기하 활용 문제 해결 2

기하 관련 지식 활용해 SW 문제 풀이하는 방법 익히기

01. 문제에서 사용하는 정의 이해: 정수 좌표 가지는 직각삼각형

02. 첫 번째 알고리즘

03. 두 번째 알고리즘

04. 세 번째 알고리즘

05. 지금까지 본 알고리즘의 구현 방법 이해

06. 실습 문제 풀이 & 질의 응답

<왜 기하/수학의 활용 예를 보는가? >

- **SW/HW 시스템 설계** 시 **수학 활용**하여 문제 해결되는 경우 많음
- 적절한 수학 활용 (기하 포함) → (i) **좋은 성능의 시스템** 설계 + (ii) **설계 시간 단축**
- 기하 관련 지식은 컴퓨터 그래픽 분야 및 수학 활용 SW 개발에 많이 활용됨 (예: Octave, Matlab)

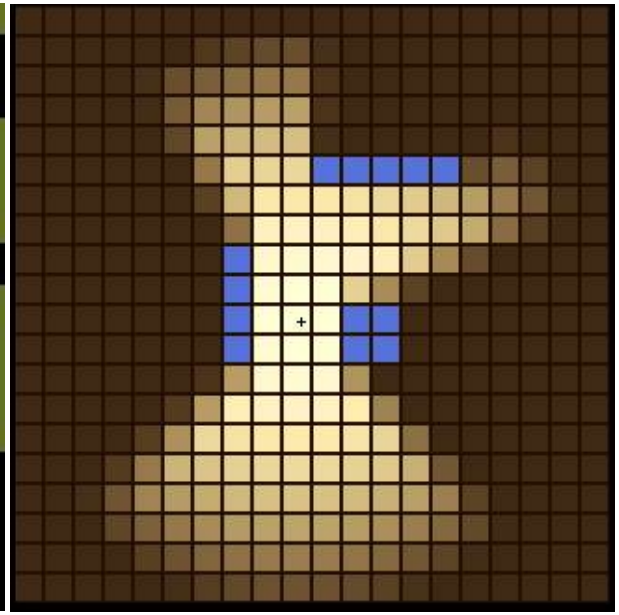
예습자료 주요 내용 & 첨부 코드 복습: AngleRange 클래스

- 길이 ~N인 두 range의 리스트 간 교집합이 있는지 (intersect) 비교

- 길이 ~N인 두 range의 리스트를 병합(merge)

- 위 두 작업이 ~N에 완료하기 위한 조건: range의 리스트가 정렬된 상태를 유지
  - 처음 생성할 때 & 병합할 때 계속해서 정렬된 상태 유지함

문제 정의: 지도에서 장애물에 가려 볼 수 없는 격자를 음영 처리



장애물을 포함한 지도와 지도에서 사용자의 위치가 주어졌을 때 사용자가 볼 수 있는 영역과 볼 수 없는 영역을 구분하여 다른 색으로 표기하는 효율적인 방법을 생각해보자. 특히 입력과 출력은 다음 페이지와 같이 주어진다.

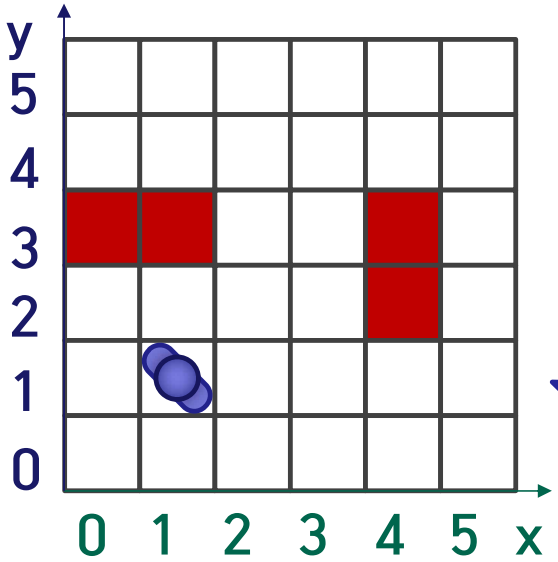
문제 정의: 지도에서 장애물에 가려 볼 수 없는 격자를 음영 처리

<입력>

① **N × N 지도**: N × N개의 **정사각형 격자**를 가진다. 지도의 각 격자는 (x, y)로 표기하며 (x는 가로좌표, y는 세로좌표), 왼쪽 아래가 (0, 0), 오른쪽 위가 (N-1, N-1) 이다.

② **지도 각 격자에 장애물 존재 여부**: 각 격자는 장애물이거나 장애물이 아니다.

