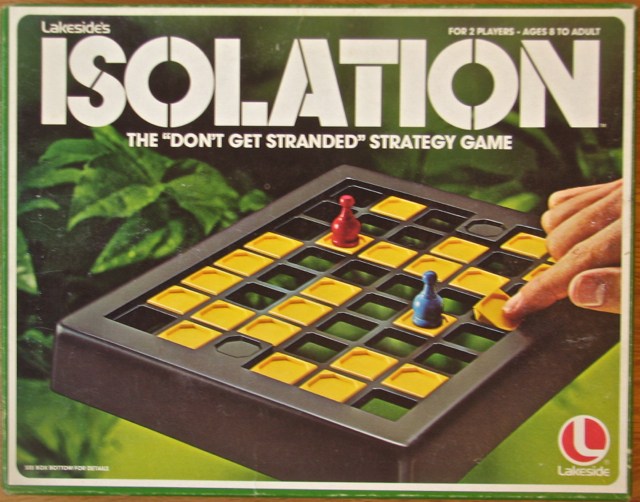
Heuristic Analysis Report for Adversarial Game Playing Agent for Isolation

By Tejasvi Nuthalapati

As part of **Artificial Intelligence Nano Degree (AIND)** at **Udacity**



INDEX OF CONTENTS

|  |  |  |
| --- | --- | --- |
| SNO | Title | Page No. |
| 1 | **Introduction** | 4 |
| 2 | **Evaluation** | 4 |
| 2.1 | **Opponent Heuristics** | 5 |
| 2.2 | **Student Heuristics** | 6 |
| 3 | **Analysis** | 7 |
| 4 | **Conclusion** | 8 |
| 5 | **Appendix** | 9 |

INTENTIONALLY LEFT BLANK

# **INTRODUCTION**

Isolation is a deterministic, two player game where players alternatively take turns to move their pieces from one cell to the other, on occupying a cell, that cell is deactivated/blocked for further moves. As the game proceeds the last player who has no more moves loses the game. In this project the moves of the pieces are ‘L’ shaped like the Knights in chess. The agent can move to any open cell on the board in the form of 2-row + 1-column or 2-column + 1-row with restrictions at the edges without wrap-around but they can jump over blocked or occupied cell. In this game there is a Time Limit on each turn to search a best move and play that player’s turn. On time limit expiry that player loses the match letting the opponent win.

## **EVALUATION**

In the code base of AIND-Isolation, the files

* **game\_agent.py -** Has the Heuristics, Minimax, Alpha-Beta algorithms and Iterative Deepening Search
* **MinimaxPlayer.**minimax() - implements the full recursive search procedure described in lecture ([AIMA Minimax Decision](https://github.com/aimacode/aima-pseudocode/blob/master/md/Minimax-Decision.md)).
* **AlphaBetaPlayer**.alphabeta() - method to implement the full recursive search procedure described in lecture (ref. [AIMA Alpha-Beta Search](https://github.com/aimacode/aima-pseudocode/blob/master/md/Alpha-Beta-Search.md))
* **AlphaBetaPlayer.get\_move()**to implement Iterative Deepening. (ref. [AIMA Iterative Deepening Search](https://github.com/aimacode/aima-pseudocode/blob/master/md/Iterative-Deepening-Search.md))
* custom\_score() , custom\_score\_2(), custom\_score\_3() – **Student** heuristics
* **tournament.py -** script is used to evaluate the effectiveness of my (named "**Student**" in the tournament) custom heuristics. The script measures relative performance of my agent in a round-robin tournament against several other pre-defined agents. The **Student** agent uses time-limited Iterative Deepening along with my custom heuristics.

