TASK 1 DOCUMENTATION

Team Members:

- Bernalte Sánchez, Raúl
- Rodríguez-Barbero González, Aarón
- Rodríguez Villafranca, Enrique

Repository name: A1-05

- raulbs7
- Aharon1377
- ERV501

Language chosen:

We have decided to use Python as the programming language for our project due to its versatility and simplicity, just as the used libraries have proven to be

Structure of the Cube:

The structure of the cube consists of an MD5 identifier (as required), whose structure is a String, and the representation of the cube's faces which has been implemented as a dictionary of lists, each list containing a face of the cube.

The reasoning why we implemented this specific structures was in hopes of the higher efficiency and comprehension level as well as the lowest memory waste.

Code:

In this section, we are going to explain the main functionalities of the most important parts of the code:

Class Cube:

```
def __init__(self, id, faces):  # constructor
    self.id = id
    self.faces = faces
    self.dict_colours = {0: 'white', 1: 'yellow', 2: 'red', 3: 'orange', 4: 'blue', 5: 'green'}
    self.dict_faces = {0: 'UP', 1: 'DOWN', 2: 'FRONT', 3: 'BACK', 4: 'LEFT', 5: 'RIGHT'}
```

Our Cube is defined by

-id: it's the md5 code that contains the current cube's faces codified

-faces:

position of each face in the following format →


```
{
  "BACK": [[4,4,4],[3,3,3],[3,3,3]],
   "DOWN": [[1,1,1],[2,4,4],[1,1,1]],
   "FRONT": [[2,2,2],[2,2,2],[5,5,5]],
   "LEFT": [[2,4,4],[0,0,0],[2,4,4]],
   "RIGHT": [[5,5,3],[1,1,1],[5,5,3]],
   "UP": [[0,0,0],[5,5,3],[0,0,0]]
}
```


-dict_colours: python dictionary used in the graphical representation of the cube, it contains the color-number relation

-dict_faces: python dictionary used in the movement of the faces

Method generateMoves:

Here we compute the valid methods for the cube, depending on its dimensions (NxNxN)

***Method move*:**

Performs the select move:

```
if (mov == "L"):
    self.moveL(layer, inv, aux, length)
if (mov == 'l'):
    for i in range(3):
        self.moveL(layer, inv, aux, length)
if (mov == "D"):
    self.moveD(layer, inv, aux, length)
if (mov == 'd'):
    for i in range(3):
        self.moveD(layer, inv, aux, length)
if (mov == "B"):
    self.moveB(layer, inv, aux, length)
if (mov == 'b'):
    for i in range(3):
        self.moveB(layer, inv, aux, length)
```

***Method printState*:**

Draws and prints the current state of the cube (using Turtle library) taking into account the positions and the colors we specified in the dictionary “dict_colours”

***Method turnRight*:**

Rotates a face of the cube 90º to the right

```
matrix = self.faces[self.dict_faces[index]]

res = [[0 for i in range(len(matrix))] for j in range(len(matrix[0]))]
aux = [0 for i in range(len(matrix))]

for i in range(len(matrix)):
    for j in range(len(matrix[0])):
        aux[j] = matrix[i][j]

    for j in range(len(matrix)):
        res[j][len(matrix) - 1 - i] = aux[j]

for i in range(len(matrix)):
    for j in range(len(matrix)):
        self.faces[self.dict_faces[index]][i][j] = res[i][j]
```

Method cubeString:

Creates the string with the current cube state

Method cubeMD5:

Codifies the previously created String into md5

Method json2cube:

Opens the json file we are provided and transforms it into an object cube

Method cube2Json:

Writes into a new json file the state of the cube

Class Frontier:

Defines our frontier

Contains the insert method in order to include nodes in the frontier, and the remove method in order to pop them

Class TreeNode:

Definition of the cube's node

```
def __init__(self, state, cost, action, d, f, parent=None):
    self.state = state
    self.parent = parent
    self.cost = cost
    self.action = action
    self.d = d
    self.f = f
```

Class Sucessors:

Generates the sucessors of the cube state

```
    ret = [Cube(None,None) for i in range(12)]

    state.cube2Json("state.json")# create a new json file with the a copy of the cube's state

    for i in range(12):
        ret[i].json2cube("state.json")# reads the previously created json file with the last state

    moves = iter(state.generateMoves())

    for i in range(12):

        movement = next(moves)

        ret[i].move(movement)
        print(movement,ret[i],1)

    return ret
```