

MP2 Report

Course: ECE448-LE1

Team: Yuhang Chen

Jiakai Lin

Wenbo Ye

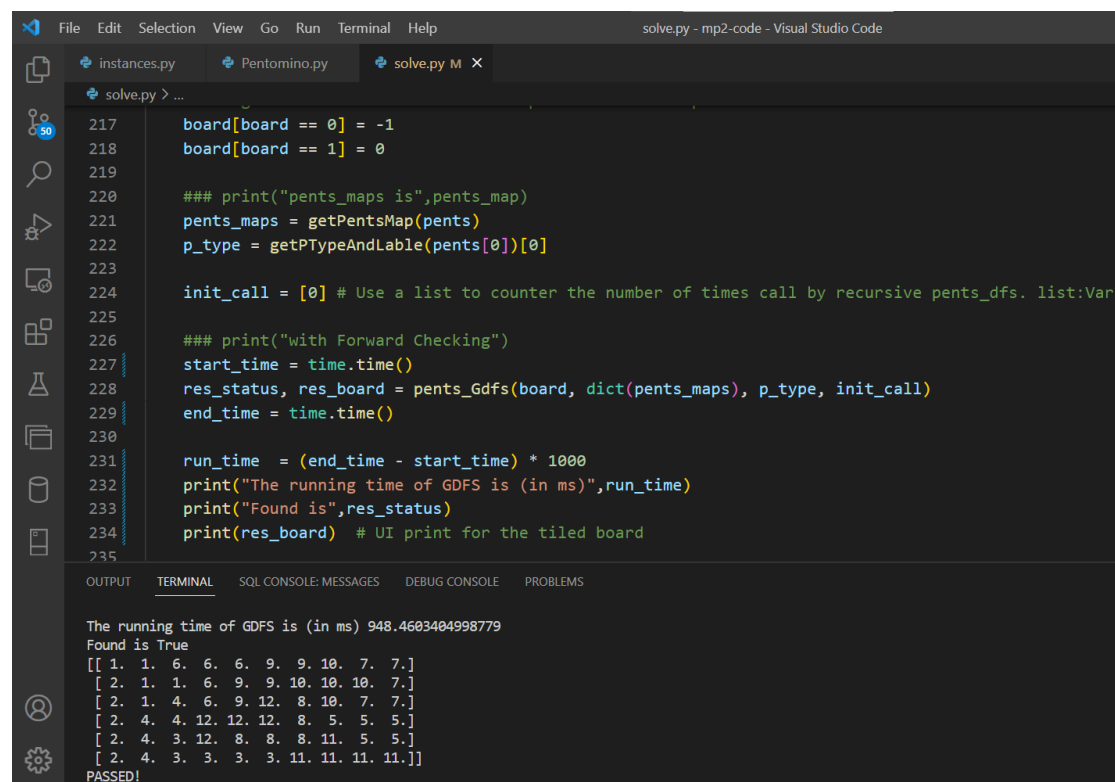
Date: Apr. 1st, 2023

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:

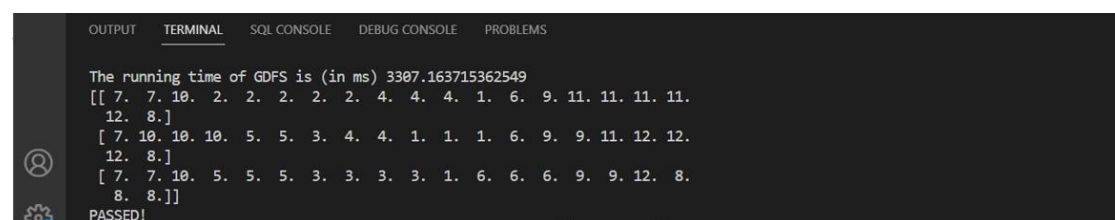


```
217 board[board == 0] = -1
218 board[board == 1] = 0
219
220 ### print("pents_maps is",pents_map)
221 pents_maps = getPentsMap(pents)
222 p_type = getPTypeAndLable(pents[0])[0]
223
224 init_call = [0] # Use a list to counter the number of times call by recursive pents_dfs. list:Var
225
226 ### print("with Forward Checking")
227 start_time = time.time()
228 res_status, res_board = pents_Gdfs(board, dict(pents_maps), p_type, init_call)
229 end_time = time.time()
230
231 run_time = (end_time - start_time) * 1000
232 print("The running time of GDFS is (in ms)",run_time)
233 print("Found is",res_status)
234 print(res_board) # UI print for the tiled board
235
```

