ENDGAME(Team-8)
Nonidh Singh(20171203)
Sarthak Singhal(20171091)

AI(Tic Tac Toe Bot)

OVERVIEW

So basically in this assignment we have to implement an AI bot which would compete in a tic tac toe competition with other bots.

PROBLEM STATEMENT

- The Extreme Tic Tac Toe is an extension of 3x3 Ultimate Tic Tac Toe which in turn is an extension of standard 3x3 Tic Tac Toe. Extreme Tic Tac Toe comprises of 2 boards of 3x3 board in which each cell further is a 3x3 board.
- The game is between two teams.
- Coin is flipped to choose who will move first.
- Player 1 marks ‘x’ and Player 2 marks ‘o’ on the board.
- The player who makes a legitimate pattern wins the whole board.
- Refer to the PDF for more details.

SPECIFICATIONS

HEURISTIC

Our heuristic is as follows:

- We start by checking the status of game:
    - If we have won then we return inf.
    - If opponent has won then we return -inf.
    - If there is a draw we calculate the utility from a separate function based on the points of each block which helps us to maximize the score.
    - If the time limit (less than official time) exceeds, then we return utility of the current board without exploring any other nodes.
    - Else if we have time, we keep on exploring the nodes to look ahead.

- In our main utility function we check all possible winning combinations:
    - Rows
    - Columns
    - Diagonals

- For each such combination (say row) we check all possible cases (row1, row2, row3), our win chance and how many symbols we have in each combination (1,2 or 3).
    - We assign different weights to each case:
        - Won 3/3:- inf
        - Won 2/3:- 134
        - Won 1/3:- 17
    - We subtract the utility of opponent player (same but -ve) from our player and return the calculated utility.

- Now to handle initial stages (when small board remains empty->”-”) we calculate utility from the status of big board through a probabilistic approach.
    - We calculate the probability of our bot to win any cell in small board.
    - If we have a high chance of winning, then we add utility of corresponding block in big board to our main utility value.

- In the big board to calculate the utility of a block, we do the same thing for each block as done for a cell in small board but with different weights.
    - Weights for big board cell are smaller than the ones in the small board as they have less effect in winning the game as compared to small boards.
    - Weights for big board block are:
        - Won 3/3:- 17
        - Won 2/3:- 4
        - Won 1/3:- 1.2

- Finally we sum the utility of both the boards and return it.
- All the weights are decided by testing the bot’s performance by playing matches with ourselves.
- We can adjust the weights suitably to make it more attacking or defending.

SEARCH STRATEGY

- Minimax Algorithm:
  - Minimax is a kind of backtracking algorithm that is used in decision making and game theory to find the optimal move for a player, assuming that your opponent also plays optimally. It is widely used in two player turn-based games such as Tic-Tac-Toe, Backgammon, Mancala, Chess, etc.
  - In Minimax the two players are called maximizer and minimizer. The maximizer tries to get the highest score possible while the minimizer tries to do the opposite and get the lowest score possible.
  - Every board state has a value associated with it. In a given state if the maximizer has upper hand then, the score of the board will tend to be some positive value. If the minimizer has the upper hand in that board state then it will tend to be some negative value. The values of the board are calculated by some heuristics which are unique for every type of game.
- Alpha-beta pruning
  - Alpha-Beta pruning is not actually a new algorithm, rather an optimization technique for minimax algorithm. It reduces the computation time by a huge factor. This allows us to search much faster and even go into deeper levels in the game tree.
  - It cuts off branches in the game tree which need not be searched because there already exists a better move available. It is called Alpha-Beta pruning because it passes 2 extra parameters in the minimax function, namely alpha and beta.
- Idfs
  - Iterative deepening search (or iterative deepening depth-first search) is a general strategy, often used in combination with depth-first tree search, that finds the best depth limit. It does this by gradually increasing the limit—first 0, then 1, then 2, and so on—until a goal is found. This will occur when the depth limit reaches d, the depth of the shallowest goal node.
  - Iterative deepening combines the benefits of depth-first and breadth-first search. Like depth-first search, its memory requirements are modest: O(bd) to be precise. Like breadth-first search, it is complete when the branching factor is finite and optimal when the path cost is a nondecreasing function of the depth of the node.
  - We have used a variation of IDFS.
- Cutting off search

    - Since we cannot traverse to the end of the search tree, we will cut off the search in between and return best move found so far. This is done by checking against time and when time exceeds certain limit, we return the best move found so far.
- Move ordering
    - In this we order the move according to our heuristic function so best move according to heuristic is picked up first.This ordering can increase efficiency of alpha beta pruning.

