Jewels (BFS and DFS Search Exercise)

Andrew Smiley

The jewels problem is stated as follows. Given a 3x3 grid of jewels, select specific jewels with your magic wand to change that jewel and its adjacent jewels (above, below, left, right) from the current jewel to the next jewel in the sequence. Jewels transition from diamond to ruby, ruby to emerald, and emerald to diamond. Your goal is to translate a given 3x3 grid (the start state) to a specified goal state by determining which jewels to tap with your wand and in which sequence.

Output

```
Found [['E', 'E', 'E'], ['E', 'E', 'E'], ['E', 'E', 'E']] in Depth First Search:
Took 3446 moves of 5165 states visited


Found [['E', 'E', 'E'], ['E', 'E', 'E'], ['E', 'E', 'E']] in Best First Search:
Took 5 moves of 6 states visited


Found [['E', 'E', 'R'], ['E', 'R', 'E'], ['R', 'E', 'R']] in Depth First Search:
Took 14397 moves of 21592 states visited


Found [['E', 'E', 'R'], ['E', 'R', 'E'], ['R', 'E', 'R']] in Best First Search:
Took 87 moves of 135 states visited


Found [['R', 'R', 'E'], ['R', 'E', 'R'], ['E', 'R', 'R']] in Depth First Search:
Took 13530 moves of 20293 states visited


Found [['R', 'R', 'E'], ['R', 'E', 'R'], ['E', 'R', 'R']] in Best First Search:
Took 180 moves of 247 states visited


Found [['R', 'D', 'R'], ['D', 'R', 'D'], ['R', 'D', 'R']] in Depth First Search:
Took 10706 moves of 16055 states visited


Found [['R', 'D', 'R'], ['D', 'R', 'D'], ['R', 'D', 'R']] in Best First Search:
Took 473 moves of 630 states visited
```

Reporting

A short report that describes who in your group implemented which portions (if you worked in a group), what you learned from the assignment, how easy or challenging the assignment was

To be completely honest, I enjoyed this assignment. It gave me a good refersher on recursion, something I don't utilize often in the professsional world. As far as learning goes, I learned how much you can improve search by utilizing an heuristic function. As far as challenge goes, I wouldn't say the assignment was too challenging, although it did require a decent amount of critical thinking given the recursive nature of the algorithms as well as the affect of a move on the board itself.

An explanation of your heuristic and whether you felt it improved the search and whether a better heuristic might help

```
#our hueristic function
#node: the node we are calculating score for
#goal_state: the desired goal state
def hueristic(node, goal_state):
    #total score for incrementing
    total_score = 0
    #well, if this node is our goal give it the highest score
    if node.data == goal_state:
        return pow(9, 5)
    #generate the children
    generate_children(node)
    #if the children of this node contain the solution, it's the next best solution
    if len (filter(lambda x: x.data == goal_state, node.children)) >0:
        return pow(9,4)
    #kill the children \M/
    del node.children[:]
    #iterate through and calculate a given score based upon the number of steps it wi
ll take a space to reach the goal space value
    for y in range(0, len(node.data)):
        for x in range(0, len(node.data[y])):
            distance_to_goal = get_space_distance(node.data[y][x], goal_state[y][x])
            #if the next move will get us there, we want to give the highest value
            if distance_to_goal == 1:
                total_score = total_score+4
            else:
                #otherwise, we score the distance, which will not be > 3 moves
                total_score = total_score+distance_to_goal
    return total_score
```

What It Does

Essentially, what my heuristic does is done in 3 parts. The first, is to determine if the node we are currently looking at is the goal state. This will return the highest score. The next, is to generate the children of the node we are looking at, if the a child of the current node contains the goal state, we return the second highest score for this node. The two aforementioned scorings ensure that we always select the node that is the goal state or the node in which the goal state will be in the next recursion. Finally, since we've generated children for the node (and python being a pass by reference language), we kill the children of the node so we do not generate duplicate children for the node. Finally, if neither of the above options are hit, we look space by space to determine how close a given space on the grid (space= grid[y][x]) to the goal state at that same position. I.e. if `node.data[y][x] == 'D'` and `goal_state[y][x]=='E'` then the score for that space is 1, or how many moves until the goal state is reached

Did It Improve Search?
In short, yes. I believe the heuristic improved the search. However, don't take my word for it, the resulting data speaks for itself. In some cases, boasting a 2223% improvement over DFS.

Would A Better heuristic Help?
A better heuristic would have helped, absolutely. Time permitting, I would have liked to tweak the 3rd check in the heuristic to not look at individual spaces, but rather look at moves as a whole by their bounds. There is a function in `jewels.py` called `find_bounds(pos)` that determines, given a position, what spaces around it will be impacted. I would like to improve the heuristic to not look at individual spaces, but rather entire moves to determine what percentatge of the next move would be a part of the goal state. However, this is not entirely accurate given that the bounds of a move can overlap with other moves, howevr, it would be a first step.