MP2 Report

Course: ECE448-LE1

Team: Yuhang Chen

Jiakai Lin

Wenbo Ye

Date: Apr. 1st, 2023

**Section I: CSP**

According to the descriptions of the mp2 file, the task in part I is the problem of pentominoes tiling, which is a common CSP. Our group takes this problem as a search problem and use GDFS algorithm to solve it. We consider to tile each pent onto the board to fix all the points of the board matrix, where the tiling order is column by column (from up to down, left to right). And we use heuristics during the DFS process that help to accelerate the searching by the below methods:

1) choose the next variable (the point) to assign by LRV.

2) use early detection of failure by use forward checking.

Attached with some test codes, the running result with test functions shows above:

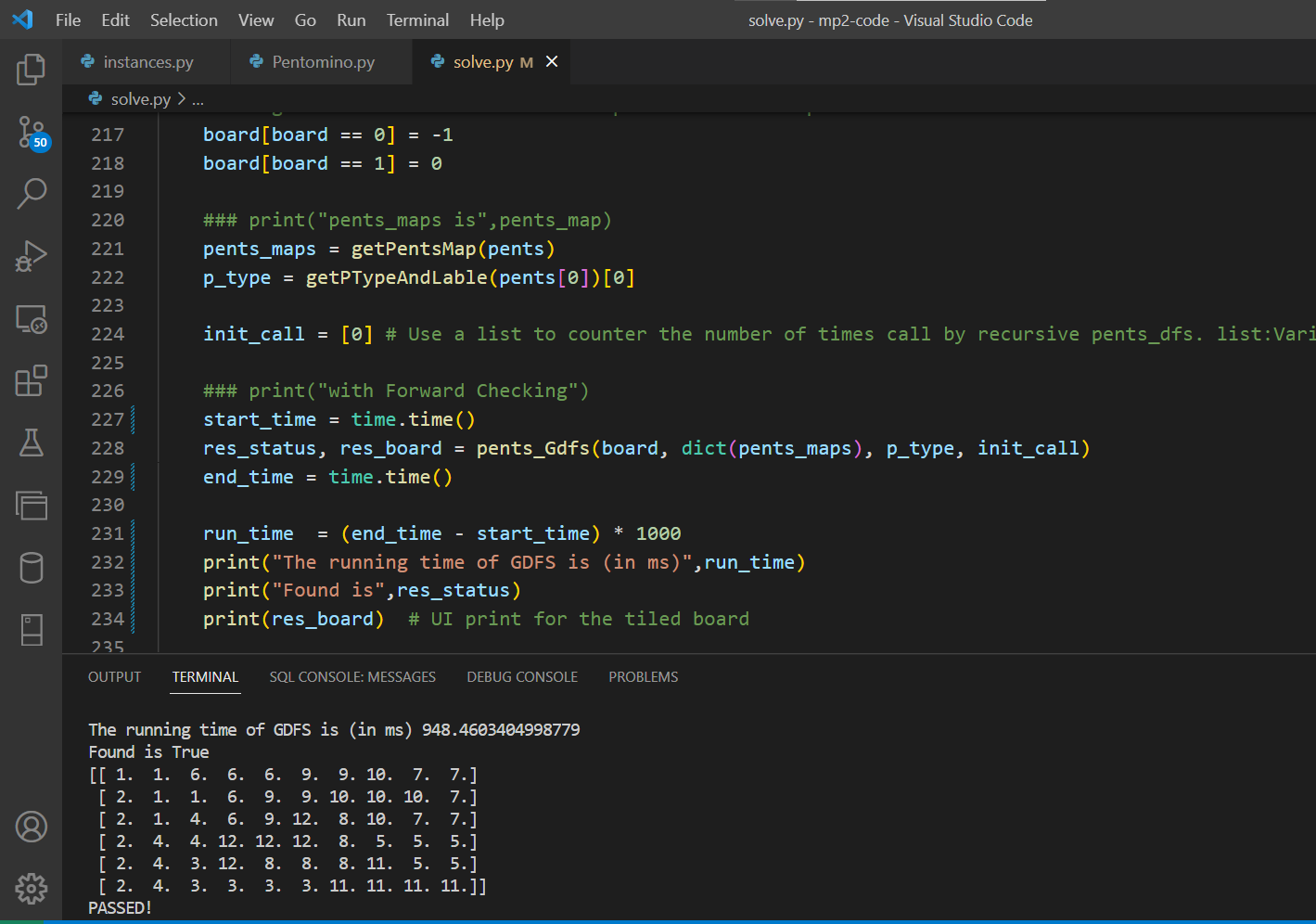


Figure 1. The running result of test on board\_6x10

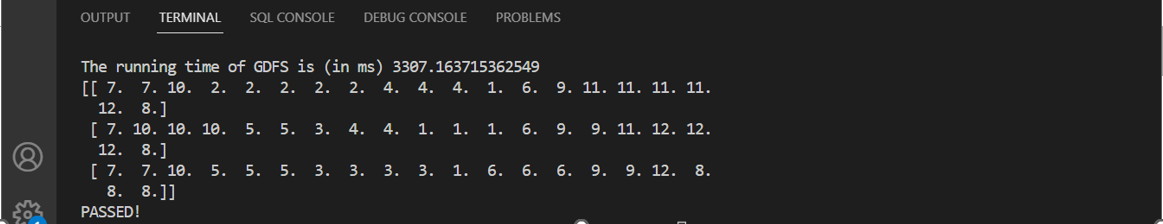


Figure 2. The running result of test on board\_3x20

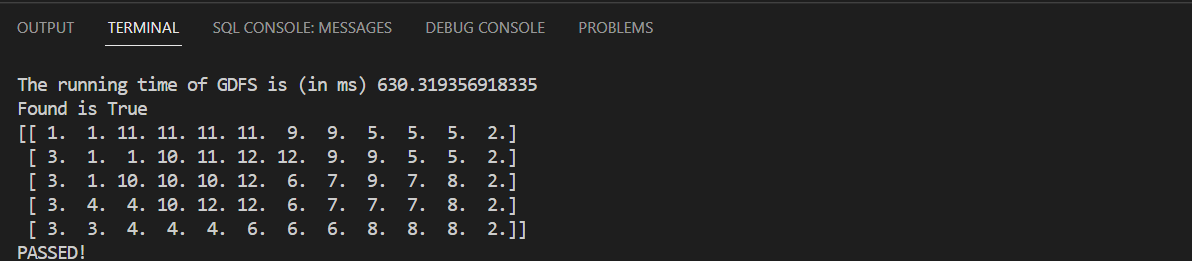


Figure 3. The running result of test on board\_5x12

**Section II:**

Ultimate Tic-Tac-Toe: for each of the four games of predefined agents, report the final

game board positions, number of expanded nodes, and the final winner.

**Section III:**

Ultimate Tic-Tac-Toe: for at least 20 games of offensive agent vs your agent, explain your

formulation and advantages of evaluation function. Report the percentage of winning time

and number of expanded nodes for each game. Report 3-5 representative final game boards

that show the advantage of your evaluation function vs predefined offensive evaluation

function. If your own defined agent fails to beat the predefined agent, explain why that happened.

**Section IV:**

Ultimate Tic-Tac-Toe: for at least 10 games of human vs your agent, discuss your observations,

including the percentage of winning time, the advantages or disadvantages of your defined

evaluation functions. Report 3-5 representative final game boards that show the advantages or

disadvantages of your evaluation function.

**Extra Credit:**

In part I, the data type we use for convenient of the search is dictionary type that can map the pents with its origin position matrix. And in the GDFS loop we use generator type to decrease the loop nested times that largely speed up the program. The second idea is come out to accelerate the GDFS for loop searching and generated from <https://developer.aliyun.com/article/919359>. i.e. For testing board\_3x20, it speed up about 30% from the original (from nearly 4s to 3s in Yuhang’s Computer).

Besides, we do change frequent times of the heuristic that change the searching and back-tracing order of the GDFS. And we finally decide to use the codes submitted version use the forward checking to do the back-tracing earlier while it does not have much effect on such tiny boards in the instance. The effect is unstable due to the actual running efficiency of the local machine. However, we think that it will speed up a lot when the size of the board become increasingly large.

