Artificial Intelligence
2048 Game Contest

Goal Write an agent to play 2048.

GitHub classroom You can get the codebase using GitHub classroom. All submissions should be made by pushing to the repository that it creates for you.

Due Monday, Jun 23 at 5am

Performance You agent's performance will be judged by it's average performace player 2048. It will be run at least 10 times with the mean and geometric means calculated.

Leaderboard You can find the leaderboard at `https://cs.slu.edu/~letscher/ai/contest3.html`. Approximate grades on a 100 point scale will be posted.

Files You should create an agent file MyAgent.py to implement your solution. If you have a data file it should be in the filename MyData.py.

Running the program Using Python run `Play.py`. It will tell you the command line options that it expects.

Write-up You must document your entire development process. This includes testing results, proposed improvements and design. Your write-up should be committed to your repository as Development.pdf.

Intermediate versions Every version of your agent should be committed to your git repository and documented.

Game2048 class Internally, the board game is represented by a vector of length 16 in row-major order. The values in the vector are from 0-15. With 0 representing an empty location and the value of i representing a tile with value 2^i.

Here are the methods you might need to use:

- The constructor takes optional parameters of the tile locations and current score.
- **randomize()** initializes the game to a random starting state.
- **actions()** the legal moves from the current state stored as a string.
- **result(a)** the result of applying a to the current state. Returns a new game state. Note that this does both the shifting/merging of tiles and places a new random tile.
- **getScore()** return the current score
- **getTile(r,c)** get ths tile value (0-15) at that position of the board.
- **possibleResults(a)** returns a list of all the possible results of taking the action a and their probabilities of occuring. The list contains tuples with the game state and the probability.
- **possibleTiles()** returns a list of possible tile locations that could be added to the board. It returns a list of tuples, each storing the position of the board (as a number 0-15) and the value that can be added (represented as 1 or 2).

- **addTile(t, v)** adds a tile with value v in the array at index t.
- **move(a)** returns a new game state with the tiles shifted and merged, but no new tile added.
- **rotate(n)** returns a new game with the board rotated n quarter turns.
- **gameOver()** checks if the game is over.