Artificial Intelligence: Assignment 1

Seung-Hoon Na

April 13, 2024

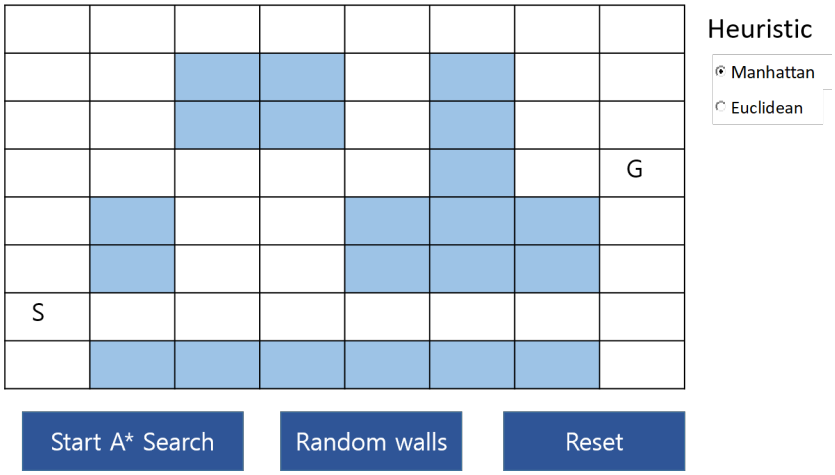
1 A* Algorithm

본 과제에서는 $M \times N$ Grid world에서 장애물이 랜덤(random)하게 배치되고, 시작 지점에서 장애물을 피해 목표지점까지 도달하는 최단 경로를 찾는 A* 알고리즘을 구현하고 이를 GUI환경에서 시뮬레이션 하는 것이 목표이다 (**python 코드로 구현**). 이동은 diagonal방향은 허용하지 않고 상하좌우 4방향으로만 가능하다.

- 참고예제: `https://qiao.github.io/PathFinding.js/visual/`

GUI환경은 다음과 같이 $M \times N$의 Grid world, 3개의 버튼과 2개의 Ratio button으로 구성되도록 하고, 장애물은 gray color로 표시한다. 보드를 구성하기 위해 다음의 **Pygame** 및 python GUI library인 **PGU**를 이용하라.

- Pygram: `https://www.pygame.org/news`
- PGU: `https://github.com/parogers/pgu`



다음은 상세 요구사항이다.

- **Grid world 크기**: Grid cells의 행과 열의 갯수 M과 N의 default 값은 30이며, `argparse`를 이용하여 command line에서 입력받을 수 있도록 한다.

  - Argparse참조: `https://docs.python.org/3.3/library/argparse.html`

- **Grid world 화면 출력**: 전체 Grid world의 화면 크기는 픽셀단위로 고정으로 하며 (예: 600×600), 각 Grid cell의 화면상 크기는 M과 N의 값에 따라 동적으로 바뀌도록 한다. 즉, default셋팅에서 Grid world의 크기가 600×600이고 $M = N = 30$일 때, 각 grid cell의 크기는 20×20이 된다.

- **장애물 - 랜덤 배치**: 장애물은 사용자가 마우스 클릭하여 배치하는 방법이 있고, random하게 배치하는 방법 있다. 먼저 랜덤 배치에서는, $M \times N \times$ *inc_obstacle_ratio* 갯수만큼의 장애물이 비어있는 grid cell에 새롭게 생성된다. 이때, *inc_obstacle_ratio*의 default값은 0.2로 하고, $M = N = 30$ 이라면, $M \times N \times inc_obstacle_ratio = 120$개의 장애물이 새롭게 생성된다. *inc_obstacle_ratio*의 값은 `argparse`를 이용하여 command line에서 옵션으로 입력받을 수 있도록 한다.

- **장애물 - 사용자 배치**: 다음으로 사용자 배치에서는, grid world상의 마우스 클릭된 grid cell을 *focus cell*이라 할때 해당 cell이 비어있다면 장애물로 바뀐다 (즉, gray color로 배경색 변경). 만약 해당 focus cell이 장애물이었다면 toogle되어 다시 비어있는 cell로 바뀐다.

- **시작지점과 목표지점**: 시작지점과 목표지점은 초기에 랜덤하게 또는 고정위치에 S와 G로 표시되도록 하고, 사용자 마우스 drag를 통해 이동될 수 있도록 한다.

- **A* Search 수행**: **Start A* Search** 버튼 클릭시 S와 G에 A* search를 수행하여, 최단경로를 찾을 경우 해당 경로를 **yellow line**으로 표시한다. 또한, 콘솔화면에 해를 탐색할때까지의 총 explored nodes갯수를 출력한다.