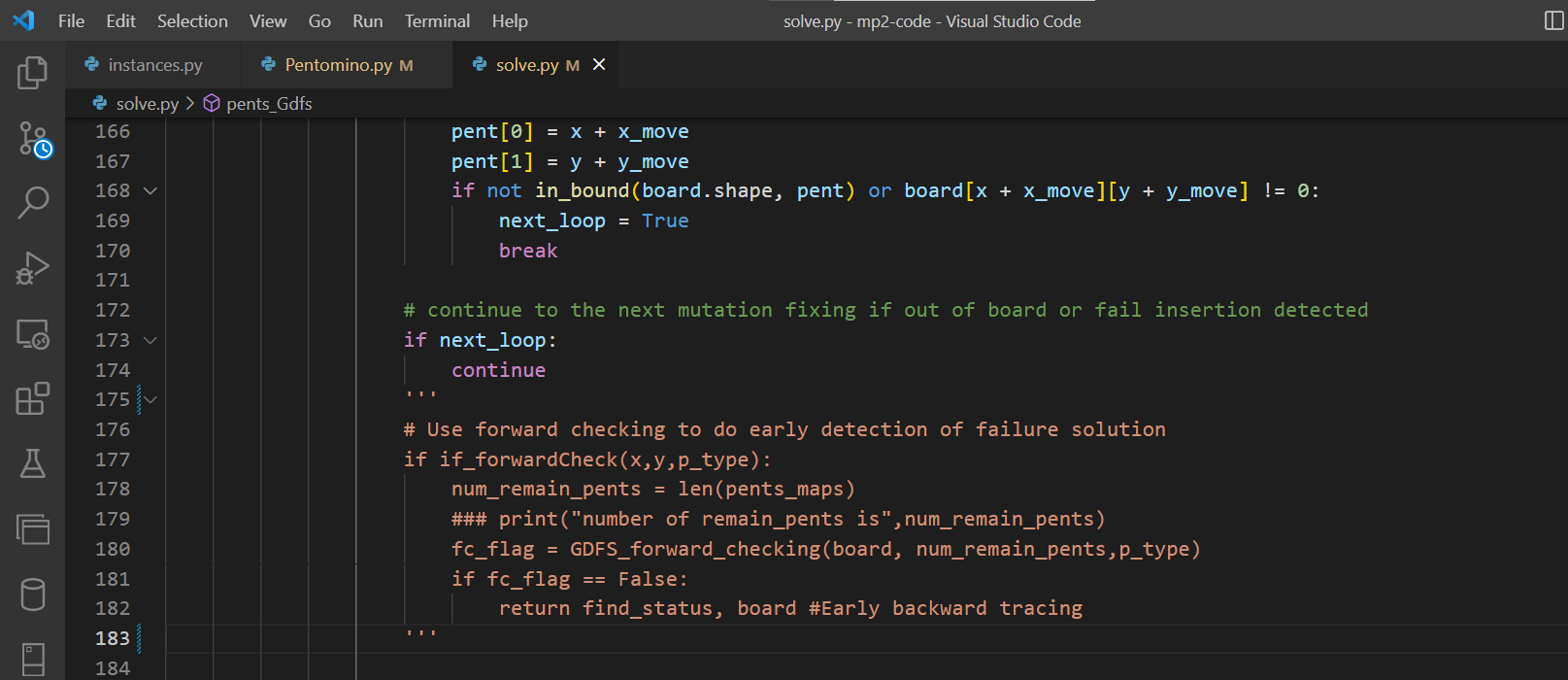


Figure 5. Forward checking Commented out

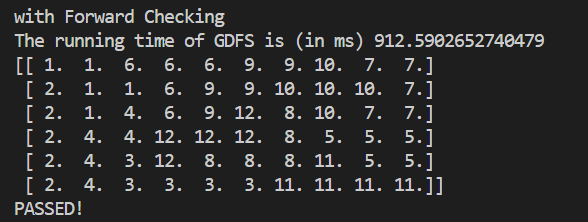


Figure 6. The running result of test on board\_6x10 with forward checking

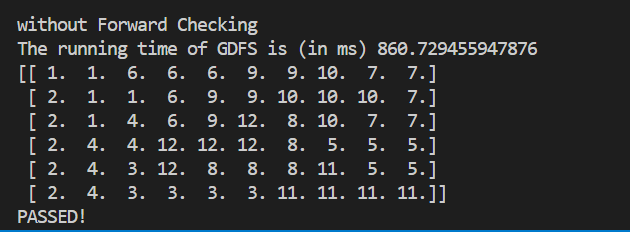


Figure 7. The running result of test on board\_6x10 without forward checking

In Part II,

**Statement of Contribution:**

Yuhang implement the algorithm and do the programing for part 1. Yuhang’ code is debugged by himself and tested well, so his code for part 1 is submitted.

The report of Section I is written by Yuhang.

Wenbo implemented the BFS function.

Jiakai and Yuhang both implemented the greedy function; their results were compared for debugging and Yuhang's code was submitted.

Yuhang implemented the Astar function.

Jiakai implemented the part of extra credit.