**\*END GOAL: Develop a heuristic that makes ‘Student’ outperform ID\_Improved.**

**2.1** **Opponent Heuristics** in this tournament are:

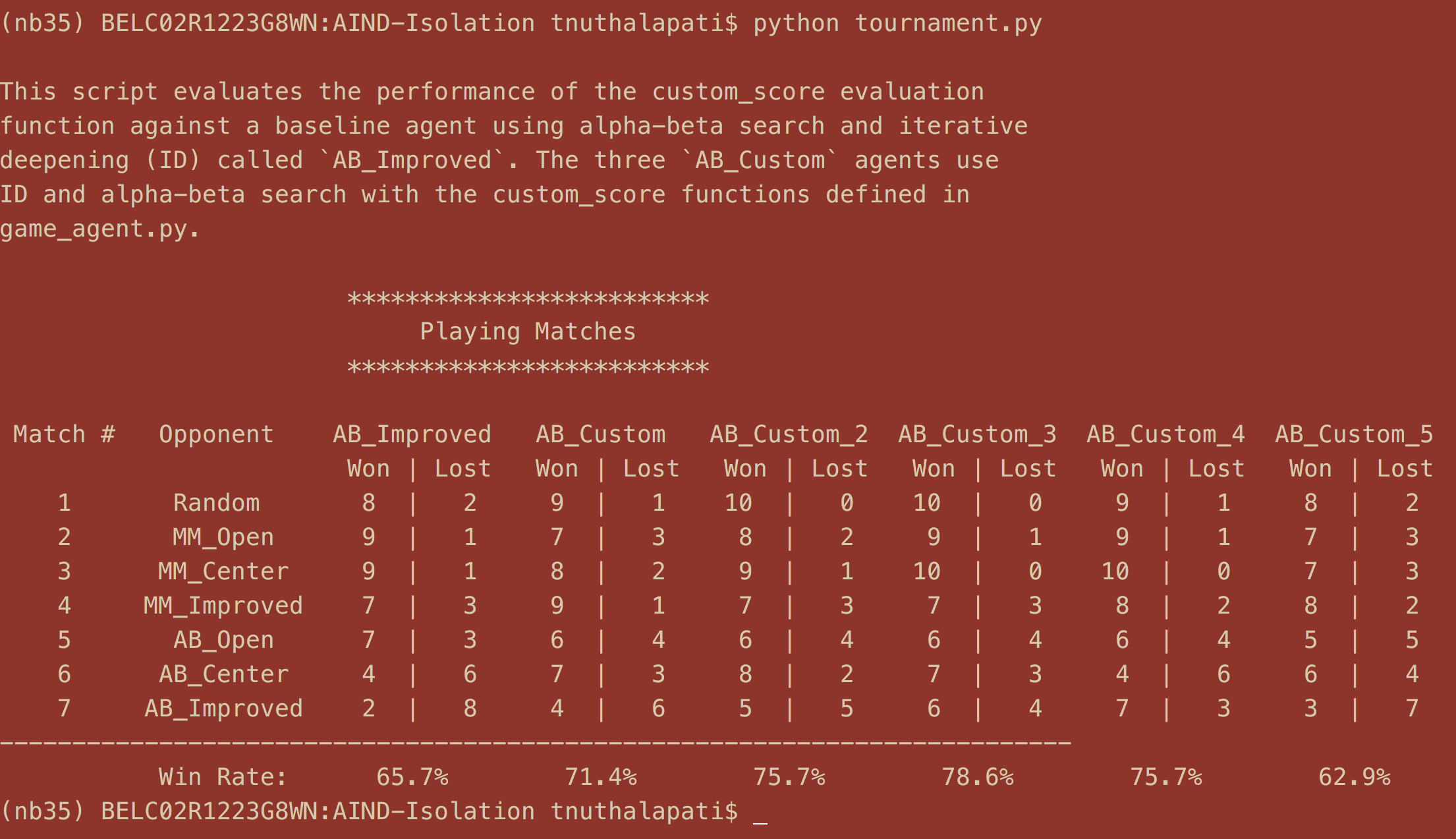
|  |  |  |
| --- | --- | --- |
| SNO | ­Heuristic | Description |
| 1 | **Random** | An agent that randomly chooses a move each turn |
| 2 | **MM\_Open** | MinimaxPlayer agent using the open\_move\_score heuristic with search depth 3 |
| 3 | **MM\_Center:** | MinimaxPlayer agent using the center\_score heuristic with search depth 3 |
| 4 | **MM\_Improved** | MinimaxPlayer agent using the improved\_score heuristic with search depth 3 |
| 5 | **AB\_Open** | AlphaBetaPlayer using iterative deepening alpha-beta search and the open\_move\_score heuristic |
| 6 | **AB\_Center** | AlphaBetaPlayer using iterative deepening alpha-beta search and the center\_score heuristic |
| 7 | **AB\_Improved** | AlphaBetaPlayer using iterative deepening alpha-beta search and the improved\_score heuristic |