- **A* Search Heuristic 함수**: 휴리스틱 함수로 현재 node의 위치와 gold node위치간의 1) *Manhattan 거리*, 2) *Euclidean 거리* 두 가지중 하나를 화면 우측의 ratio button을 통해 선택할 수 있도록 한다.

- **A* Search 결과 - 해가 없을 때** : 만약 A* search 결과 해가 없다면, 콘솔화면에 path가 발견되지 않았다는 메시지를 출력하고, GUI 화면에서는 explored nodes중 가장 $f(n)$이 낮은 node까지의 경로를 **yellow line**으로 표시한다.

2 Pocket Cube: $2 \times 2 \times 2$ Rubik's Cube

본 과제에서는 $2 \times 2 \times 2$ Rubik's Cube가 랜덤하게 셋팅되고, 이를 IDA* 알고리즘을 이용해서 Cube문제를 풀고, 이를 pygame를 통해 시물레이션 하는 것이 목표이다.

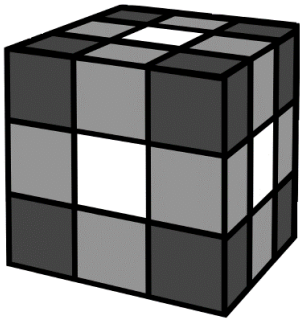
Rubik's Cube에서 사용되는 **cubie**, **edge**, **corner**용어의 의미는 다음과 같다.

- **Cubie**: Cube를 구성하는 단위 sub-cube를 의미한다. $2 \times 2 \times 2$ Rubik's cube에서는 총 8개의 cubie, $3 \times 3 \times 3$ Rubik's cube에서는 총 26개의 cubie들로 구성된다 (중앙 hidden cubie제외).

- **Edge**: Cubie의 한 종류로 2개의 color가 보여지는 cubies를 가리킨다. 다음 그림에서 Light Grey.

- **Corner**: Cubie의 한 종류로 3개의 color가 보여지는 cubies를 가리킨다. 다음 그림에서 Dark Grey.

- **Center**: Cubie의 한 종류로 1개의 color만 나타나는 cubies를 가리킨다. 다음 그림에서 White

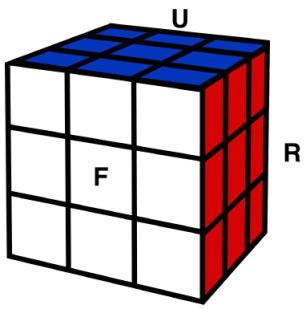


Cubie의 하나의 단면 (face)이 취할 수 있는 색상은 6개로 **R**, **G**, **B**, **W**, **O**, **Y** 이다.

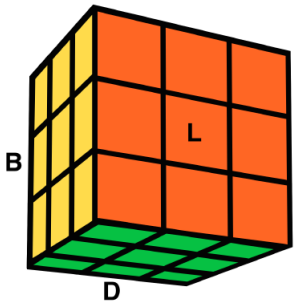
Rubik's Cube의 face와 move에 대한 표준 notation은 다음을 참조하시오.

`http://www.rubiksplace.com/move-notations/`

위의 표준 표기에 따르면, Rubik's Cube의 6개의 면(face)은 **F**, **B**, **R**, **L**, **U**, **D**로 지칭한다.



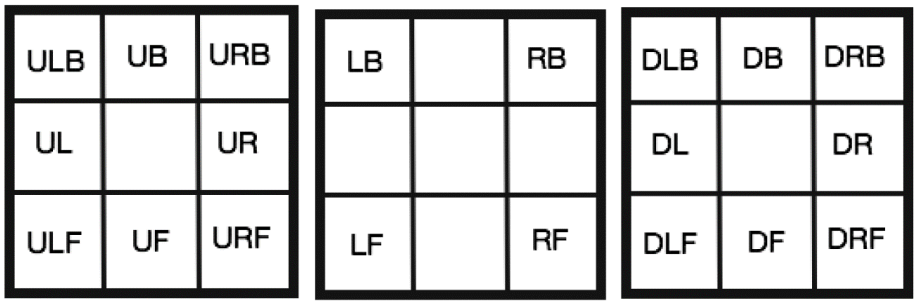
(a) 앞 면 (front side)



(b) 뒷 면 (back side)

- F **(front)**: the face facing the solver.
- **B (back)**: the back face.
- **R (right)**: the right face.
- **L (left)**: the left face.
- **U (up)**: the upper face.
- **D (down)**: the face opposite to the upper face.