③ **사용자가 위치한 격자 (xPlayer, yPlayer)**



N=6
즉 6x6 지도 예

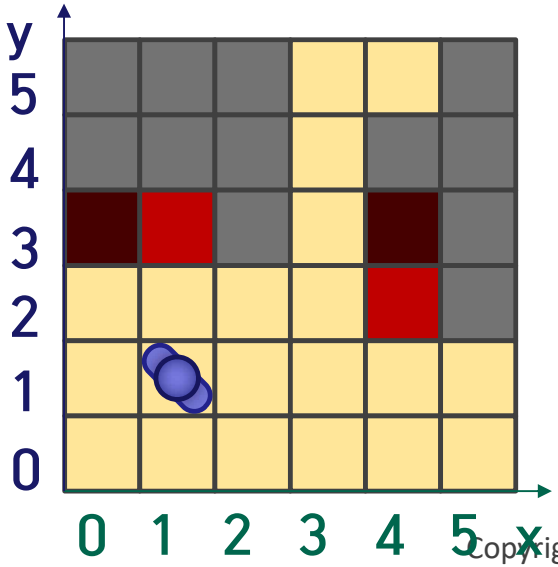
사용자는 장애물이 없는 곳에 있음

<출력>

① 장애물의 위치를 고려할 때 사용자가 볼 수 있는 격자와 볼 수 없는 격자를 구분해 출력

② 사용자가 360° 전 방향을 돌아본다고 가정하고 볼 수 있는 곳과 볼 수 없는 곳 출력

③ 장애물에 의해 **조금이라도 가려지는 격자는 볼 수 없는 격자로** 처리. 반대로 사각이 전혀 없는 격자만 볼 수 있는 격자로 처리



위 입력에 대한 결과

문제 풀이 과정

① **이번 시간 문제를 잘 이해**하기 위해 간단한 경우부터 시작해 문제 이해하고 답 이끌어내 보기

② 이번 시간 문제에 대한 **풀이 방법 생각**

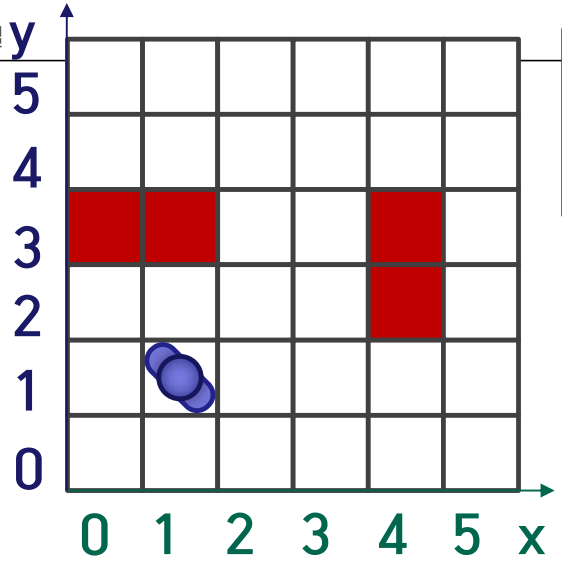
③ 생각한 풀이 방법을 보다 **효율적으로 개선**

④ 제안한 방법을 코드로 구현하는 방법 이해

문제 명확히 이해하는데 도움

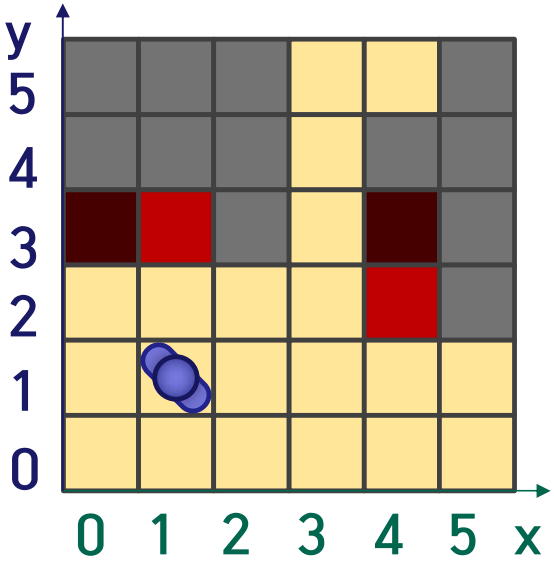
이 과정에서 문제에 대한 해법도 생각해 낼 수 있음

N=6
즉 6x6 지도 예

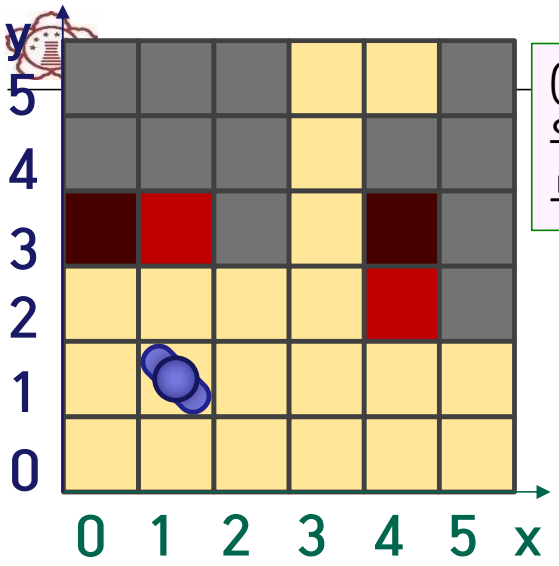


(Q) 이번 시간 문제의 입력에 대해 이해해 보자.
왼쪽 입력에서 사용자의 위치를 (x, y)로 나타내면 무엇인가? 또한 장애물 각각의 위치는 무엇인가?

위 입력에 대한 결과

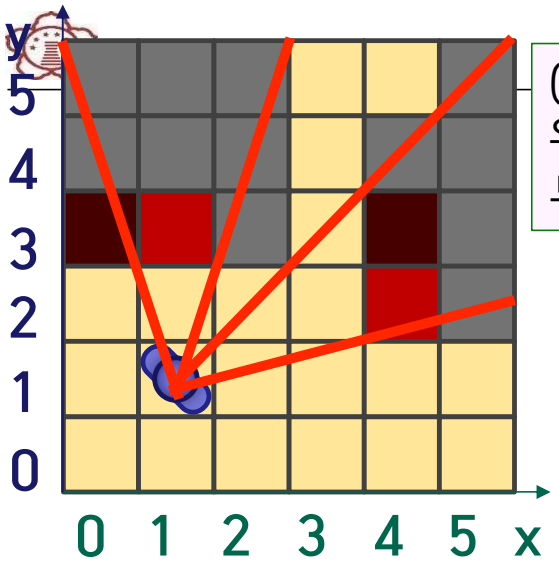


(Q) 이번 시간 문제의 출력에 대해 이해해 보자.
왼쪽 출력에서 사용자의 위, 아래, 왼쪽, 오른쪽에 모두 볼 수 있는 격자가 존재하는 이유는 무엇인가?



