**ASSIGNMENT 2 – REPORT**

**Part 1**

**Steps to run -**

**How the problem was formulated –**

**How the algorithm works –**

**Problems faced –**

**Part 2**

**Steps to run -**  Same as original program.

Worked on the simple version of the program, so to run use: **python tetris.py computer simple**

**How the problem was formulated –** The problem was formulated as a Game of Chance as shown in the figure below:

**Max Layer (for any particular piece for Eg. ‘xxxx’)**

*Actions – “mmn”* Actions – “bbn”

**Chance Layer**

Other Chance nodes corresponding to different actions

**How the algorithm works –**

The algorithm uses expecti-minimax with the following heuristics and expectation values –

1. The following heuristics were used to rank actions in order to predict the next best action –
2. **Row Heuristic: Space-Chunks** – Space chunks were calculated from bottom up until a row is found which has a space chunk with spaces equal to or greater than the lowest part of the piece being considered. The heuristic is calculated as –

**constant\_multiplier \* (chunk\_tuple[1][index] - non\_space\_chars)** where chunk\_tuple[1][index] is the space chunk’s size for a particular row, the non\_space\_chars are the non-space-chars in the lower part of the piece which will make contact with the board, and the constant\_multiplier is -1 as lesser the difference between the space chunk size and non\_space\_chars of the successor\_piece, the higher it is likely to be favorable. So, fundamentally the comparison would be between a lower negative value and a higher negative value.

Once found, a second heuristic was used on that row.

1. **Column Heuristic: Obstructing Columns –** Once a space-chunk in a row is located, column of the successor piece (piece after applying intermediate actions) is checked to see if there is another piece obstructing the column by examining rows top to bottom. If there is indeed another piece’s part obstructing the space chunk in the row in which the space-chunk was found in (a) above, a constant is subtracted from the heuristic, making it more negative and hence lower in value and also consequently marking the resulting position of the action as less favorable.
2. **Expectation values -** Pieces from various program runs were written to a file. They were then read using the file - **read\_file.py** to calculate probability of their occurrence. The probability values were then multiplied with the heuristic for each piece based on which piece is being evaluated giving it’s expectation value. (The negative multiplier was not removed, because then more negative values would have become higher positive values and the heuristics would have worked in an inverse manner)

**Problems faced –**

1. The heuristics used above are only a part of the many aspects needed to find the optimal action to choose next. Therefore, they were not effective in every scenario. They resulted in a better gameplay but were not effective in increasing the score.