OUTPUT TERMINAL SQL CONSOLE: MESSAGES DEBUG CONSOLE PROBLEMS

The running time of GDFS is (in ms) 948.4603404998779
Found is True
[[1. 1. 6. 6. 6. 9. 9. 10. 7. 7.]
[2. 1. 1. 6. 9. 9. 10. 10. 10. 7.]
[2. 1. 4. 6. 9. 12. 8. 10. 7. 7.]
[2. 4. 4. 12. 12. 12. 8. 5. 5. 5.]
[2. 4. 3. 12. 8. 8. 8. 11. 5. 5.]
[2. 4. 3. 3. 3. 3. 11. 11. 11. 11.]]
PASSED!

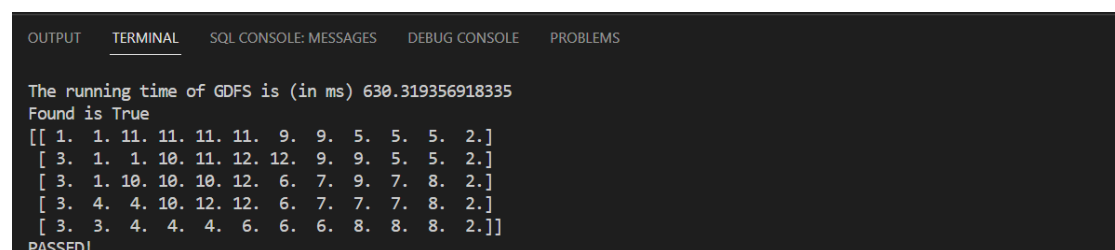
Figure 1. The running result of test on board_6x10



```
OUTPUT TERMINAL SQL CONSOLE: MESSAGES DEBUG CONSOLE PROBLEMS
```

The running time of GDFS is (in ms) 3307.163715362549
[[7. 7. 10. 2. 2. 2. 2. 4. 4. 4. 1. 6. 9. 11. 11. 11. 11.
12. 8.]
[7. 10. 10. 10. 5. 5. 3. 4. 4. 1. 1. 1. 6. 9. 9. 11. 12. 12.
12. 8.]
[7. 7. 10. 5. 5. 5. 3. 3. 3. 3. 1. 6. 6. 6. 9. 9. 12. 8.
8. 8.]]
PASSED!

Figure 2. The running result of test on board_3x20



```
OUTPUT TERMINAL SQL CONSOLE: MESSAGES DEBUG CONSOLE PROBLEMS
```

The running time of GDFS is (in ms) 630.319356918335
Found is True
[[1. 1. 11. 11. 11. 11. 9. 9. 5. 5. 5. 2.]
[3. 1. 1. 10. 11. 12. 12. 9. 9. 5. 5. 2.]
[3. 1. 10. 10. 10. 12. 6. 7. 9. 7. 8. 2.]
[3. 4. 4. 10. 12. 12. 6. 7. 7. 7. 8. 2.]
[3. 3. 4. 4. 4. 6. 6. 6. 8. 8. 8. 2.]]
PASSED!

Figure 3. The running result of test on board_5x12

Section II:

In the first game, offensive player (maxPlayer) goes first, with both agents using minimax algorithm. The final game board position is shown in the following figure.

```
O X _ _ O X _ _ O
O _ O X O _ X X _
X _ X _ O X _ _ O

O X _ X O X X _ _
O X X O _ O _ X X
_ O _ _ _ O _ O

X O _ _ X _ O _ O
_ _ _ _ O X _ _
X O _ _ X O O X X

Average expanded nodes: 372.7826086956522
The winner is minPlayer!!!
```

Figure 4. The running result of Game 1

As shown in the figure, the final winner is minPlayer (defensive) and the number of expanded nodes is 372.