**2.2 Student** **Heuristics** in this tournament are:

|  |  |  |
| --- | --- | --- |
| SNO | Heuristic | Description |
| 1 | **Heuristic\_1** | Increases the Weighted Chance Heuristic of ‘Student’ agent  = my\_moves^2 – 1.5\*opp\_moves^2 |
| 2 | **Heuristic\_2** | This Heuristic maximizes the winning chance of the ‘Student’ agent  = my\_moves/opp\_moves |
| 3 | **Heuristic\_3** | This Heuristic reduces the loosing chance of the ‘Student’ agent  = - opp\_moves/my\_moves |
| 4 | **Heuristic\_4** | This Heuristic maximizes the winning chance of the ‘Student’ agent  = my\_moves\* my\_moves /opp\_moves |
| 5 | **Heuristic\_5** | This Heuristic maximizes the winning chance of the ‘Student’ agent  = 3\*my\_moves\* my\_moves /opp\_moves |

1. **ANALYSIS**

**3.1 This graph represents the Tournament analysis with winning ratios for 10 games average against each of the Agents**



**3.2 Console output of the Tournament Analysis**

1. **CONCLUSION**

Most of the heuristics performed significantly better than the **AB\_Improved** heuristic, where the **winning heuristic ratio AB\_custom\_3** outperforms of them all with a winning 78.6% of the matches.

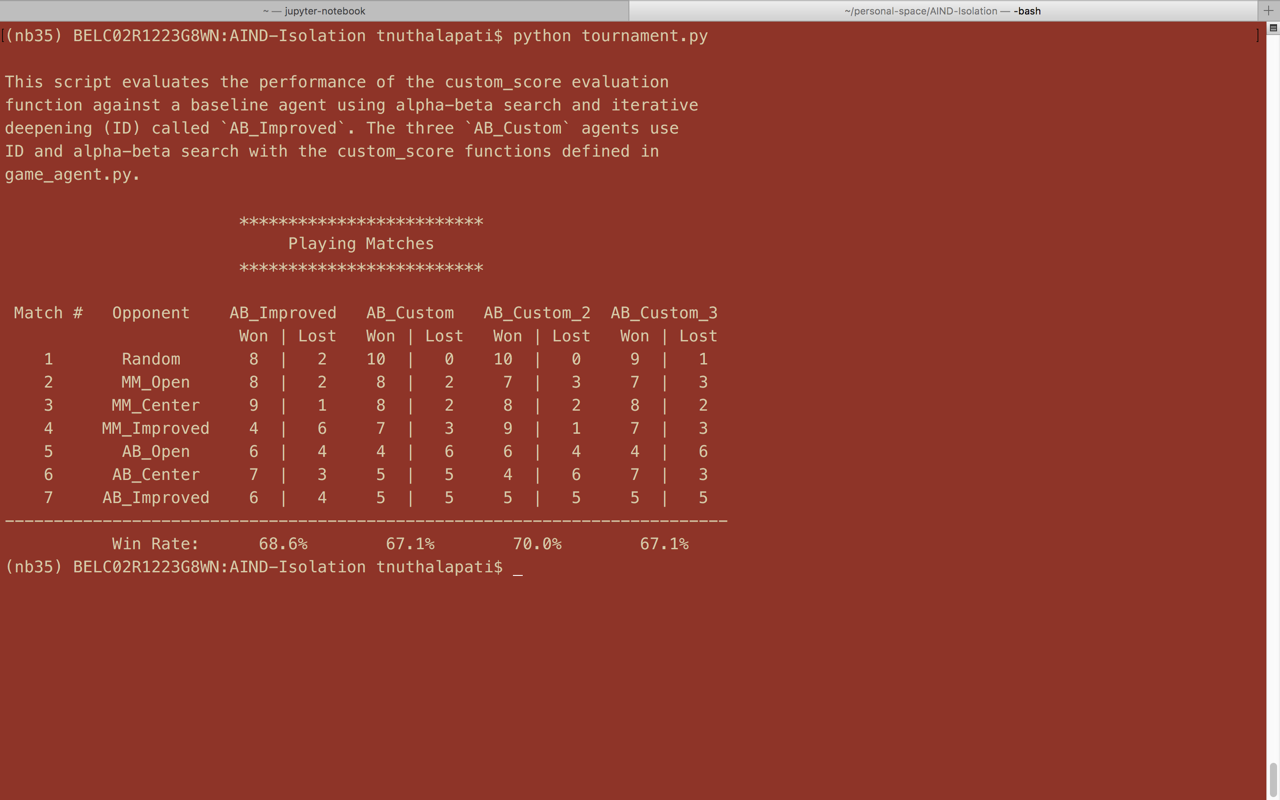
I recommend using AB\_custom\_3 heuristics as:

