Brennan McFarland
October 22, 2017
Programming Assignment 1 Extra Credit Writeup

The only major change to the codebase from the original assignment is the addition of the PocketCube class. Like EightPuzzle, it inherits from NodeStateData, and is represented by a 24 element tuple representing the faces of the cube with the indices of the tuple mapped to the faces of the cube like so:

```
          0  1
          2  3
 4  5  6  7  8  9  10 11
12 13 14 15 16 17 18 19
          20 21
          22 23
```

The contents of the faces tuple itself is the first letter of a color in the classic rubik's cube color scheme: white, yellow, orange, red, green, and blue. PocketCube's implementation is analogous to EightPuzzle with a few exceptions.

```
def __get_faces(self):
  face1 = self.__faces[0:4]
  face2 = self.__faces[4:6] + self.__faces[12:14]
  face3 = self.__faces[6:8] + self.__faces[14:16]
  face4 = self.__faces[8:10] + self.__faces[16:18]
  face5 = self.__faces[10:11] + self.__faces[18:19]
  face6 = self.__faces[20:]
  return {face1, face2, face3, face4, face5, face6}
```

The get_faces method simply slices the faces tuple into 4-element arrays representing each side of the cube for easier processing. goal_test uses this, testing to see if every face contains only one color. The methods for calculating moves are r, rprime, d, and dprime, named according to common rubik's cube move notation. Any other moves are simply a form of these viewed from a different angle, so they are all that is needed. r rotates the right face, whereas d rotates the downward face, with prime indicating the opposite direction. Their states are calculated by sequentially switching the faces in the tuple. For example, the r move is calculated by switching faces 4 through 10 with its neighbor, which results in the entire third "row" of the net to be shifted over by one, equivalent to rotating around the right side of the cube.

Both h1 and h2 call the same heuristic for the pocket cube, which is calculated as the maximum number of different faces on a side minus 1 (since the goal state would have 1 type of face on each side).

The commands are the same as for the EightPuzzle, with the exception that “setMode pocket-cube” and “setMode 8-puzzle” can be used to switch between the two modes. The application is set to 8-puzzle mode by default. Entering h1 or h2 for A* will run the same heuristic mentioned above.

The pocket cube heuristic is correct because for the side containing the maximum unique faces, each face must be independently moved to the correct side as faces on that side can only be moved to another side one side at a time. For example, if there were red, green, and blue faces on the maximum side, a move can only put red on the red side, green on the green side, or blue on the blue side (though it may take more moves depending on the configuration of the cube). Thus, the heuristic never overestimates the remaining path cost and is therefore admissible.

According to the test file (writeup_extra_credit1), both algorithm implementations appear to perform poorly even when reasonably close to the goal state, as both the A* and local beam searches exceeded 10,000 nodes with as few as 5 moves away from the goal state. However, for the second test file (writeup_extra_credit2), it can be seen that A* can solve for states that are 10 moves away from the goal state in around .4 seconds. It is clear that the heuristic for the pocket cube is not ideal, and though it is able to solve the pocket cube, for most of the possible starting configurations it is too slow to be practical.