(Q) 이번에는 볼 수 없는 것으로 표기된 격자에 대해 생각해보자.
왼쪽 출력에서 격자 (2, 3), (2, 4), (2, 5)가 볼 수 없는 곳으로 표기된 이유는 무엇인가? 이들은 모두 볼 수 있는 곳과의 경계에 있는 격자들이다.

(Q) 또한 격자 (4,4), (5,5)가 볼 수 없는 곳으로 표기된 이유는 무엇인가?



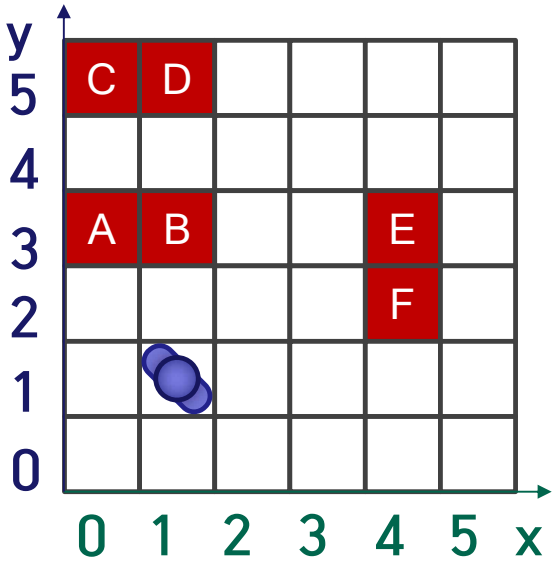
(Q) 이번에는 볼 수 없는 것으로 표기된 격자에 대해 생각해보자.
왼쪽 출력에서 격자 (2, 3), (2, 4), (2, 5)가 볼 수 없는 곳으로 표기된 이유는 무엇인가? 이들은 모두 볼 수 있는 곳과의 경계에 있는 격자들이다.

(Q) 또한 격자 (4,4), (5,5)가 볼 수 없는 곳으로 표기된 이유는 무엇인가?

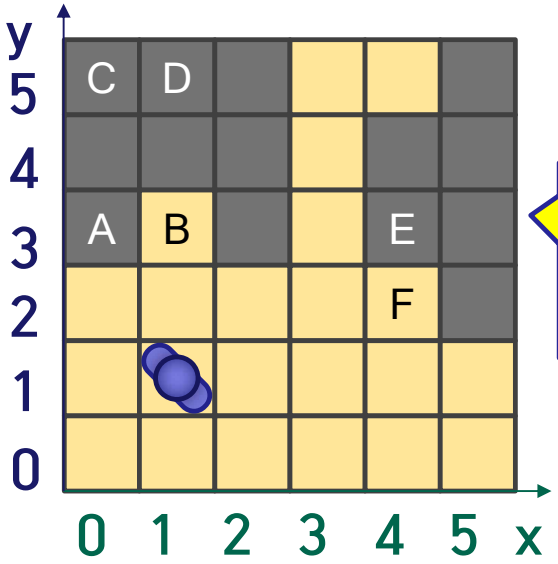
(0,5)의 경우

텍스트

입력 지도 예



입력에 대한 결과



<u>장애물 A~F도 음영 처리</u>함에 유의하세요.
즉, 다른 장애물에 의해 가려지는지 혹은 그렇지 않은지에 따라 1 or 0으로 표기

사용자 위치는 장애물이 없으며, 장애물에 의해 가려지지도 않습니다.

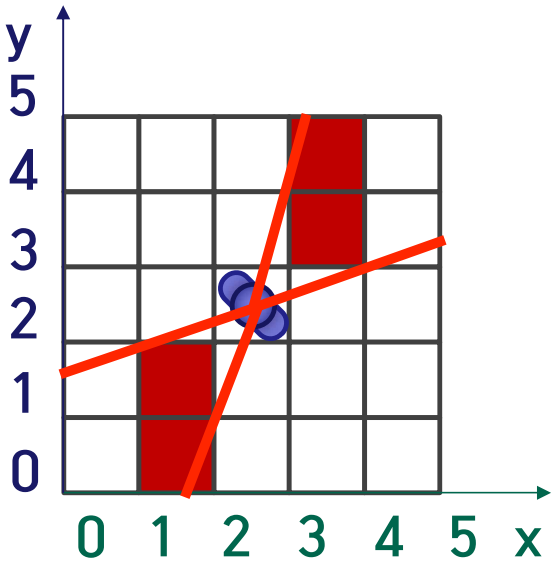
(Q) 격자 (0, 5), (1, 5)가 볼 수 없는 곳으로 표기된 이유는 무엇인가??