* It outperforms others with a 78.6% winning chance
* It’s quick and easy to implement
* The heuristic always focuses on reducing the loosing chance which in-turn helps to increase the winning chance
* This is a simple and fast heuristic which proceeds deeper in the game tree
* No strings attached, it means the heuristic doesn’t make any assumptions about the previous/next state but just based on the current state of the player

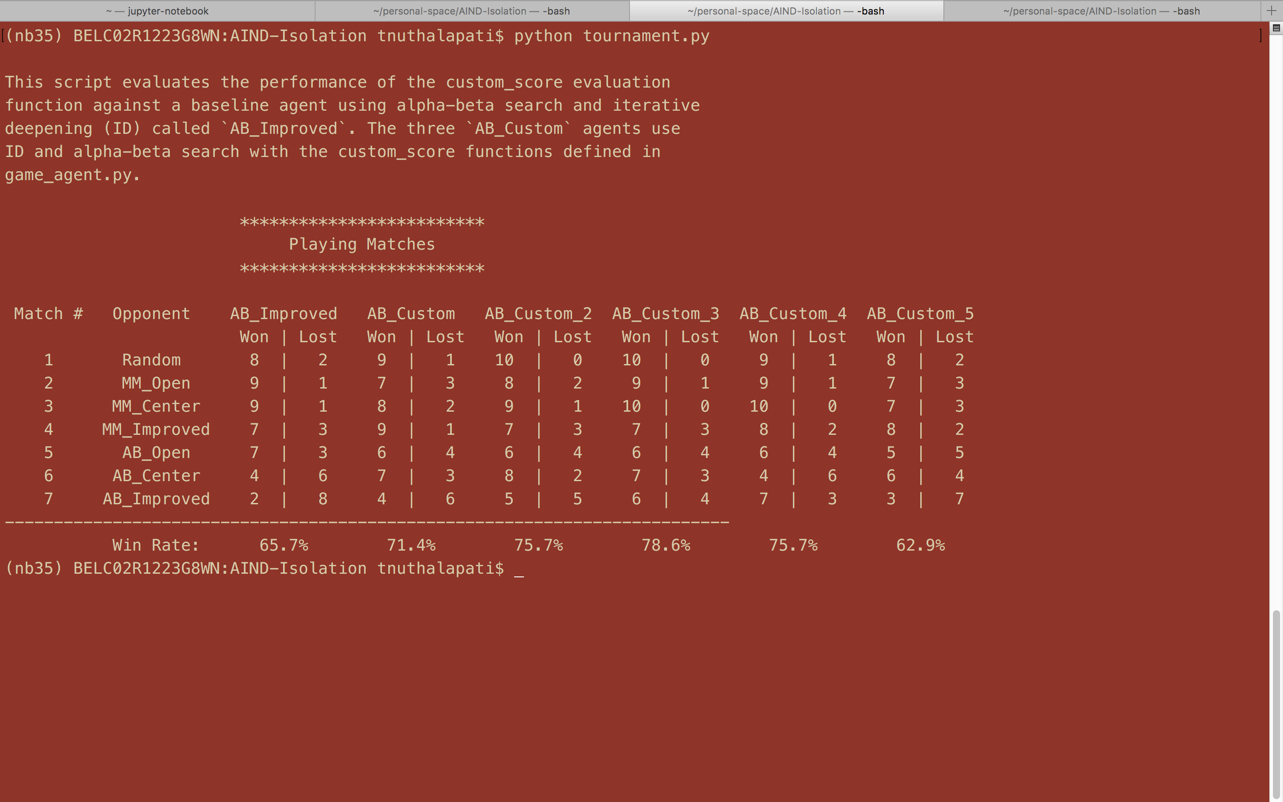
This result is dependent upon the Hardware performance but my assumption after running comparatively on a Windows/Mac PC the results slightly differ but never significantly. By this analysis and project implementation I can conclude that using my Student AB\_Custom\_3 heuristic there are 78.6% of chances of winning a tournament.

1. APPENDIX:

Other Analysis reports performed before concluding the above results.



**5.1 First report generated after successful execution of the code**



* 1. **Second report generated after minor tweeks to the old heuristics and introducing the new ones**