PERFORMANCE ANALYSIS

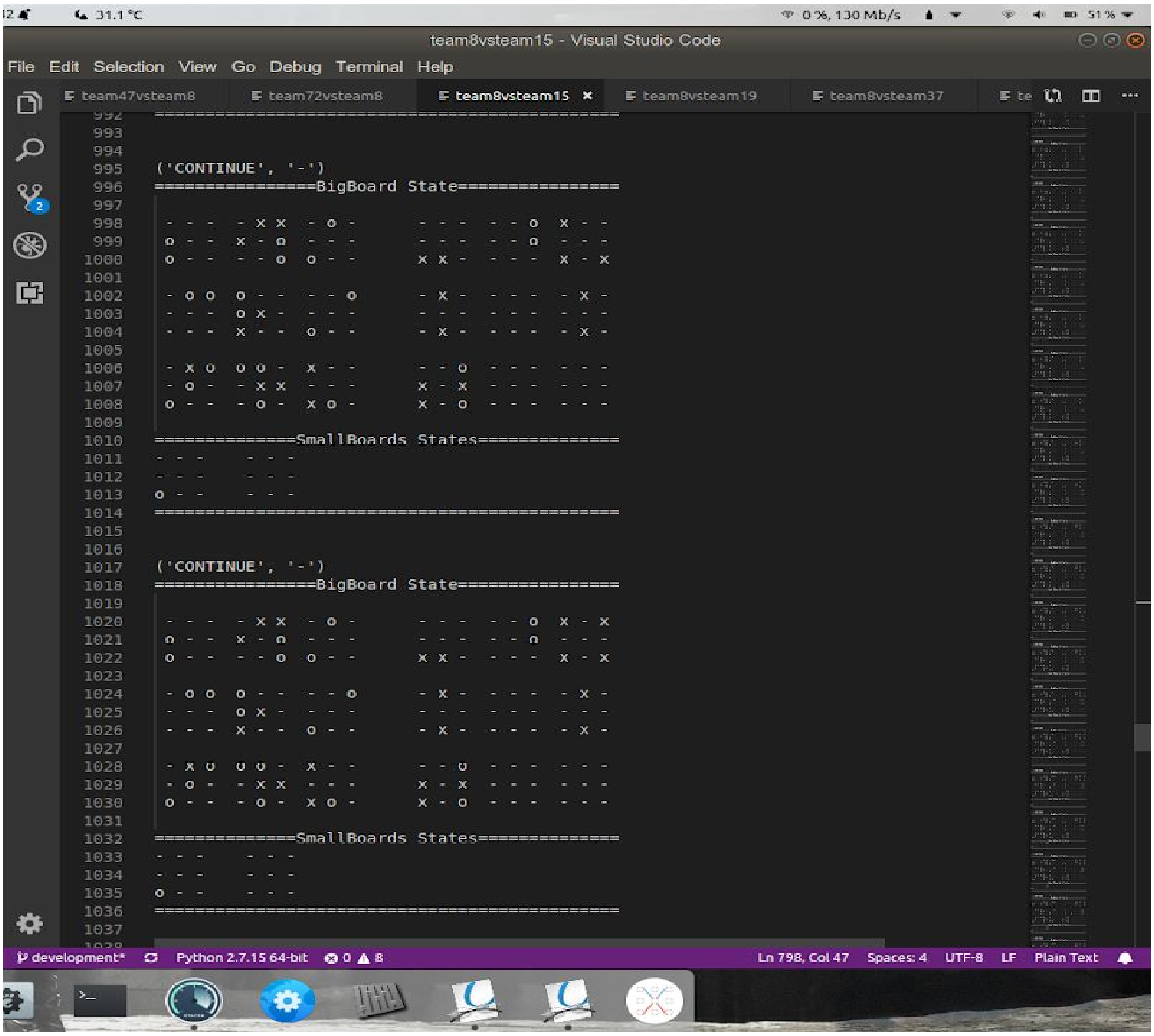
Some visible comments on various teams from the logs are as follows:-

- TEAM-8:
    - Giving random moves
- TEAM-47:
    - Giving random moves
- TEAM-38:
    - Plays in one board only.
    - When one cell about to draw, purposely gives moves in that cell which is of no use.

