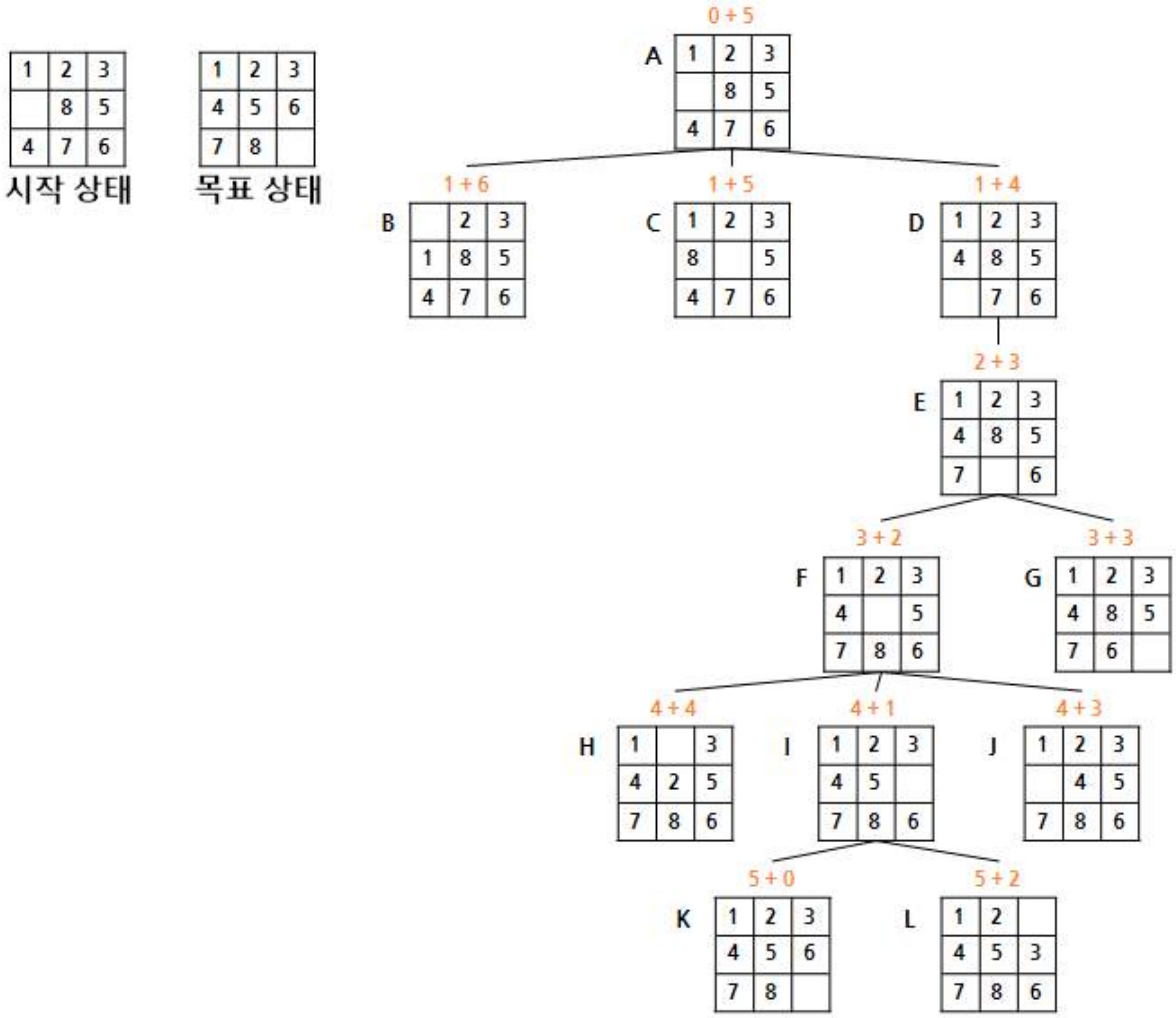
8-puzzle

(1) 8퍼즐문제 팀원별 휴리스틱 함수 구현

Default : 현재 상태와 목표 상태와 타일을 비교, 타일의 위치가 다르면 휴리스틱 값을 +1 해준다.

