AI Project 2
13D100032, 130070007, 130070009
Objective

To make a Bot for playing the very popular game 2048. It can be played [here](#).

In this project, we have

- Made a GUI to easily visualise the bot’s gameplay
- Implement and compare various algorithms such as
  - Minimax algorithm
  - Minimax with alpha-beta pruning
  - Expectimax algorithm
  - Monte-carlo algorithm
Algorithms Being Used

Minimax (Alpha - Beta) Search

This involves modelling the game as a state and then using an evaluation function to implement a depth limited min-max search and then subsequently adding pruning to it.

Expectimax Search

This is a slight modification of the standard Min-Max search which is essential to capture the inherent randomness in the game which the above algorithm misses.
Algorithms Being Used

Monte - Carlo Tree Search

This is a very simplistic algorithm that often gives good results. The idea of the algorithm is to play a series of random games from each of the possible next states and select the one giving maximum average score.

This does not need any heuristic function. In our case due to computational constraints the performance wasn’t so good. But it is an interesting method due to the very little programmer effort required.
Gameplay

Few screenshots of the game in action
Results

Following is an analysis of the final scores reached by each of the algorithms.
Contribution of Individual members

The contribution of all 3 team members was roughly equal.

We did everything till the first project evaluation together. This involved creating the game board, implementing minimax and alpha-beta algorithms, etc..

Roughly, the work was split as -

- Anchit - Implementing algorithms, Optimizing transposition tables
- Anand - Optimizing algorithms, Adding evaluation function for minimax
- Charmi - Linking with UI, Heuristic Caching
Key takeaways from the project

- Importance and Significance of code profiling -
  - We used profiling to figure out which parts of the code were taking up the maximum time during execution and first improved those parts.
  - This led to significant improvements in performance (the depth of the search increased by almost 2 levels!)

- Relevance of a good heuristic function -
  - We tried a range of several different heuristics (from a very simple “board score” to a complicated heuristic to give more priority to a “snake-like pattern”.
  - We realised how slight modifications to the evaluation function significantly changed the final result that we obtain.
  - Finally, we stuck to a combination of smaller heuristics and assigning appropriate weights to each of them
Key takeaways from the project

● Speeding up code using various techniques -
  ○ Using minimum space to store game state (powers instead of numbers)
  ○ Using bit shifts and bitwise operators for fast processing
  ○ Reordering the nodes we check in alpha-beta pruning to enable maximum pruning of the search tree.
Suggestions for future work

- Enhancing the evaluation function (improving the heuristic)
- Improving the program logic and enhancing the speed (improving depth). We were able to increase the speed by a factor of 2. Further improvements are possible.
- Expectimax pruning using Star1 algorithm
- Porting to a better UI (Web-based interface)
References

All the game code and search written is original except the GUI which is tkinter based.

We have used code written by Raphaël Seban for the GUI and interfaced it with our bot to give an intuitive demo of our bot. The code can be found here.

The inspiration for many of our approaches was found at this stackoverflow link.