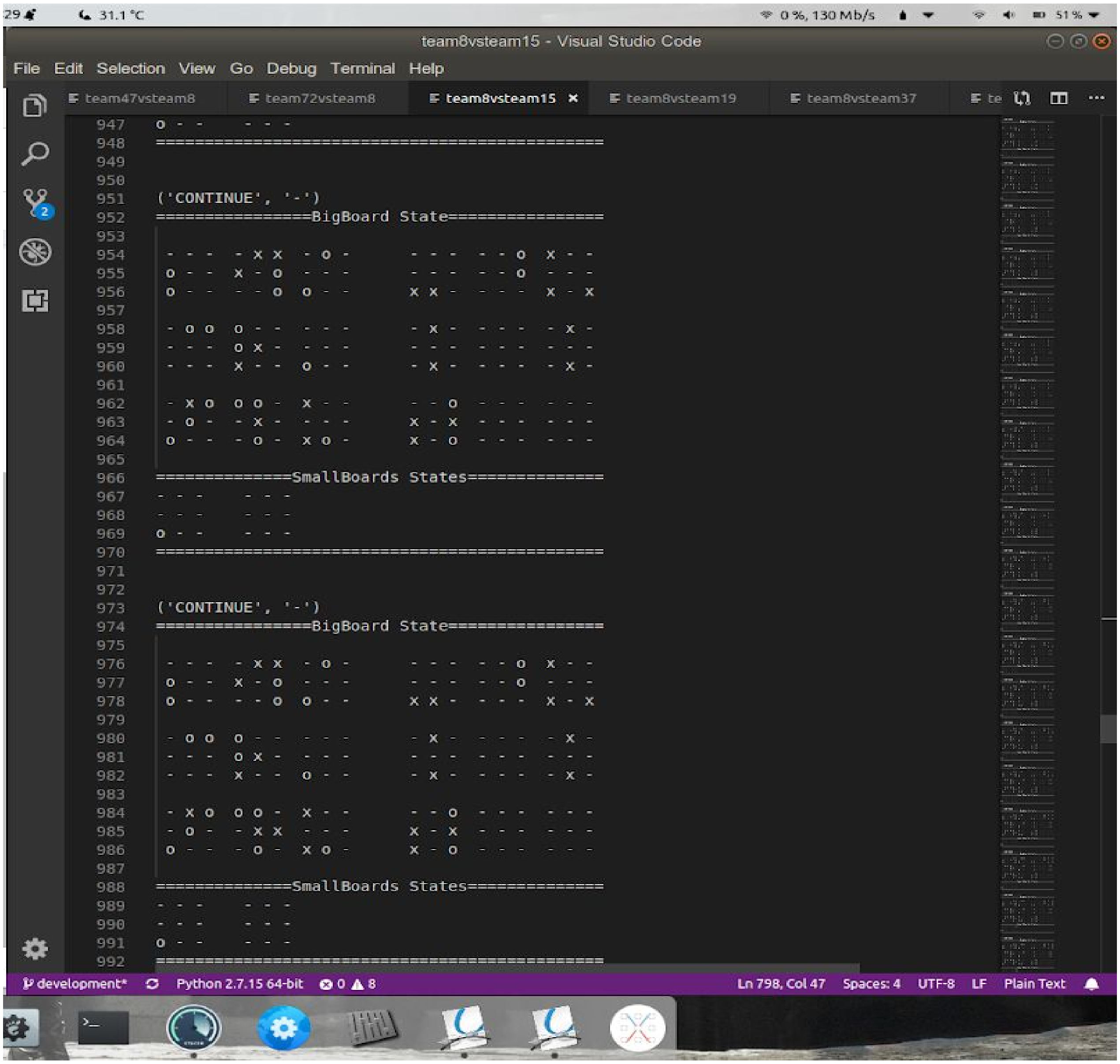
MATCH-REPORT

Similar instances of moves made by the bot are reported only once.