<참고 그림>



목표 지점을 찾기 전까지 큐에 들어가는 값

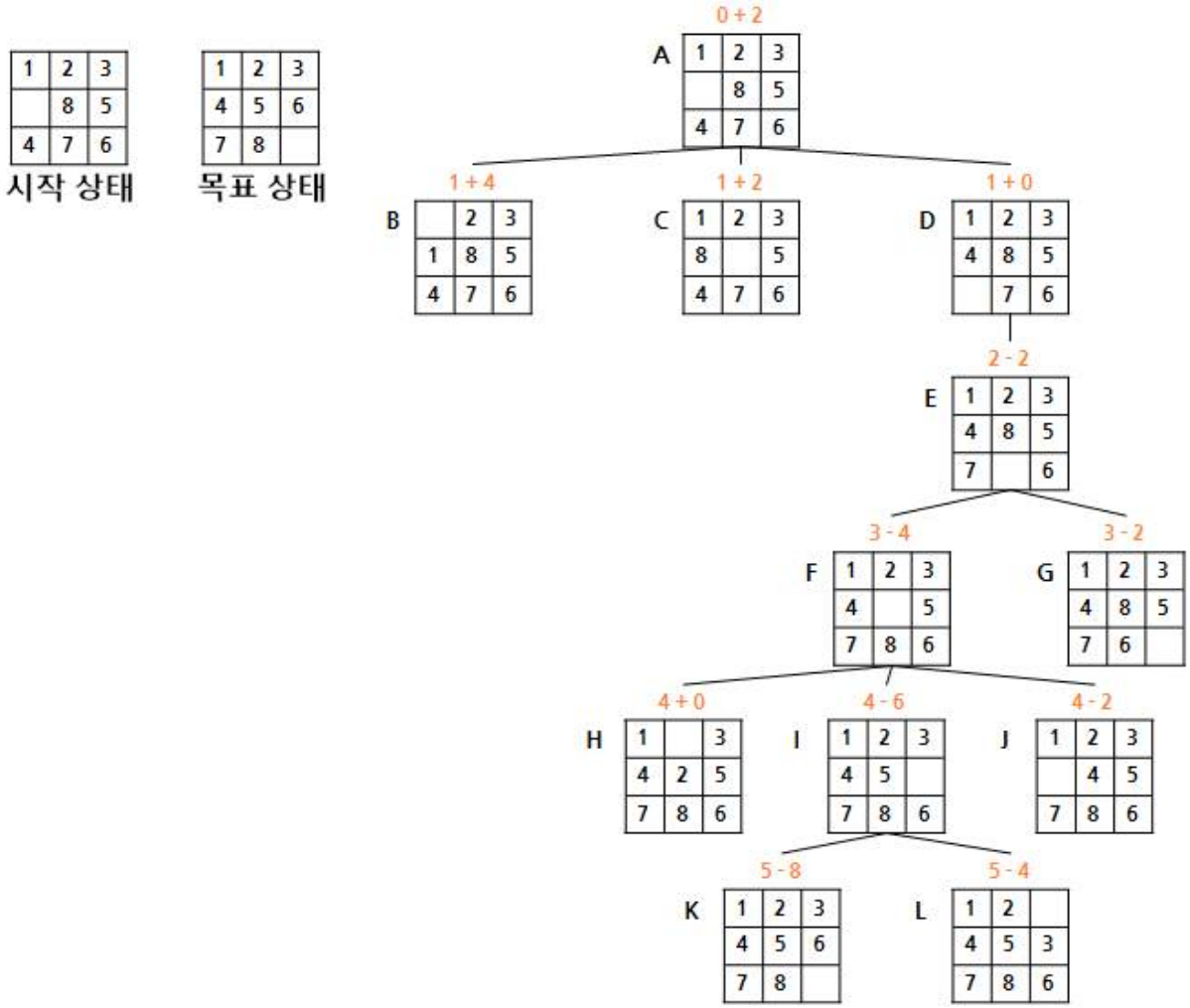
open	close
<A, 5>	
<D, 5> <C, 6> <B, 7>	<A, 5>
<E, 5> <C, 6> <B, 7>	<A, 5> <D, 5>
<F, 5> <C, 6> <G, 6> <B, 7>	<A, 5> <D, 5> <E, 5>
<I, 5> <C, 6> <G, 6> <B, 7> <J, 7> <H, 8>	<A, 5> <D, 5> <E, 5> <F, 5>
<K, 5> <C, 6> <G, 6> <B, 7> <J, 7> <L, 7> <H, 8>	<A, 5> <D, 5> <E, 5> <F, 5> <I, 5>
<C, 6> <G, 6> <B, 7> <J, 7> <L, 7> <H, 8>	<A, 5> <D, 5> <E, 5> <F, 5> <I, 5> <K, 5>