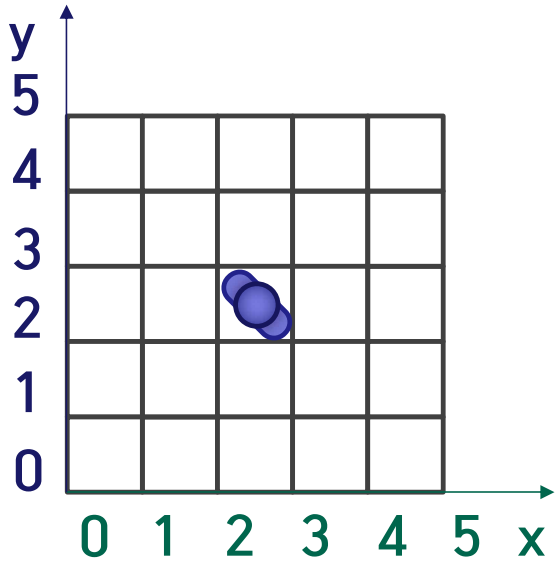
(0,5)의 경우: A와 B

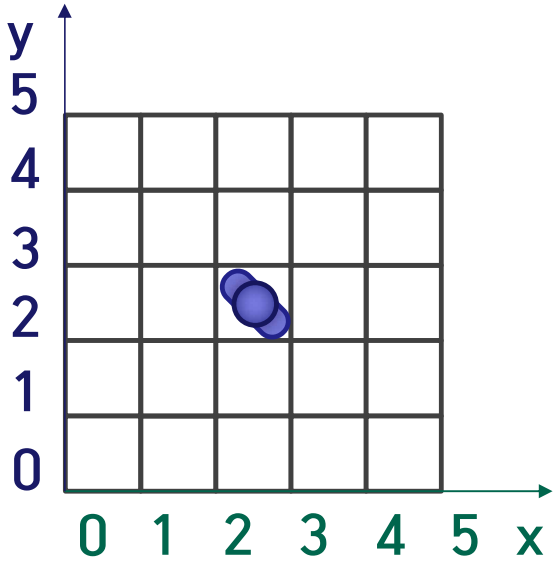
(1,5)의 경우: B

(Q) 또한 격자 (0, 3), (4, 3)이 볼 수 없는 곳으로 표기된 이유는 무엇인가??

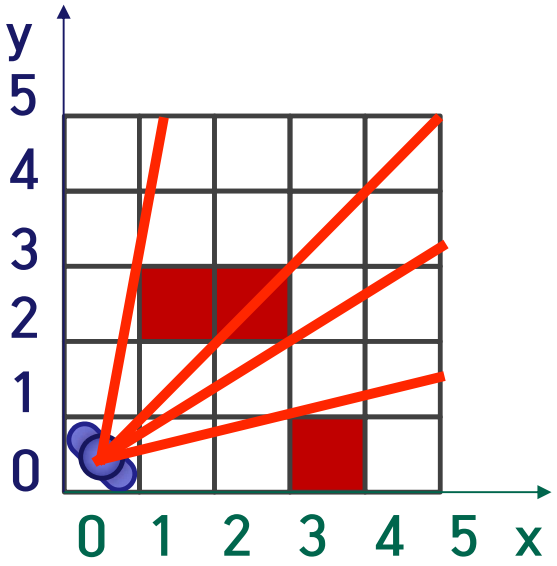
(Q) 아래 왼쪽과 같은 입력이 주어졌을 때
지금까지 본 문제의 조건에 따라 음영처리한 결과를 오른쪽에 표시해 보시오.

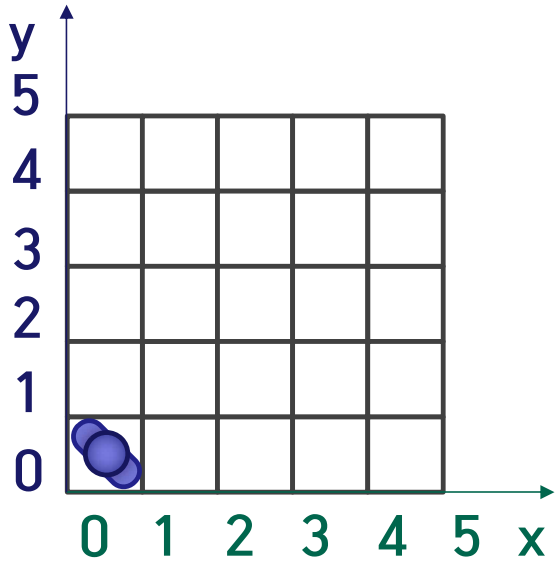


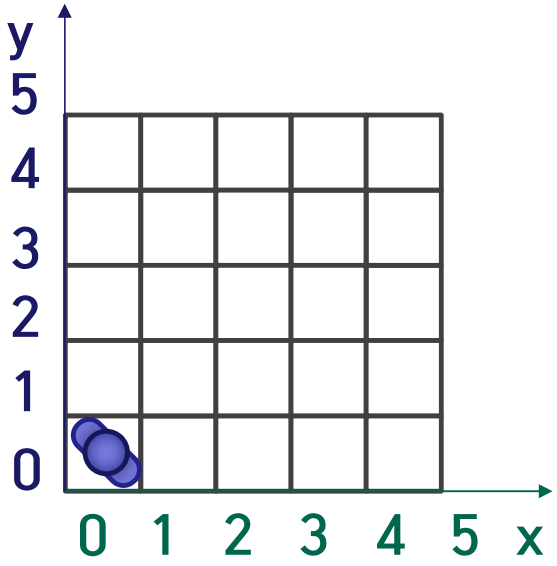




(Q) 아래 왼쪽과 같은 입력이 주어졌을 때
지금까지 본 문제의 조건에 따라 음영처리한 결과를 오른쪽에 표시해 보시오.