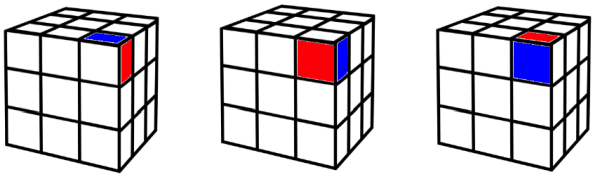
$3 \times 3 \times 3$ Rubik's Cube에서 Cubie는 총 26개로 구성되며 notation은 다음과 같다.



(c) 상단 레이어 (top layer) (d) 중간 레이어 (middle layer) (e) 하단 레이어 (bottom layer)

$2 \times 2 \times 2$ Rubik's Cube의 경우 Cubie는 총 8개로 구성되며, 모두 corner 유형으로, $3 \times 3 \times 3$ 표기에서 ULB, URB, ULF, URF, DLB, DRB, DLF, DRF notation만을 사용한다.

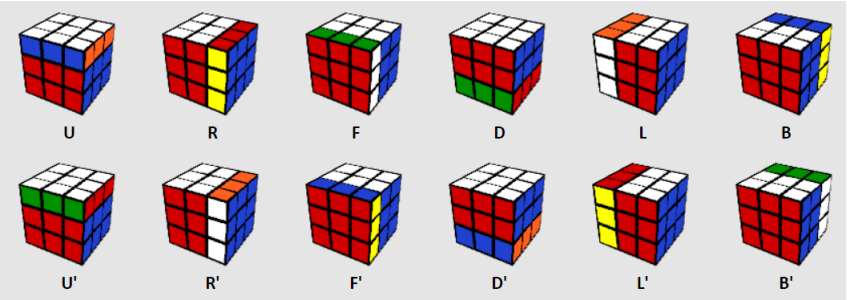
각 corner cubie는 방향에 따라 **3가지의 orientation type**이 가능하며, 다음과 같다.



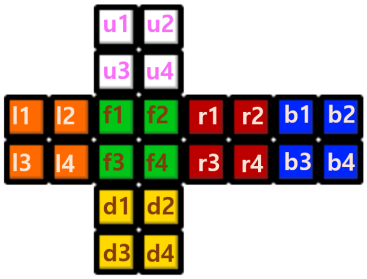
(f) Corner 0 (g) Corner 1 (h) Corner 2

즉, 위의 그림은 white가 F, U, R의 3가지 face에 위치할때의 orientation type을 보여주고 있다.

본 과제에서는 Rubik's Cube의 Move (turn)는 한번에 90도 회전만 허용하여, 6개의 면을 90도 시계방향, 반시계방향으로 하여 총 12개의 move 가능하다. 아래 그림과 같이 12개의 move를 위한 표기로, **F**, **B**, **R**, **L**, **U**, **D**은 해당 면의 시계방향 move를, **F'**, **B'**, **R'**, **L'**, **U'**, **D'**은 해당 면의 반시계방향 move를 지칭한다.



Pocket Cube ($2 \times 2 \times 2$ Rubik's Cube)의 상태는 다음 그림과 같이 총 24개의 단면의 색상들로 표현되고, 이는 각 문자가 color (**R**, **G**, **B**, **W**, **O**, **Y** 중 하나) 를 가리킬때, 총 24개의 문자로 구성된 문자열로 표시할 수 있다.



$$u_1u_2u_3u_4r_1r_2r_3r_4f_1f_2f_3f_4d_1d_2d_3d_4l_1l_2l_3l_4b_1b_2b_3b_4$$

예를 들어, 위의 goal 상태에 해당되는 state는 다음과 같다.

```
WWWWRRRRGGGGYYYYOOOOBBBB
```

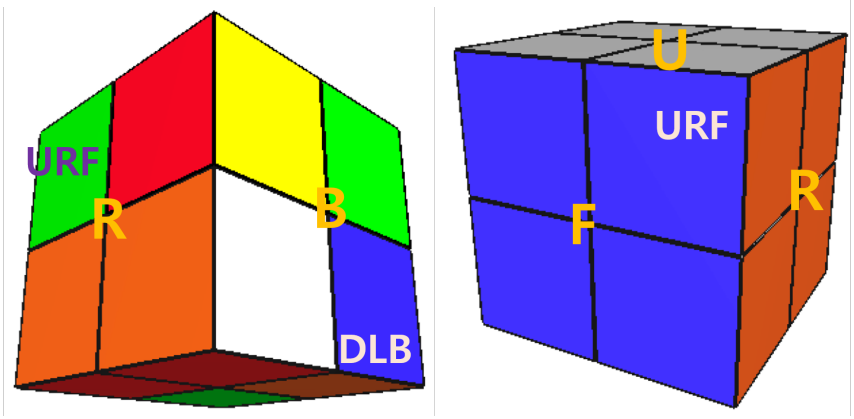
2.1 Heuristic

Pocket Cube를 풀기 위해, cubie별로 최소 이동수를 독립적으로 계산하여, 이들의 합 또는 최대치를 취하는 heuristic을 사용한다.

