How to Run

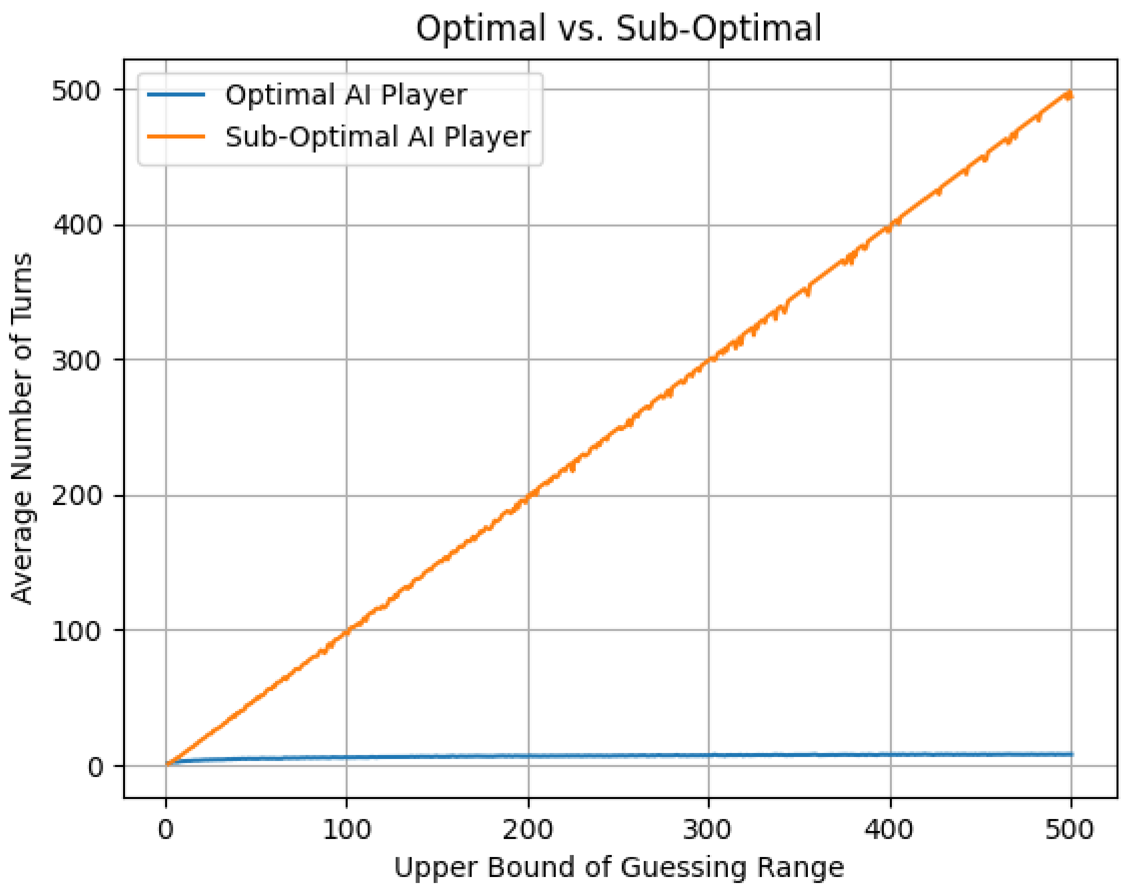
To run this program from the terminal, enter `python Lab8.py`.