TEAM-8 VS TEAM-15

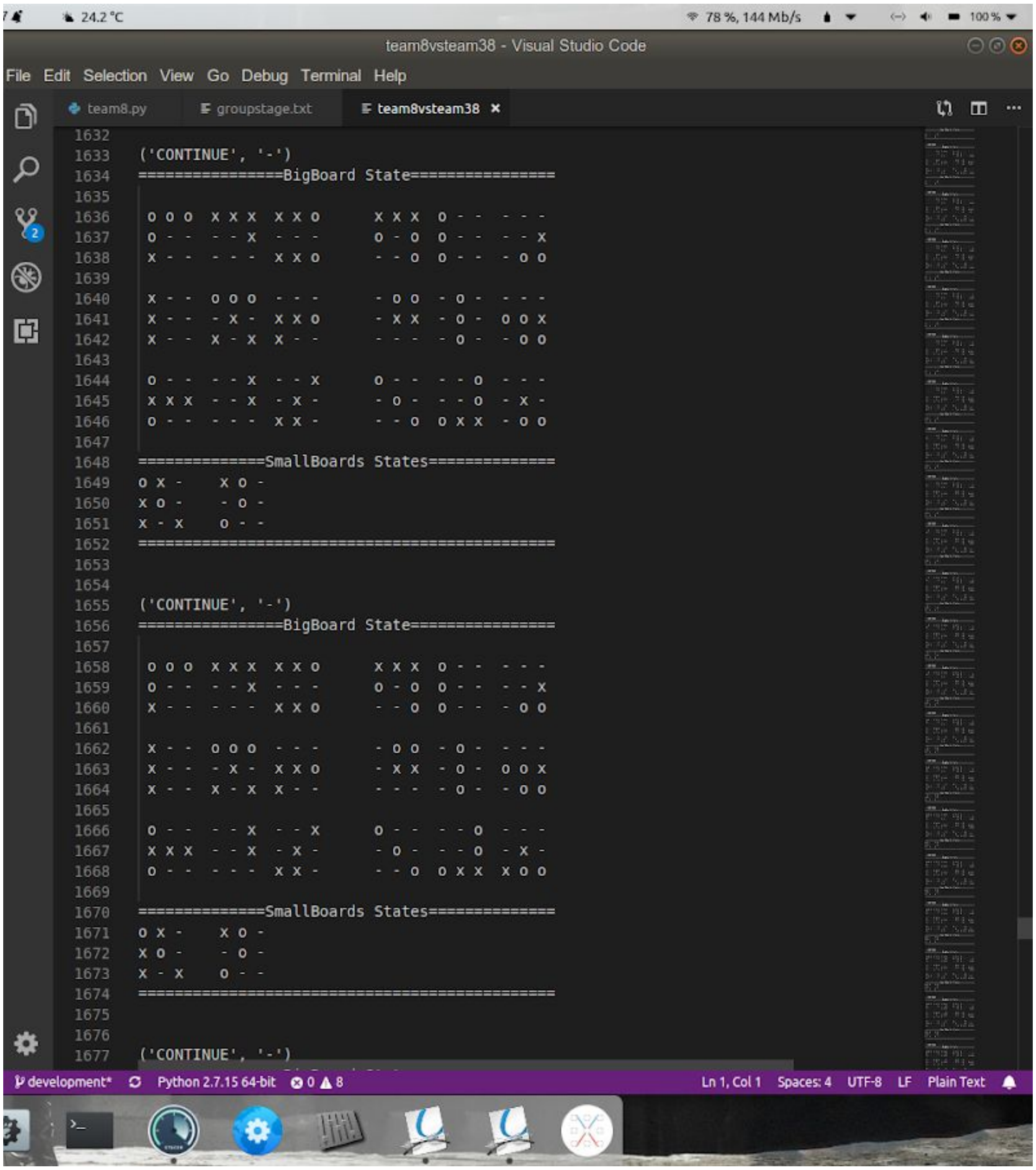


- What human brain should do
    - Clearly a human would win the block
- Required changes in the heuristic
    - We can increase the weight of winning a cell in big board