첫 번째 방법 (완전 탐색, brute-force)

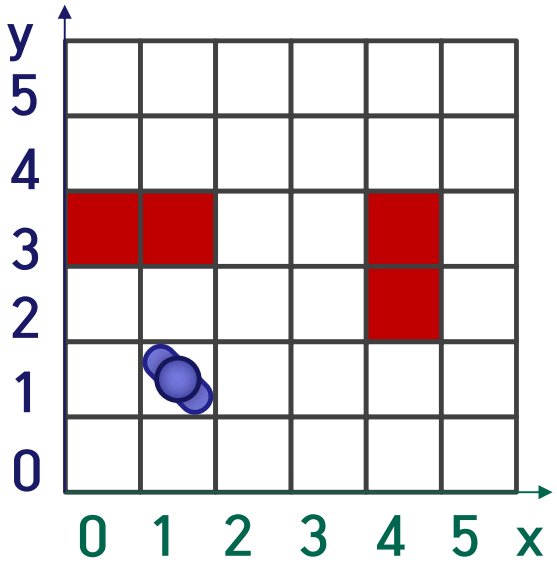
① **for** each cell c1(x1, y1) in the N × N map, such that c1 != cplayer

② **for** each cell c2(x2, y2) in the N×N map, such that c2 != cplayer **and** c2 != c1

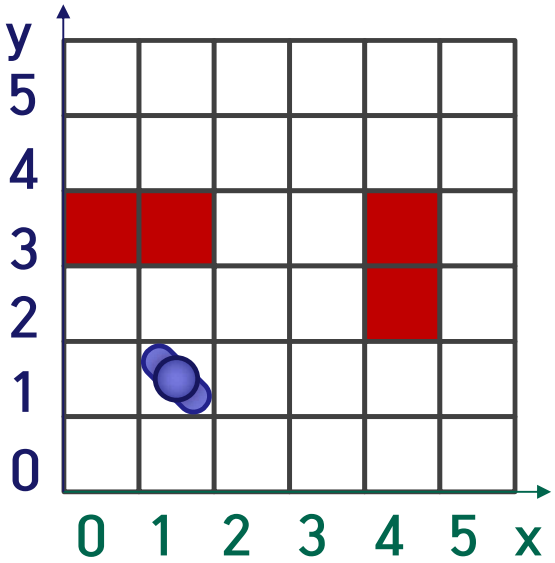
if (c2에 장애물 있음 **and** c 2가 c1과 cplayer 사이에 있음) **then** c1을 음영 처리

c: cell을 나타냄

(Q) 위 방법이 어떻게 동작하는지 왼쪽 지도의 예를 들어 설명해 보시오.



① **for** each cell c1(x1, y1) in the N × N map, such that c1 != cplayer
② **for** each cell c2(x2, y2) in the N×N map, such that c2 != cplayer **and** c2 != c1
if (c2에 장애물 있음 **and** c2가 c1과 cplayer 사이에 있음) **then** c1을 음영 처리



(Q) 위 방법에서 c1과 c2가 정해졌으며, c2에 장애물이 있다고 가정하자. If 문의 2번째 조건 ' c2가 c1와 cplayer 사이에 있음 ' 은 어떻게 확인해야 할까?

① for each cell c1(x1, y1) in the N × N map, such that c1 != cplayer
② for each cell c2(x2, y2) in the N×N map, such that c2 != cplayer and c2 != c1
if (c2에 장애물 있음 and c2가 c1과 cplayer 사이에 있음) then c1을 음영 처리

Shade.py 내 Board class의 멤버 함수 createShade1()

```
def createShade1(self):
  def distanceSquare(x1, y1, x2, y2):
    return (x1 - x2)**2 + (y1 - y2)**2

  for x1 in range(len(self.board)):
    for y1 in range(len(self.board)):
      if x1 == self.xPlayer and y1 == self.yPlayer: continue
      ag1 = self.angleRangeToCell(x1, y1)
      for x2 in range(len(self.board)):
        for y2 in range(len(self.board)):
          if x2 == x1 and y2 == y1: continue
          if self.board[y2][x2] == 0: continue
          if distanceSquare(self.xPlayer, self.yPlayer, x2, y2) >
                distanceSquare(self.xPlayer, self.yPlayer, x1, y1): continue
          ag2 = self.angleRangeToCell(x2, y2)
          if ag1.intersectWith(ag2):
            self.shade[y1][x1] = 1
            break
```

self.board: 지도 정보 저장하는 2차원 배열로 board[y][x]가 0이면 (x, y)에 장애물이 없고, 1이면 장애물이 있음을 나타냄

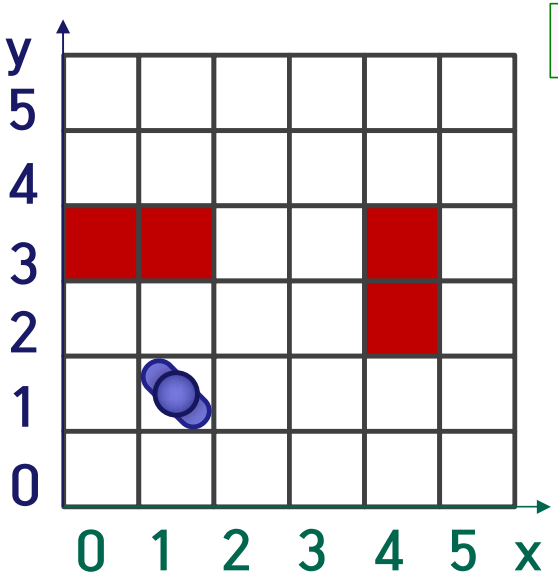
self.xPlayer, self.yPlayer: 사용자의 x, y 좌표

self.shade: 음영 정보 저장하는 2차원 배열로 shade[y][x]가 0이면 장애물에 가려지지 않음을, 1이면 가려짐을 나타냄. 처음에는 0으로 초기화

① **for** each cell c1(x1, y1) in the N × N map, such that c1 != cplayer

② **for** each cell c2(x2, y2) in the N×N map, such that c2 != cplayer **and** c2 != c1

if (c2에 장애물 있음 **and** c 2가 c1과 cplayer 사이에 있음) **then** c1을 음영 처리



(Q) 위 방법의 실행 시간은 어떤 값에 비례하는지를 입력 N을 사용해 나타내시오.

