Matthew Stroble

Bonus #3

The main heuristic for my program is consistent. First by calculating the spawn cost frequency for every number in the spawn pool (done in main), this generates a total cost it takes to get a specified value out of the spawn pool and stores them in a dictionary. The main heuristic predictCost takes in the goal and recursively breaks down the goal by generating the subsequent cost of all of the numbers needed to be spawned to get the current number. If 2048 is the goal the heuristic generates the cost needed to get a 1024, if a 1024 is already on the board, the cost for that number is 0 as its already exists. To get a second 1024 that does not exist yet, it recursively calls itself searching for the cost of the 512… 128… until it reaches the minimum number in the spawn pool. Since the recursion is memorized the incurred costs of the numbers from 2048 are only generated a single time they are encountered. In the main heuristic function a discount number is applied to offset duplicate spawns as the binary tree of generated number branches to account for numbers that have combined already. This is done by   
2^(b-a) – ( b-a) / current highest value. The main reason I divide by the current highest value is to prevent the heuristic from becoming negative. As the moves are made at no time does the subsequent children ever have a heuristic value higher than that of the parent. This is mainly due to calculating the internal discount cost of zero to the numbers that already exist on the board. In theory it should always be consistent and never go negative. The initial overestimate is due to the system not being able to see the future nodes that combine but once the first two combinations occur it never overestimates again.