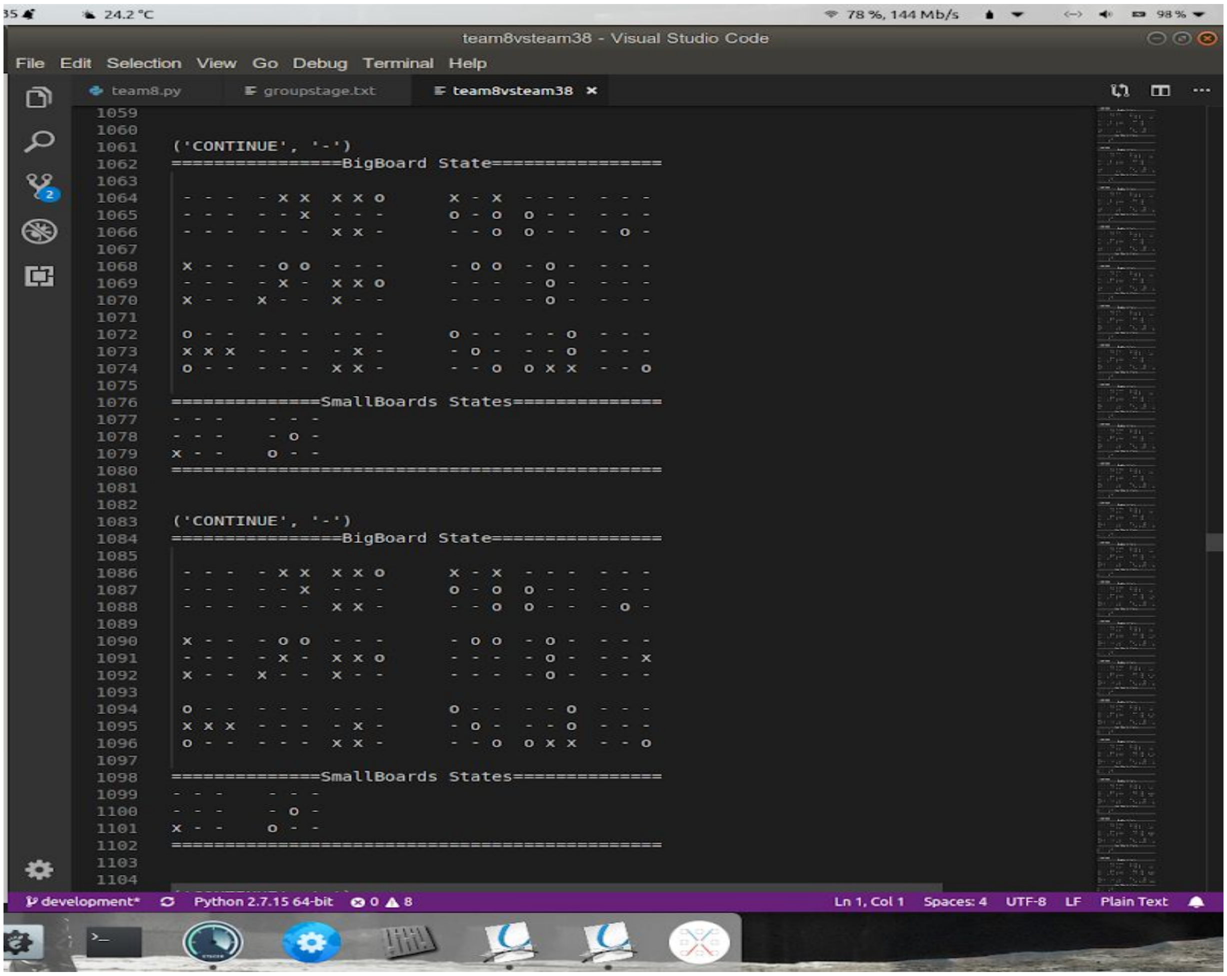


- What human brain should do
  - A human would block the opponent
- Required changes in the heuristic
  - We can increase the -ve weight when an opponent wins a block in big board

TEAM-8 VS TEAM-38

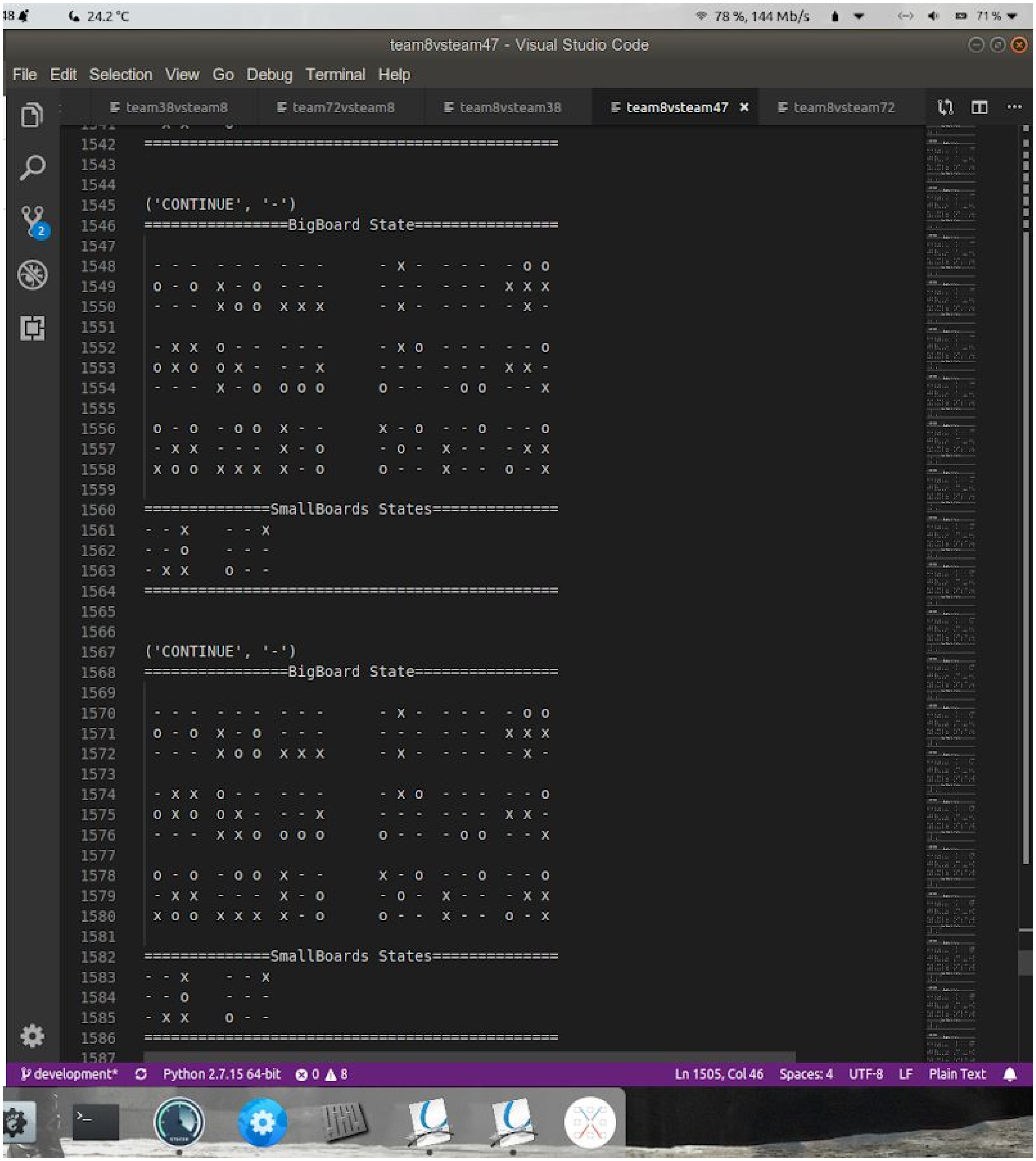


- What human brain should do
    - One would not give random move to opponent at this stage when the game is about to end
- Required changes in the heuristic
    - Incorporate cases when the opponent gets random moves