In the second game, offensive player (maxPlayer) also goes first, but with maxPlayer using minimax algorithm and minPlayer using alpha-beta algorithm. The final game board position is shown in the following figure.

```
O X _ _ O X _ _ O
O _ O X O _ X X _
X _ X _ O X _ _ O

O X _ X O X X _ _
O X X O _ O _ X X
_ O _ _ _ O _ O

X O _ _ X _ O _ O
_ _ _ _ O X _ _
X O _ _ X O O X X

Average expanded nodes: 242.67391304347825
The winner is minPlayer!!!
```

Figure 5. The running result of Game 2

As shown in the figure, the final winner is also minPlayer (defensive) and the number of expanded nodes is 242.

In the third game, defensive player (minPlayer) goes first, with maxPlayer using alpha-beta algorithm and minPlayer using minimax algorithm. The final game board position is shown in the following figure.

```

X O X O X O O _ X
X O X O X _ O O O
O _ _ _ O X _ _ X

X O _ O _ O O X X
X O O X _ X _ _ O
_ X O _ _ _ X _ O

_ X _ _ O X X X _
_ _ _ X _ _ O _ X
O X _ _ O X _ O O

Average expanded nodes: 220.47058823529412
The winner is minPlayer!!!

```

Figure 6. The running result of Game 3

As shown in the figure, the final winner is also minPlayer (defensive) and the number of expanded nodes is 220.

In the fourth game, defensive player (minPlayer) goes first, with both agents using alpha-beta algorithm. The final game board position is shown in the following figure.

```

X O X O X O O _ X
X O X O X _ O O O
O _ _ _ O X _ _ X

X O _ O _ O O X X
X O O X _ X _ _ O
_ X O _ _ _ X _ O

_ X _ _ O X X X _
_ _ _ X _ _ O _ X
O X _ _ O X _ O O

Average expanded nodes: 96.15686274509804
The winner is minPlayer!!!

```

Figure 7. The running result of Game 4

As shown in the figure, the final winner is also minPlayer (defensive) and the number of expanded nodes is 96.

Overall, all four tests showed that minPlayer won. If the alphabeta algorithm is used, the number of expanded nodes will significantly decrease and run faster, but it will not affect the game results.

Section III:

```
#-----Test for YourAgent-----
### uttt=ultimateTicTacToe()
total = 30
Dwin = 0 # the number of times that the designed agent wins
for i in range(total):
    uttt=ultimateTicTacToe()
    gameBoards, bestMove, winner=uttt.playGameYourAgent()
    if winner == -1:
        Dwin += 1
print("The number of times Your Agent win is",Dwin)
print("The rate that your agent win is",Dwin/total)
```

管理员: Anaconda Powershell Prompt (Anaconda)

```
(base) PS C:\Users\Wendell\desktop\ECE448> python .\uttt.py
The number of times Your Agent win is 26
The rate that your agent win is 0.8666666666666667
(base) PS C:\Users\Wendell\desktop\ECE448> _
```

Figure 8. The testing result of 30 games by our agent and the predefined agent

First, we make the agent more aggressive than minPlayer, mainly on the relative sizes of twoInARow, preventThreeInARow and corner parameters.

Secondly, we try to get our agents to take the initiative and try to get a good position at the beginning of the board. If our agent is first, it will try to choose the most central cell to ensure more possibilities. If it is the backhand, it tries to select the position of the small board relative to the whole board (such as the (0,8) position of the board numbered 2 and the (4,0) position of the board numbered 3), thus forcing the opponent to continue the next move on your board which has a certain advantage.