윤승회

함수 내용 :현재 상태와 목표 상태와의 위치가 다른 타일의 개수를 계산한다. 위치가 다른 타일이 있으면 휴리스틱 값을 1 증가하고, 위치가 다른 타일이 있으면 1 감소한다.

- 타일의 위치가 다른 타일과 위치가 동일한 타일도 고려하기 때문에 A*알고리즘을 통해 탐색할 때 더 깊은 탐색을 할 수 있다.

<참고 그림>

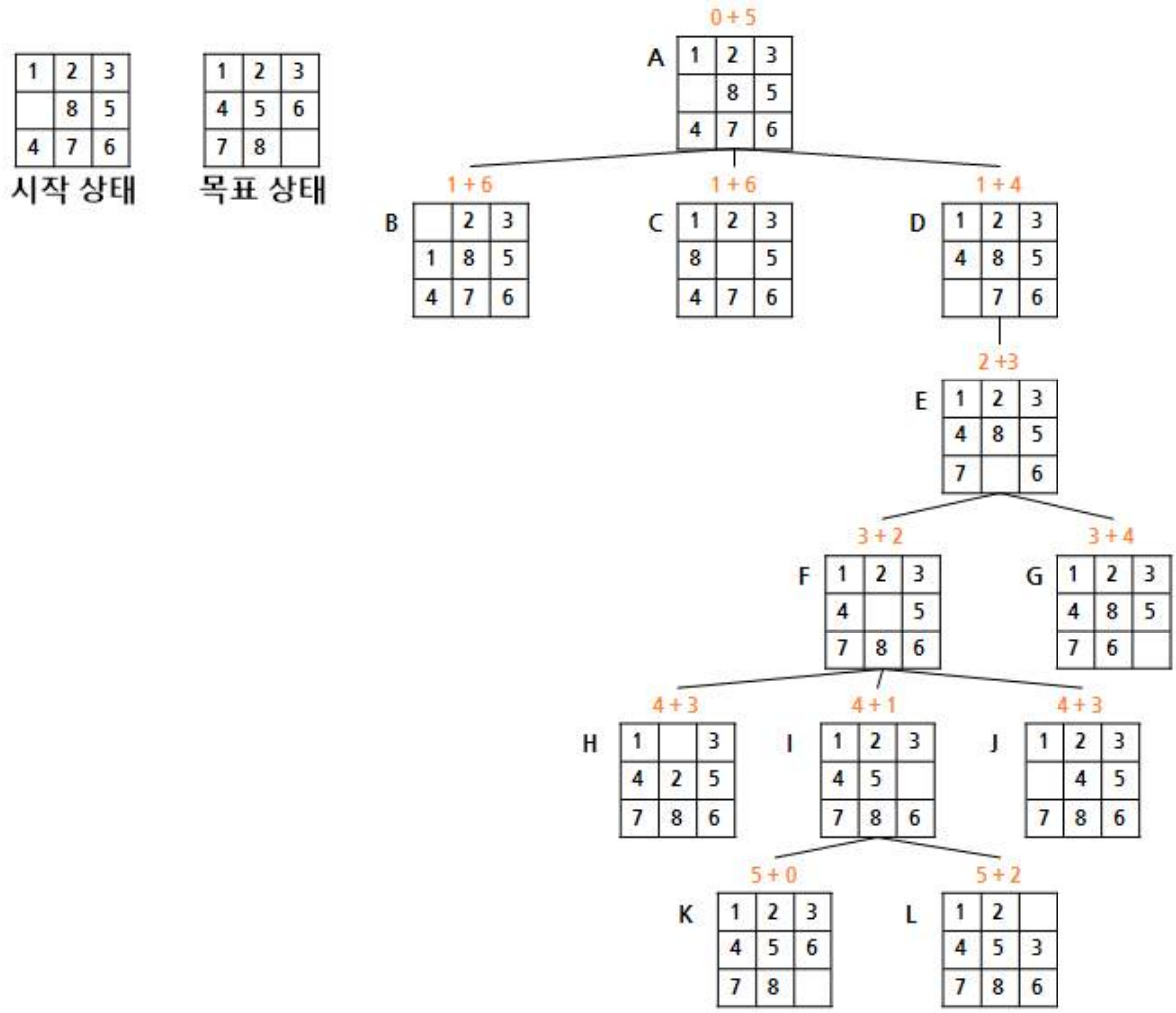


목표 지점을 찾기 전까지 큐에 들어가는 값

open	close
<A, 2>	
<D, 1> <C, 3> <B, 5>	<A, 2>
<E, 0> <C, 3> <B, 5>	<A, 2> <D, 1>
<F, -1> <G, 1> <C, 3> <B, 5>	<A, 2> <D, 1> <E, 0>
<I, -2> <G, 1> <J, 2> <C, 3> <H, 4> <B, 5>	<A, 2> <D, 1> <E, 0> <F, -1>
<K, -3> <G, 1> <K, 1> <J, 2> <C, 3> <H, 4> <B, 5>	<A, 2> <D, 1> <E, 0> <F, -1> <I, -2>
<G, 1> <K, 1> <J, 2> <C, 3> <H, 4> <B, 5>	<A, 2> <D, 1> <E, 0> <F, -1> <I, -2> <K, -3>

차영원 : 현재 상태와 목표 상태 타일을 비교하여 각각의 타일의 위치 차이의 합으로 휴리스틱 함수 구현.
- 각 타일의 목표 위치를 구하고 현재 좌표를 구해 둘의 차이를 계산하여 거리를 구함.