$minmove(cubie_i(state), cubie_i(goal))$는 i-번째 cubie ($cubie_i$)가 현재 $state$의 위치와 방향 (position, orientation)이 목표 상태인 $goal$의 위치와 방향이 되기 위해 필요한 **최소한의 move수**이다.

여기서 $cubie_i(state)$는 i-th cubie가 해당 state에서의 position, orientation으로 구성되는 tuple을 리턴한다 [1].

예를 들어, 다음의 예에서 현재상태의 DLB cubie를 목표상태의 URF의 위치로 이동하기 위해 필요한 **최소한의 move수**는 $minmove(\langle DLB, OBW\rangle, \langle URF, OBW\rangle)$로 3이 된다 (즉, 순서대로 L, L, F를 수행하면 됨) 이때 orientation은 시계방향으로의 각 face의 color를 나열하는 식으로 정의하였음.



(i) 현재 상태 ($state$) (j) 목표 상태 ($goal$)

[1] orientation은 시계방향으로의 각 face의 color를 나열하는 식으로 정의될 수 있다

1. **Summation 기반 heuristic**

$$h_{SUM}(state) = \frac{1}{4}\sum_{i=1}^{8} minmove(cubie_i(state), cubie_i(goal))$$

2. **Maximum기반 heuristic**

$$h_{MAX}(state) = \max_{1\le i\le 8} minmove(cubie_i(state), cubie_i(goal))$$

2.2 구현 내용

위의 Pocket Cube의 내용 및 heuristic참조하여, 주어진 Pocket Cube문제에 대해서 goal상태로 가기 위한 최적의 move steps를 얻는 IDA*알고리즘을 작성하고 이를 시뮬레이션 하시오. 이를 위한 세부 구현 내용은 다음과 같다.

- $minmove$ **함수 구현 및 테스트**:

  앞 절의 heuristic함수 구현을 위해 모든 cubie의 position ULB, URB, ULF, URF, DLB, DRB, DLF, DRF과 orientation 0, 1, 2에 대해 $minmove(\langle pos, ori\rangle, \langle pos', ori'\rangle)$를 breadth first search (또는 다른 search)로 계산하는 함수를 **python code**로 작성하시오. 만약 $\langle pos, ori\rangle$에서 $\langle pos', ori'\rangle$의 이동이 불가능하다면 -1을 리턴하도록 하시오.

  예를 들어, $minmove(\langle DLB, 0\rangle, \langle URF, 0\rangle) = 3$ [2].

  또한, 모든 가능한 $\langle pos, ori\rangle$, $\langle pos', ori'\rangle$에 대해 $minmove(\langle pos, ori\rangle, \langle pos', ori'\rangle)$를 미리 계산하여 pattern db로 저장하는 코드도 작성하시오. 즉, pos로 취할 수 있는 값은 8개, ori의 경우는 3개이므로 $\langle pos, ori\rangle$에 대해 총 24개의 경우가 있으며, 따라서 $minmove(\langle pos, ori\rangle, \langle pos', ori'\rangle)$는 24×24 크기의 table로 db화 될 수 있다.

- **Pocket Cube 퍼즐의 랜덤 초기화 및 파일로 저장**: Pocket cube 퍼즐 문제 제시를 위해 12개의 move – **F**, **B**, **R**, **L**, **U**, **D**, **F'**, **B'**, **R'**, **L'**, **U'**, **D'** 중 1개를 랜덤하게 택하여 이를 N번 수행한후, 결과 cube 상태를 퍼즐 문제를 파일로 저장하시오 (파일의 포맷은 적절히 정의하라.) Default로 N은 20 이상으로 하고, argparse를 통해 N은 option으로 지정할 수 있어야 한다.

  이때, cube 상태는 1) 각 cubie들의 위치와 방향 (8개의 pairs로 구성)또는 2) 총 24개 길이의 문자열로 표현할 수 있다.