(Q) 위 방법을 N = 10, 20, …, 등에 대해 실행해 시간을 측정해 보고 앞에서 예측한 결과에 부합하는지 확인해 보시오.

① **for** each cell c1(x1, y1) in the N × N map, such that c1 != cplayer

② **for** each cell c2(x2, y2) in the N×N map, such that c2 != cplayer **and** c2 != c1

if (c2에 장애물 있음 **and** c 2가 c1과 cplayer 사이에 있음) **then** c1을 음영 처리

(Q) 위 방법의 실행 속도를 개선하는 방법을 제안하시오.

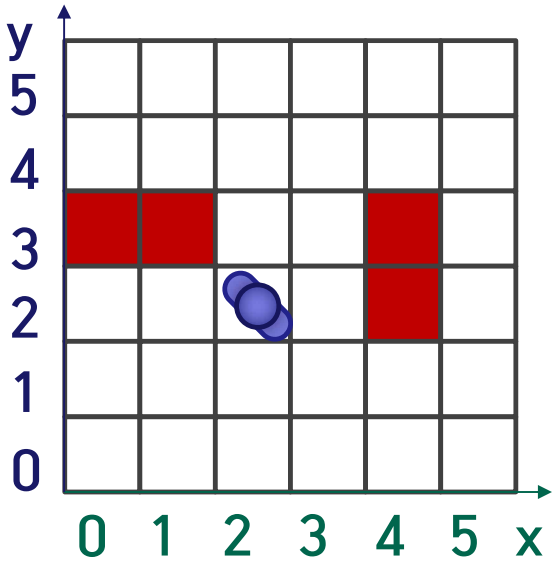
두 번째 방법 (c2 선정 범위 축소)

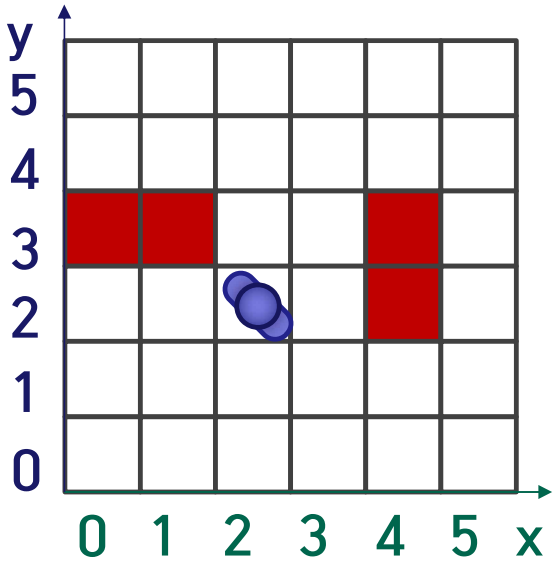
① for each cell c1(x1, y1) in the N × N map, such that c1 != cplayer

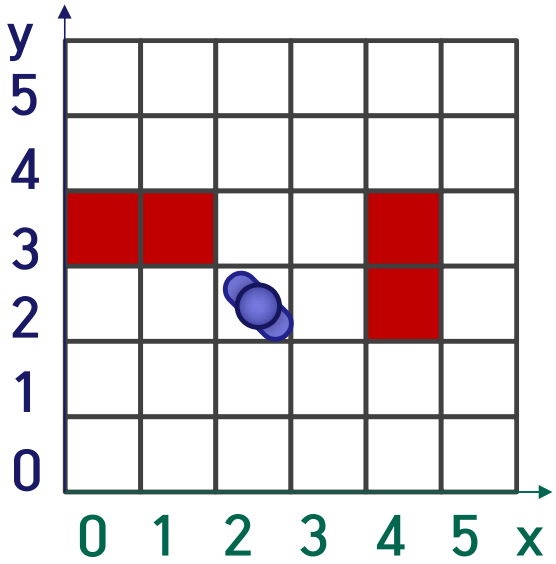
② for each cell c2(x2, y2) in the N×N map, such that c2가 c1 ~ cplyayer 사이

if (c2에 장애물 있음 and c2가 c1 가림) then c1을 음영 처리

(Q) 위 방법이 어떻게 동작하는지 지도의 예를 들어 설명해 보시오.