- What human brain should do
  - One would have made the block
- Required changes in the heuristic
  - Maybe increase the weight of winning a cell in big board

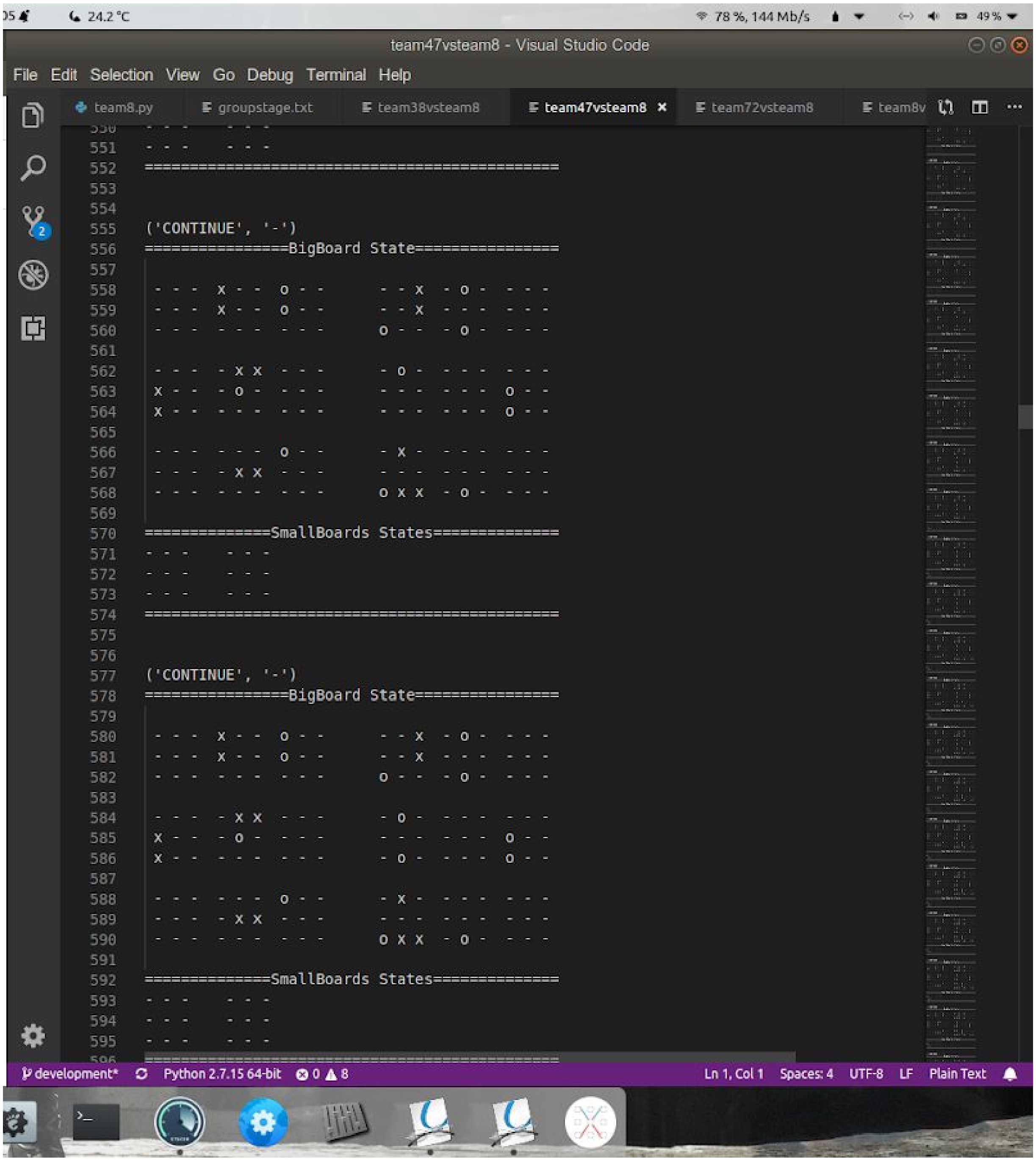
TEAM-8 VS TEAM-47



- What human brain should do
    - One would have made the centre block and then get a bonus move with which we could have won the game
- Required changes in the heuristic