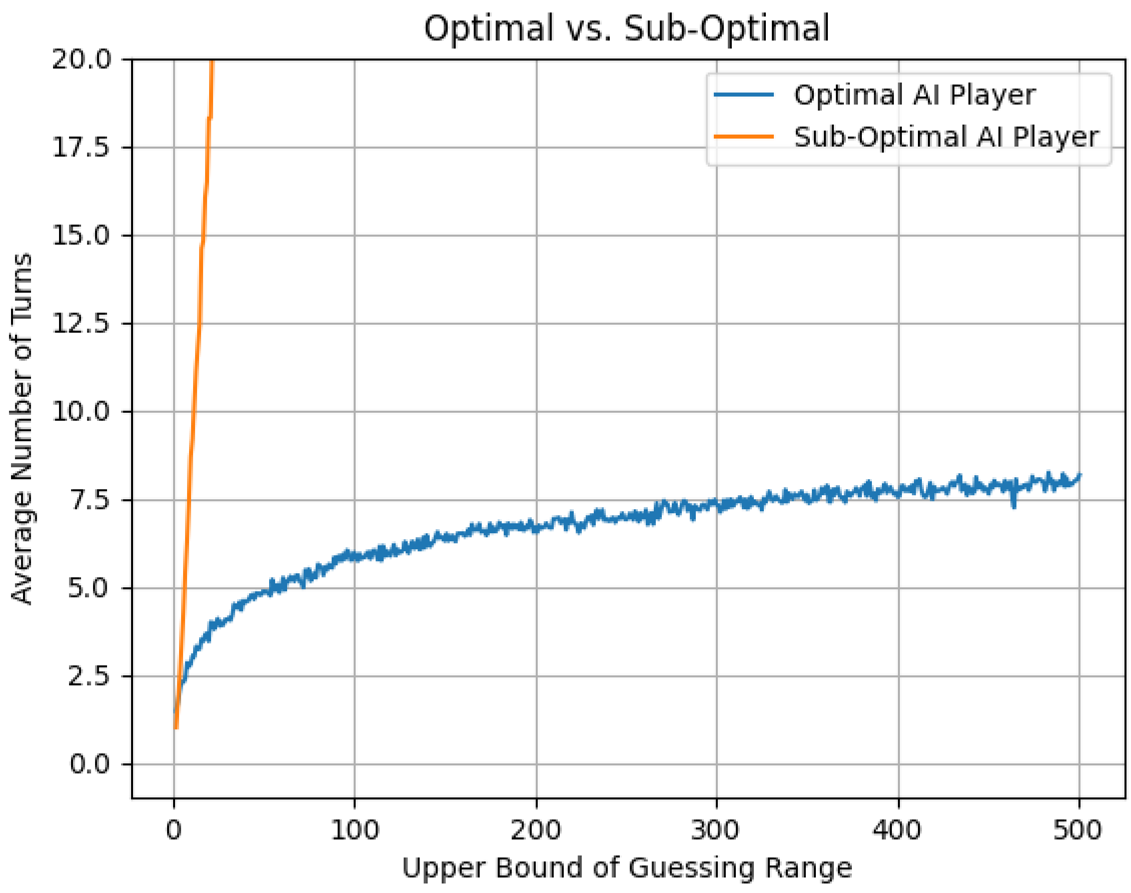
Inputs

The program does not require inputs to run, however inputs could be implemented to allow the user to select the maximum upper boundary of the guessing range and the number of trials for each upper boundary.

Outputs

The program generates a graph displaying the upper bound of the guessing range versus the average number of turns for both the Optimal and Suboptimal AI players.


Optimal vs. Sub-Optimal
Optimal AI Player
Sub-Optimal AI Player
Average Number of Turns
Upper Bound of Guessing Range
0
100
200
300
400
500


Optimal vs. Sub-Optimal
Optimal AI Player
Sub-Optimal AI Player
Average Number of Turns
Upper Bound of Guessing Range
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
100
200
300
400
500

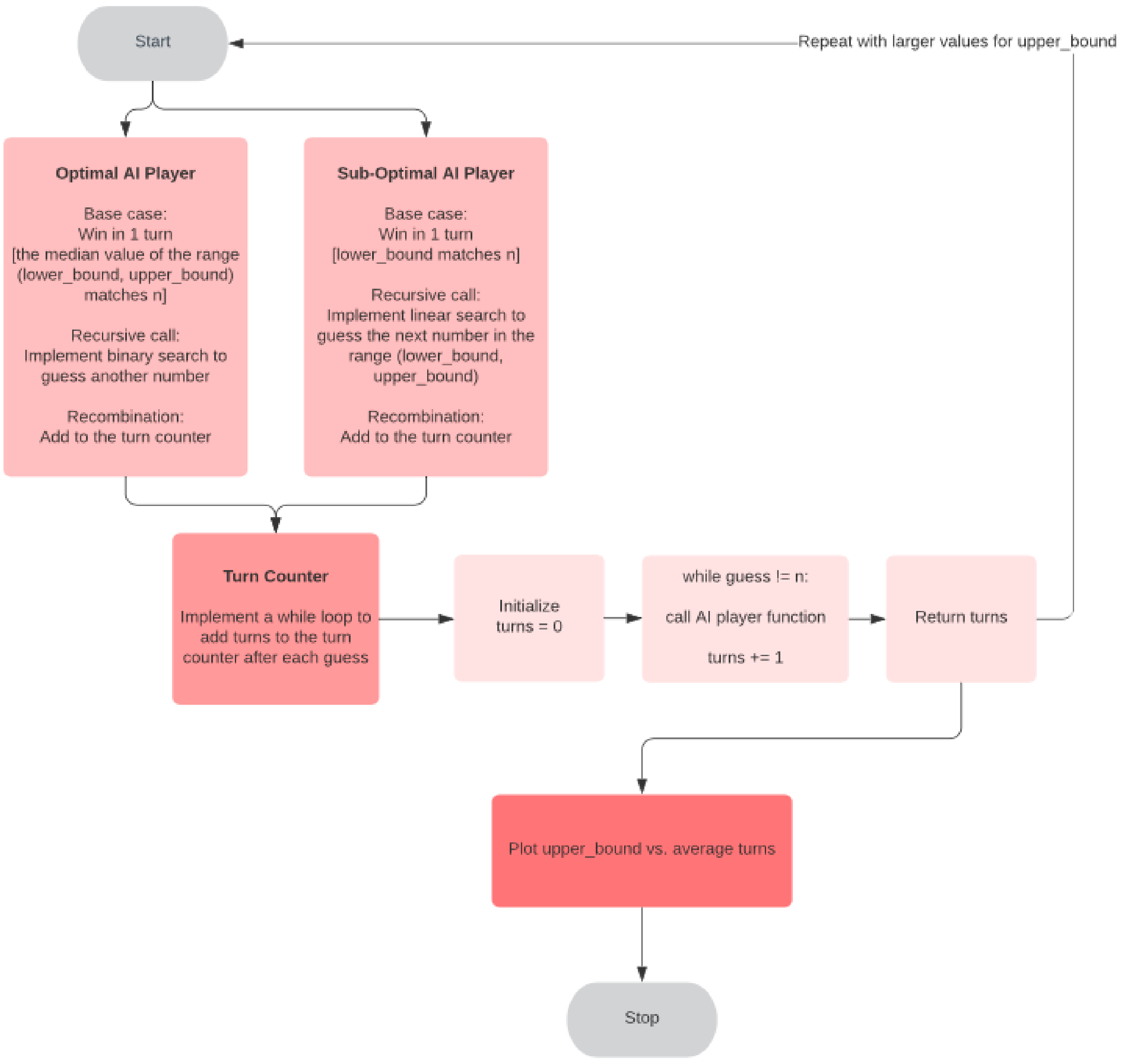
> In the second example, the y-limit is set to 20 to see more detail in the Optimal AI curve as Suboptimal AI's number of turns skyrockets.