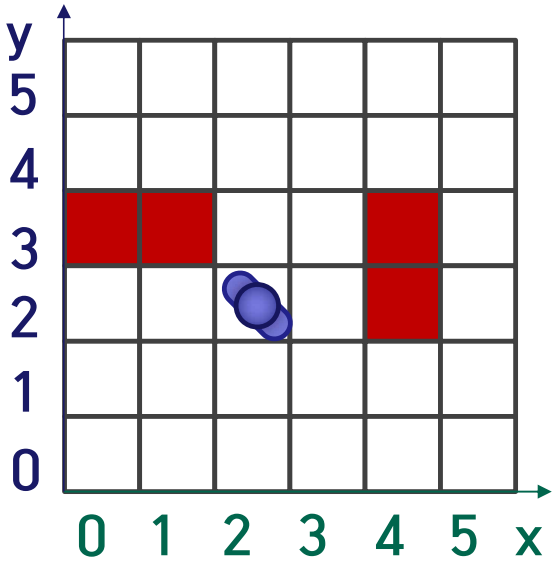


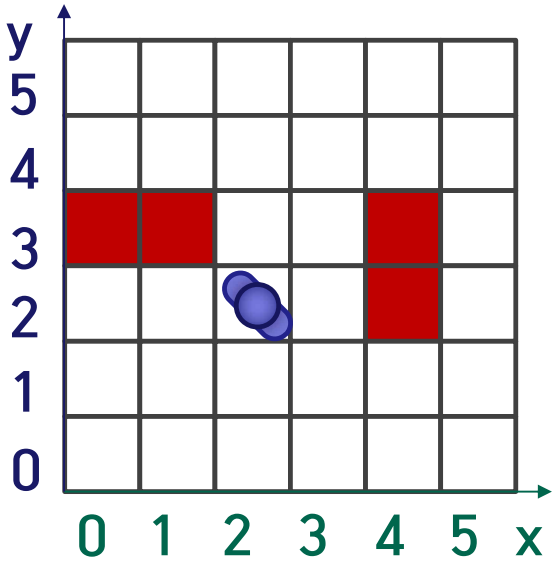


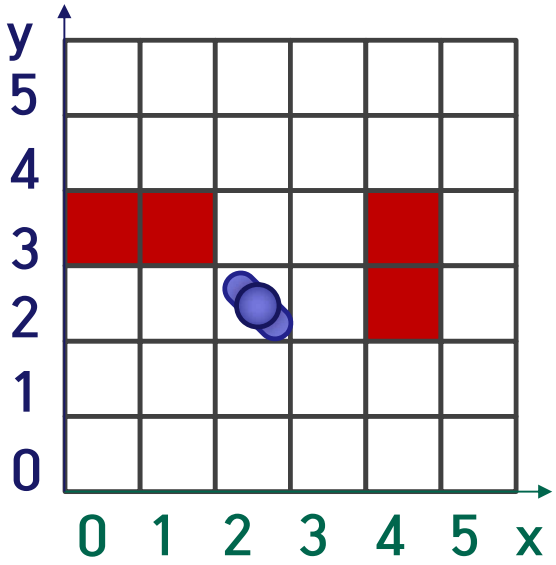


① for each cell c1(x1, y1) in the N × N map, such that c1 != cplayer
② for each cell c2(x2, y2) in the N×N map, such that c2가 c1 ~ cplyayer 사이
if (c2에 장애물 있음 and c2가 c1 가림) then c1을 음영 처리

(Q) 위 방법에서 c1이 정해졌을 때, c1과 cplayer 사이의 c2를 (복잡한 계산 없이) 어떻게 선정해야 할까?







① for each cell c1(x1, y1) in the N × N map, such that c1 != cplayer
② for each cell c2(x2, y2) in the N×N map, such that c2가 c1 ~ cplyayer 사이
if (c2에 장애물 있음 and c2가 c1 가림) then c1을 음영 처리

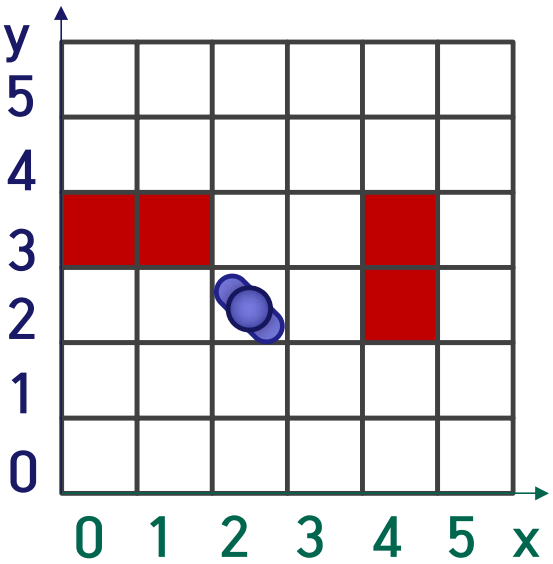
```
def createShade2(self):
        def minMax(v1, v2):
            if v1 <= v2: return v1, v2
            else: return v2, v1

        for x1 in range(len(self.board)):
            for y1 in range(len(self.board)):
                if x1 == self.xPlayer and y1 == self.yPlayer: continue
                ag1 = self.angleRangeToCell(x1, y1)
                xMin, xMax = minMax(x1, self.xPlayer)
                yMin, yMax = minMax(y1, self.yPlayer)
                for x2 in range(xMin, xMax+1):
                    for y2 in range(yMin, yMax+1):
                        if x2 == x1 and y2 == y1: continue
                        if self.board[y2][x2] == 0: continue # (x2, y2) is not an obstacle
                        ag2 = self.angleRangeToCell(x2, y2)
                        if ag1.intersectWith(ag2):
                            self.shade[y1][x1] = 1
                            break
```

Shade.py 내 Board class의 멤버 함수 createShade2()

① for each cell c1(x1, y1) in the N × N map, such that c1 != cplayer
② for each cell c2(x2, y2) in the N×N map, such that c2가 c1 ~ cplyayer 사이
if (c2에 장애물 있음 and c2가 c1 가림) then c1을 음영 처리

(Q) 위 방법의 실행 시간은 어떤 값에 비례하는지를 입력 N을 사용해 나타내 보시오.

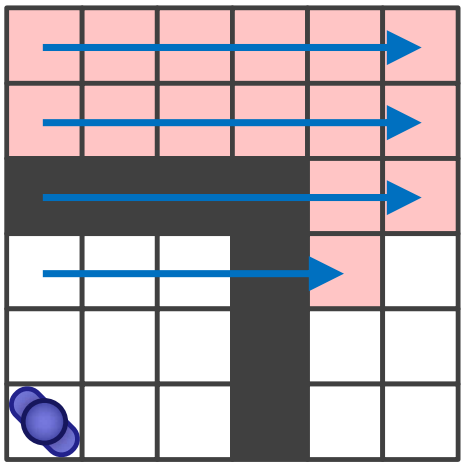


p가 끝쪽에 붙어있으면 확인해야할 디스턴스가 늘어나 워스트
p가 가운데이면 거리가 줄어들어 베스트

(Q) 위 방법의 실행 시간을 측정해 보고 예측한 결과에 부합하는지 확인해 보시오.

