Github: <https://github.com/aknandi/backgammon>

**Player 1: Autonomous Player using weights**

Create a python autonomous player that chooses moves based on an empirically identified novel weighting function. The goal is to identify a weight function that can beat CompareAllMovesWeightingDistance and MoveFurthestBackStrategy statistically 50% of the time or more (run in batches of 100 and compare mean and standard deviation). The existing board features available via the src/compare-all-moves-strategy.py:assess\_board are:

* 'number\_occupied\_spaces': number\_occupied\_spaces,
* 'opponents\_taken\_pieces': opponents\_taken\_pieces,
* 'sum\_distances': sum\_distances,
* 'sum\_distances\_opponent': sum\_distances\_opponent,
* 'number\_of\_singles': number\_of\_singles,
* 'sum\_single\_distance\_away\_from\_home': sum\_single\_distance\_away\_from\_home,
* 'pieces\_on\_board': pieces\_on\_board,
* 'sum\_distances\_to\_endzone': sum\_distances\_to\_endzone

**Player 2: Autonomous player with novel state feature**

Create python class to run completely autonomous game using a novel board feature. Identify, code, and test the use of a board feature that is useful for choosing the next move. For instance, one of the existing features is the sum of the distances to the end zone. Progress can be measured as how much movement is needed to win the game. Once the novel feature has been completed, use it in the weighting function for the evaluating the board states resulting from candidate moves.

**Part 3 (graduate students only)**

Given the board state below, produce a game tree that looks 2 moves ahead. Using minimax and the evaluate\_board function in src/compare\_all\_moves\_strategy.py:CompareAllMovesWeightingDistance, identify white’s optimal move. You can write it by hand or you can write a program to generate it.

Table

Description automatically generated

**Turn in:**

Source code

- New python file named <crimson-id>.py with both player classes (named player1\_<crimson\_id> and player2\_<crimson\_id>) with weighting functions

- Updated python compare\_all\_moves\_strategy.py with new feature.

PDF Report

- Listing of two python files

- Explanation of novel feature

- Comparison of 5 best weighting functions

- Table comparing two player clients to CompareAllMovesWeightingDistance and MoveFurthestBackStrategy

- Game tree produced in part 3

Hints: Creating and testing an autonomous player

1. Create the python players in a copy of src/anderson.py

Graphical user interface, text, application

Description automatically generated

1. Create the main game driver in a copy of auto.py

Text

Description automatically generated

1. Update src/strategy\_factory.py

Text

Description automatically generated

1. Test players

python auto.py CompareAllMovesWeightingDistance CompareAllMovesWeightingDistanceAndSinglesWithEndGame

Hint: Add new feature

1. Update *assess\_board* in *compares\_all\_moves\_strategy.py* to calculate and return the new value
2. Create a new player that using the new value in the weighting function

Graphical user interface, text, application

Description automatically generated