For example, in the following example, maxPlayer selects the upper left corner of the board with id=2, and our agent immediately occupies the upper left corner of the board with id=0, so maxPlayer must complete the next step in the board with id=0, which saves some initiative and combines with high aggression. It can be noticed that maxPlayer on many local boards is only one step away from winning the final game, which is because our agent pays more attention to attack rather than defense.

```

Off: 1
start board: 2
O X _ O _ _ X _ _
X X O _ X O _ O X
_ _ _ X _ O X O _

O _ X _ O O X O O
O _ _ X _ _ X _ X
O _ X X _ _ _ X _

_ X _ _ _ _ X X
_ _ O _ _ O O _
O X O O X O _ _ X

```

Figure 9. Our agent(O) win the game with start board 2

Another example is that our agent started the game on the board with id=0 first. Due to choosing the center position, the opponent spent 5 chances to block our pieces on this board, while we only spent 4 times. However, we were the first one, so it is obvious that we occupied the absolute initiative on other numbered boards. Finally the other side could not stop, we successfully won on the id=2 board.

```

Off: 0
start board: 0
O X X O X _ O O O
X O O O O X _ X X
O X X X X _ _ _ _

O O X X X O X _ _
_ X _ X O _ _ _ _
_ _ _ _ _ _ O O _

X O _ O _ X _ O O
_ _ _ O _ _ _ _ _
X O _ _ _ X _ _ _

```

Figure 10. Our agent(O) win the game with start board 2

Of course, this aggressive strategy can sometimes go wrong. For example, in the following game, maxPlayer starts the game first on the board id=8. Spending too many times blocking maxPlayer's attack on the board with id=0 resulted in maxPlayer's number on the board with id=2 far exceeding the number of our pieces, and finally failed to complete the blocking and lost the game

```

Off: 1
start board: 8
O X O O _ _ X X X
X X O _ X X _ _ _
_ O _ O _ _ O X _

O O _ _ O O X _ O
_ _ _ X _ _ _ _ _
_ _ _ _ _ _ O X _

X _ _ X _ O X _ _
_ O X _ _ O _ _ _
X _ O _ _ _ X _ _

```

Figure 11. MaxPlayer(X) win the game with start board 8

Section IV:

During the game by human and the agent, we found in actual test that the agent has the higher rate (more than 70%) to beat the human. The reason for this phenomenon is the designed evaluation is taking three basic rules + additional designed rules to maximize the utility score (i.e the agent is in offensive method) and use alpha-beta algorithm to consider 3 level of depths in every step, the winning position is accumulated step by step by the agent while human is relatively weak in this by the limitation of our brain calculation capacity.

```

X _ X O _ _ O _ _
O X X X O X O X _
O _ O _ _ _ _ O X

X _ _ _ O X X X O
O O X O X X O _ _
X X O _ X O X O O

_ O O O X _ O _ X
X _ _ _ O O X O O
O X X X O X O X X

Please put in Board # 3
Please type in the row number(0-8): 

```

Figure 12. The human being defeated no matter where to put the X in Board 3

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,

First of all, Jiakai put forth an idea and Wenbo modified the original evaluation function rules so that the addition of fractions is no longer linear. For example, the score added by the first 'two in row' will be different from the score added by the second 'two in row'.

Secondly, Yuhang introduced the fourth rule to our agent that occupying the center point of each local board can earn some bonus points.

Third, Wenbo introduced an extra rule. If there is also at least one 'two in row' in the local board corresponding to current 'two in row', the maxPlayer can earn some bonus points. In other words, the opponent cannot block your 'two in row', otherwise you will win in another local board.

Statement of Contribution:

Yuhang mainly 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 mainly implemented the evaluatePredifined function, including three predefined rules. These algorithms have been understood by all team members. And he was also responsible for the checkMovesLeft function, the checkWinner function, the alphabeta function and the minimax function. Firstly, he debugs himself, and then everyone debugs together.

Jiakai mainly implemented the playGamePredifinedAgent function, the playGameYourAgent function, and the playGameHuman function. The test function was mainly written by Jiakai and Yuhang, and all team members participated in testing our agent.

The extra credit part was implemented by all three of us.