◦ Maybe increase the weight of winning a cell in big board and give even more weight to cases where we get random moves after winning a cell in big board

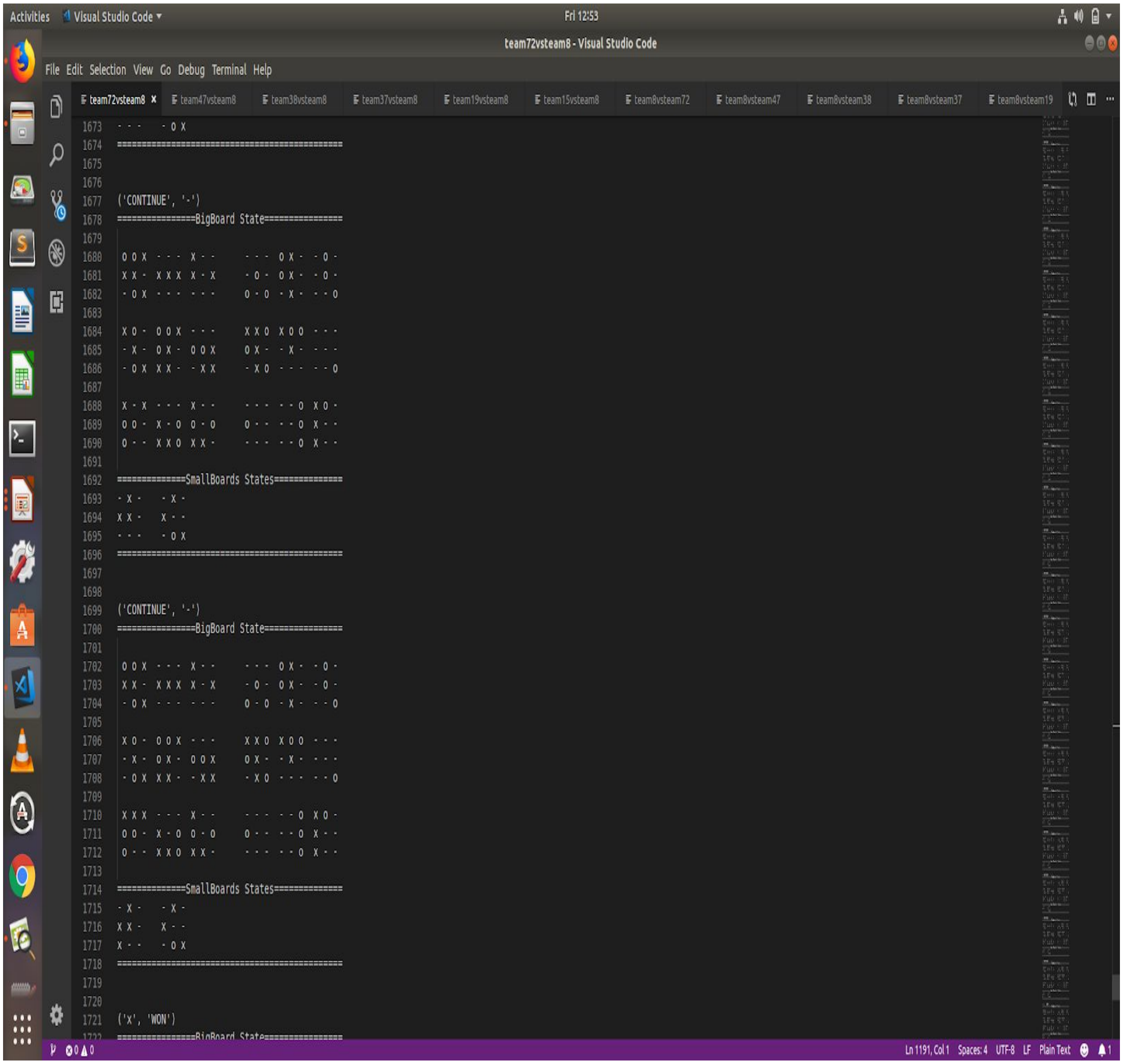
TEAM-47 VS TEAM-8



- What human brain should do
  - One would have moved in bottom right cell to prevent opponent from winning the block
- Required changes in the heuristic