- **Pocket Cube 퍼즐 Solver: IDA* 알고리즘 기반** : IDA* 알고리즘의 일반적인 코드를 구현하고, 앞절에서 정의된 $h_{SUM}(state)$, $h_{MAX}(state)$를 이용하여 IDA* 알고리즘을 적용하여 주어진 Pocket Cube 퍼즐에 대한 solution을 리턴하고 이를 저장하는 python 코드를 작성하시오.

  이때, solution리턴시 explored nodes갯수도 함께 로그 방식으로 출력하도록 하시오.

  Solution은 move들의 sequence이며, 코드 실행시 해를 찾으면, num of moves 및 move들의 sequence가 출력되어야 하며, 이후 상세하게 $state_1$,$move_1$, $state_2$, $move_2$, $\cdots$가 콘솔화면에 출력되어야 한다. 다시 말해 시작상태가 먼저 출력되고, 이후에는 각 move 유형 출력 (12개 중 하나), move적용 결과 cube상태 출력, 이들이 goal state가 될때까지 반복된다 (goal상태 출력)

[2] 여기서, orientation은 0,1,2이라고 가정하였다

이때, 마찬가지로, Cube 상태는 1) 각 cubie들의 위치와 방향 (8개의 pairs로 구성) 또는 3) 총 24개 길이의 문자열로 표현할 수 있다.

- **Pocket Cube 퍼즐 Solver: 시물레이션 및 gui구현 포함**: 주어진 Pocket Cube 퍼즐 문제 파일과, solution파일을 읽어들여 이를 시물레이션하는 pygame 코드를 작성하시오. 다음 `PyCube` 코드를 확장하면 된다.

  `https://github.com/myme/PyCube`

  시물레이션은 각 move별로 pygame상에서 cube가 회전하는 annotation이 수행되어야 하고, 버튼은 **play (stop)**, **prev**, **next**로 구성하도록 한다.

  1. **prev버튼 클릭시**: 현재 move를 취소하고 이전 상태로 돌아간다
  2. **next버튼 클릭시**: 다음 move를 수행하여 그 다음 cube상태로 진행한다.
  3. **play버튼 클릭시**: 자동으로 next가 계속 수행되며, 동시에 play 버튼은 즉각 stop버튼으로 변경 된다. 또한 stop버튼 클릭시 play동작이 멈추고 동시에 해당 버튼이 play버튼으로 바뀐다.

- **Pocket Cube 퍼즐 - 전체 통합**: 위의 Solver 시물레이션을 확장하여, 문제 random 생성 및 solution제시를 통합하여 시물레이션하는 pygame코드를 작성하시오.

  처음에 코드가 실행되면 $2 \times 2 \times 2$ Rubik's Cube가 화면에 나오고, 여기에 사용자로부터 button이나 키보드 입력을 받아, **new puzzle**, **solve**이 기능이 수행되도록 한다.

  1. **New puzzle 버튼** (또는 키보드 입력): 앞서의 Pocket Cube 퍼즐의 랜덤 초기화를 수행하여 변경된 cube모양을 화면에 보여준다.
  2. **Solve 버튼** (또는 키보드 입력): 주어진 Pocket Cube 퍼즐에 대한 IDA* 기반 solution를 구하고, 해당 solution적용하여 play를 자동 실행한다. Solution이 화면의 우측에 참고로 제시되도록 하라.
  3. **Simulation 관련 버튼**: 해 (Solution)를 화면에 play하고 이를 제어하기 위해 앞서와 마찬가지로 **play (stop)**, **prev**, **next**해당 기능은 그대로 포함시킨다.

Rubik's cube을 위한 A*알고리즘에 대한 보다 상세한 자료는 다음을 참조하시오.

- `http://www.diva-portal.org/smash/get/diva2:816583/FULLTEXT01.pdf`
- `https://www.aaai.org/Papers/AAAI/1997/AAAI97-109.pdf`
- `https://www.doc.ic.ac.uk/teaching/distinguished-projects/2015/l.hoang.pdf`

3 제출 내용 및 평가 방식

코드는 **python**으로 본 과제 결과물로 필수적으로 제출해야 내용들은 다음과 같다.

- **코드 전체**
- **테스트 결과**: 각 내용별 테스트 코드 및 해당 로그 또는 출력 결과.
- **결과보고서**: 구현 방법을 요약한 보고서.

본 과제의 평가항목 및 배점은 다음과 같다.

- **각 세부내용의 구현 정확성 및 완결성 (80점)**
- **코드의 Readability 및 체계성 (10점)**
- **결과 보고서의 구체성 및 완결성 (10점)**