<참고 그림>

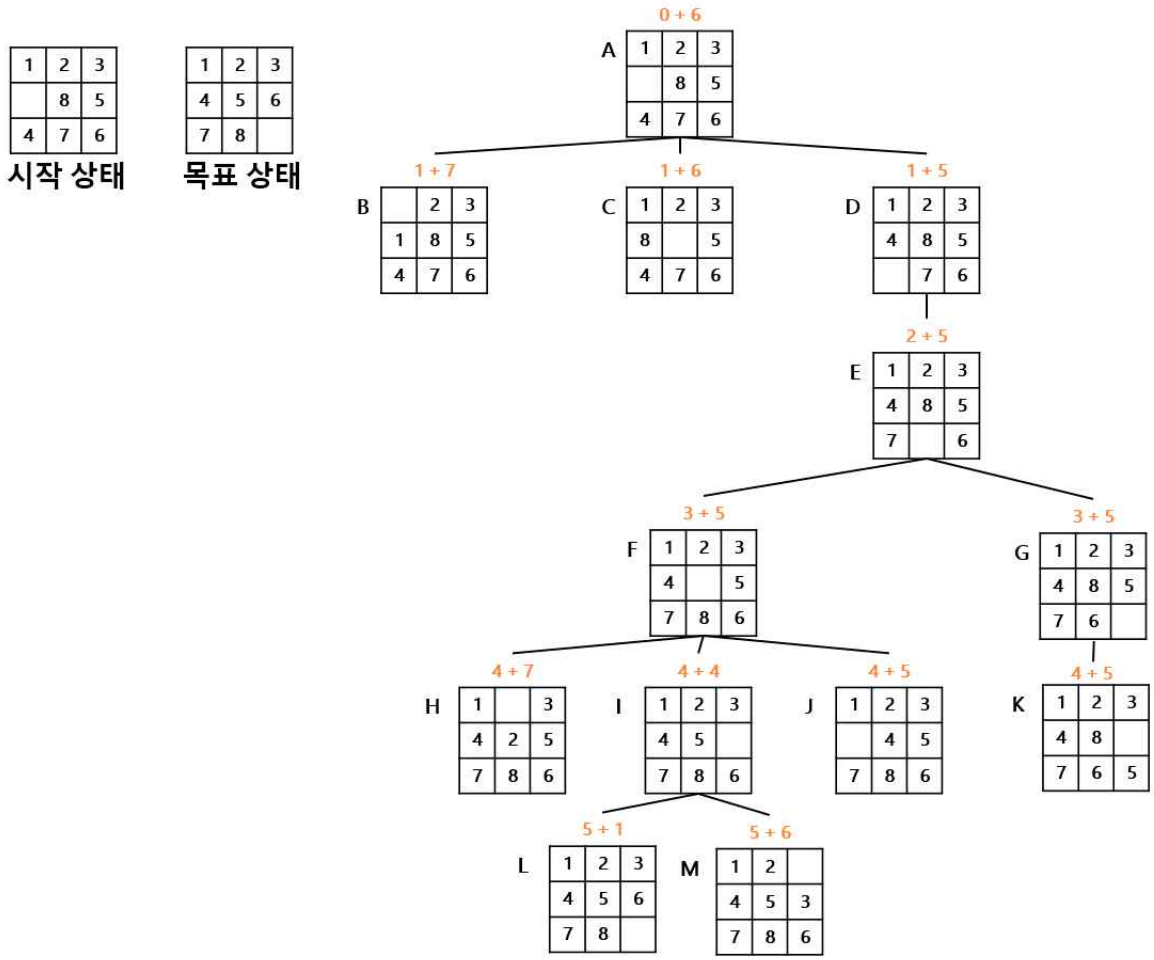


목표 지점을 찾기 전까지 큐에 들어가는 값

open	close
<A, 5>	
<D, 5> <B, 7> <C, 7>	<A, 5>
<E, 5> <B, 7> <C, 7>	<A, 5> <D, 5>
<F, 5> <B, 7> <C, 7> <G, 7>	<A, 5> <D, 5> <E, 5>
<I, 5> <B, 7> <C, 7> <G, 7> <H, 7> <J, 7>	<A, 5> <D, 5> <E, 5> <F, 5>
<K, 5> <B, 7> <C, 7> <G, 7> <H, 7> <J, 7> <L, 7>	<A, 5> <D, 5> <E, 5> <F, 5> <I, 5>
<B, 7> <C, 7> <G, 7> <H, 7> <J, 7> <L, 7>	<A, 5> <D, 5> <E, 5> <F, 5> <I, 5> <K, 5>

김예훈 : 다음 칸에 있는 숫자와의 차이가 -1이 아닌 노드의 개수로 휴리스틱 함수 구현.
- ex) A상태 : list[0]와 list[1]를 비교 값이 1 차이이므로 변화 없음.
: list[3]와 list[5]를 비교 값이 1 차이가 아니므로 휴리스틱 값 1 증가.