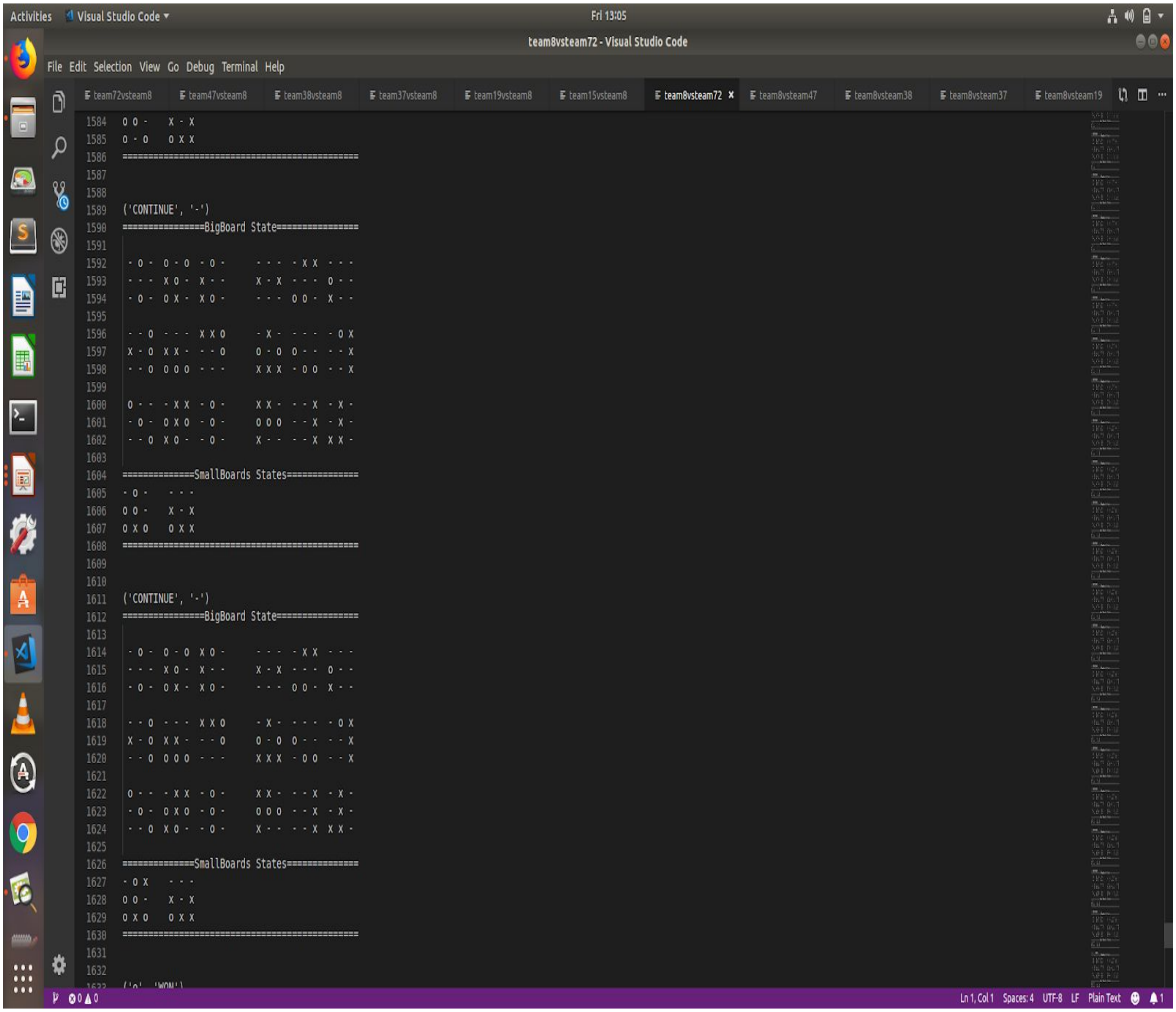
- Incorporate move which would direct the opponent to blocks which would cause less damage

TEAM-72 VS TEAM-8



- Quite poor performance:Here during when the opponent is moves away winning the bot should not give random moves.
- The heuristic should have tweaked in accordance with probability of random move

TEAM-8 VS TEAM-72



- Here the bot should have played at(1,3) and avoided the immediate loss.So the play should have been defensive here rather than forming own board and giving the game away.
- The heuristic should have tweaked in accordance with probability of random move

FURTHER ANALYSIS

STRENGTHS

➔ We have selected best moves according to heuristic and then expanded in that order(beam search).

➔ Bonus move handling.

➔ Saving time in initial stages of game by making bot think like human.

- Checked condition on draw to maximize the score in case there is a draw.
- Our utility function also takes care of the cases where we have high probability of winning.

DIFFERENCE IN STRATEGY

- Some bots may have added learning element(MCTS).
- They may have given different weights to utilities which would imply different strategies like blocking, attacking etc.
- They may have used probabilistic move picking.
- Some bots may have given less priority to cases when both blocks are won by someone and then the other player gets to move randomly in any part of the board.

EXPECTED RANK

- Expected rank also depends on the pool we get and various teams in that.
- Since we don’t have logs for the matches of our bot and other qualifiers from other pools we cannot state much about it.
- Based on our performance in the pool, we had close matches when we moved first and may beat them(top 2 teams) if another match is played.
- Commenting on the rank of our bot, we think that it is among Top-6 bots. This is because we had tested our bot with pyc files of other teams and it performed well.