Findings

The binary search method implemented by Optimal AI Player outperformed the linear search method of Suboptimal AI Player. The data visualization clearly shows that the average number of turns for Optimal AI player increases logarithmically, while the average number of turns for Suboptimal AI increases linearly as the upper boundary increases.

Concept Map

To prepare for this lab, I created an algorithm flowchart.



Start
Repeat with larger values for upper_bound
Optimal AI Player
Base case: Win in 1 turn [the median value of the range (lower_bound, upper_bound) matches n]
Recursive call: Implement binary search to guess another number
Recombination: Add to the turn counter
Sub-Optimal AI Player
Base case: Win in 1 turn [lower_bound matches n]
Recursive call: Implement linear search to guess the next number in the range (lower_bound, upper_bound)
Recombination: Add to the turn counter
Turn Counter
Implement a while loop to add turns to the turn counter after each guess
Initialize turns = 0
while guess != n: call AI player function turns += 1
Return turns
Plot upper_bound vs. average turns
Stop

The implementation of the turn counter ended up being quite different from what I had planned in my concept map. It turned out to be much simpler than I had anticipated, as it only required that I first initialize turns to 0, then increment turns by 1 within the function, so the turns would automatically increment with each recursive function call.

Collaborators

Nisa Genc and Arthur Waddell reviewed this lab in its early stages. Having to explain how the program worked to my teammates was incredibly helpful for me.

Philip Gray explained how I could count turns by initializing turns to 0 as a function parameter, then increment turns by 1 within the function.

Devon Gardner helped me to debug my code when it was nearly complete. I had run into a problem with the Suboptimal AI player not being able to find its target and returning a value of `None` for number of turns. This caused `calc_average` to malfunction, because it could not calculate an average value when one of the values was not an integer. I'm still not entirely sure why Suboptimal AI couldn't find its target, but I was able to keep the program running by implementing an `else` statement that would return the length of the list of possible target values. (Since it could not find its target, it must have searched through all possible values. So, if the upper bound were 100, for example, Suboptimal AI must have used 100 turns.)

References

GeeksforGeeks. (2020, May 27.) *Python Program for Binary Search (Recursive and Iterative)*. https://www.geeksforgeeks.org/python-program-for-binary-search/

> I referenced this source while implementing binary search for the Optimal AI Player.

GeeksforGeeks. (2018, Dec 3.) *Python Program for Linear Search*. https://www.geeksforgeeks.org/python-program-for-linear-search/

> I referenced this source while implementing linear search for the Suboptimal AI Player.

GeeksforGeeks. (2018, Nov 21.) *Find Average of a List in Python*. https://www.geeksforgeeks.org/find-average-list-python/

> I referenced this source to check the syntax for calculating the average value from the list of turns taken by each player.