<참고 그림>



목표 지점을 찾기 전까지 큐에 들어가는 값

open	close
<A, 6>	
<D, 6> <C, 7> <B, 8>	<A, 6>
<E, 7> <C, 7> <B, 8>	<A, 6> <D, 6>
<G, 8> <F, 8> <C, 7> <B, 8>	<A, 6> <D, 6> <E, 7>
<K, 9> <J, 9> <I, 8> <H, 11> <C, 7> <B, 8>	<A, 6> <D, 6> <E, 7> <F, 8> <G, 8>
<M, 11> <L, 6> <K, 9> <J, 9> <H, 11> <C, 7> <B, 8>	<A, 6> <D, 6> <E, 7> <F, 8> <G, 8> <I, 8>

(2) 8-puzzle 토론 내용
- 기존 제공된 코드는 복잡하지 않는 경우 금방 찾을 수 있다는 장점이 있음. 하지만 복잡한 경우 너비보다 깊이에 더 비중을 많이 두는 경우가 발생함.

O. 너비를 더 탐색하기 위해서 다음 숫자와의 차이가 -1이 아닌 노드의 개수로 휴리스틱을 채택하기로 함.

O. 3x3의 형태를 하나의 리스트로 본다면 목표 상태가 1, 2, 3, 4, 5, 6, 7, 8 으로 볼 수 있으나 목표 상태가 리스트에 순서대로 오지 않는 경우도 생각해 고려해야 함.

O. 더 다양한 목표 상태에 적용하기 위해서 목표 상태인 숫자들의 위치 파악하는 함수를 추가함.

O. 목표 상태의 1 ~ 8 까지 순서대로 위치하는 index를 다른 리스트에 미리 저장하여 하드코딩을 해결함.

2. Tic-Tac-Toc

(1) 틱택토 게임 알고리즘 아이디어
- 틱택토 게임에서 가질 수 있는 모든 경우의 수는 9*8*7*6*5*4*3*2*1 로 총 362880 경우.
- 사용자가 먼저 시작하고 컴퓨터가 나중에 착수하는 방식으로 게임을 진행.
- min-max 알고리즘을 사용하여 컴퓨터가 두는 곳의 수를 모두 확인하여 최적의 위치를 두게 설계.
- look ahead를 설정하여서 min-max보다는 적게 확인하고 만일 해당 look ahead안에 게임의 승패가 나오지 않는다면 랜덤한 위치에다 두도록 설계.
- 알파 베타 가지치기 알고리즘으로 min-max와 같은 깊이로 확인하지만 탐색하지 않아도 되는 부분을 탐색하지 않도록 설계.

(2) 컴퓨터와 인간 대결 틱택토 게임 구현
- 사용자가 두는 곳은 X로 표시, 컴퓨터가 두는 곳은 O로 표시
- look ahead를 많이 보면 min-max와 동일한 형태로 탐색하게 됨. 많이 적게 보면 컴퓨터가 질 가능성이 현저하게 떨어지기 때문에 min-max 방식을 채택.
- min-max 알고리즘의 단점은 총 362880의 경우를 모두 탐색하는 단점을 지님.
- min-max는 메모리를 많이 차지하고 속도가 느려 알파 베타 가지치기으로 구현함.

윤승회 : 목표 상태가 특정 값으로 결정되어 있는 경우에만 해결할 수 있어 모든 목표 상태에 대하여 동작할 수 있도록 General한 함수를 작성

김예훈 : TIC-TAC-TOC의 min-max 알고리즘을 사용하여 컴퓨터가 수를 두는 논리 구현 및 알파 베타 가지치기로 코드를 수정하여 작성

차영원 : 테스트 화면에서 숫자 0 와 컴퓨터가 착수하는 영어 O의 차이를 명확하게 구분하기 어렵고 많은 사람들이 숫자는 1부터 시작하는 점을 고려하여 UI를 유저친화적으로 코드를 작성