 to 6
  Match 4:   Student   vs MM_Improved Result: 15 to 5
  Match 5:   Student   vs   AB_Null   Result: 14 to 6
  Match 6:   Student   vs   AB_Open   Result: 11 to 9
  Match 7:   Student   vs AB_Improved Result: 10 to 10


Results:
----------
Student             65.71%
```