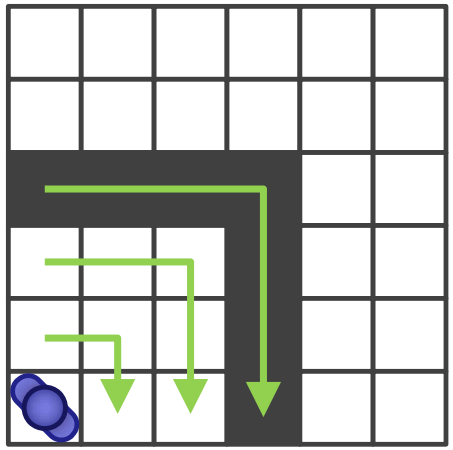
① **for** each cell c1(x1, y1) in the N × N map, such that c1 != cplayer
② **for** each cell c2(x2, y2) in the N×N map, such that **c2가 c1 ~ cplyayer 사이**
if (c2에 장애물 있음 and c2가 c1 가림) **then** c1을 음영 처리

(Q) 위 방법의 실행 속도는 **최대 어디까지 개선 가능**할까? 어떻게 개선해야 할까?

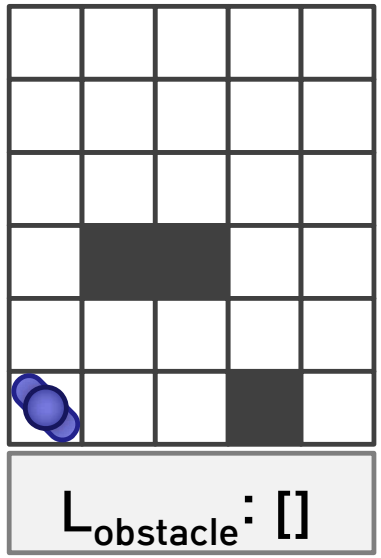


세 번째 방법: 사용자로부터 가까운 격자가 먼 격자를 가림

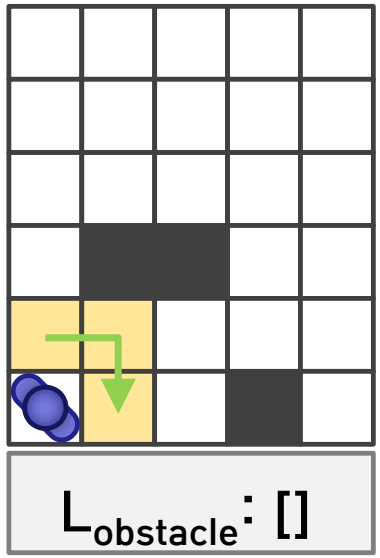
Lobstacle: player가 볼 수 없는 각도 범위의 리스트
① for distance = 1 to (지도상에서 player로부터 가장 먼 경계선까지의 거리)
② for each cell c1(x1, y1) in (player로부터 distance만큼 떨어진 정사각형)
③ if (cplayer → c1 까지 시야가 Lobstacle에 의해 가린다면) then c1을 음영처리
④ if (c1에 장애물 있음) then c1이 cplayer의 시야를 가리는 각도를 Lobstacle에 추가



<방법 3 개요>

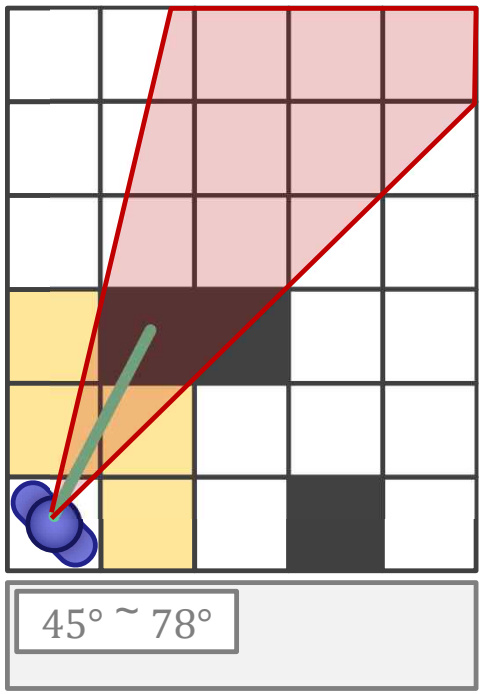


<방법 3 적용 전>

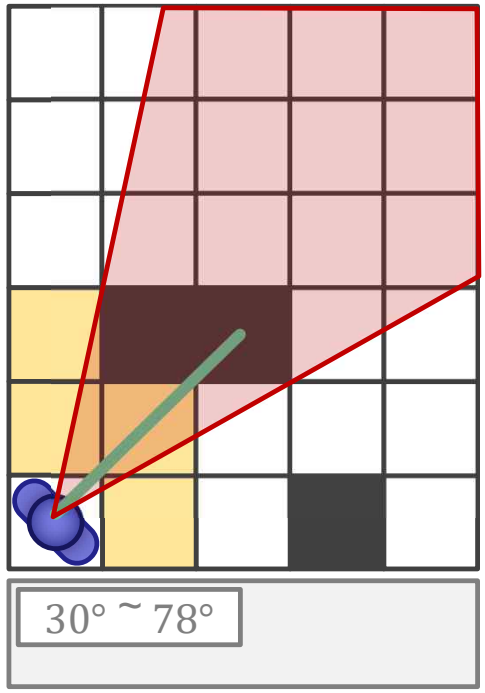


<distance = 1>

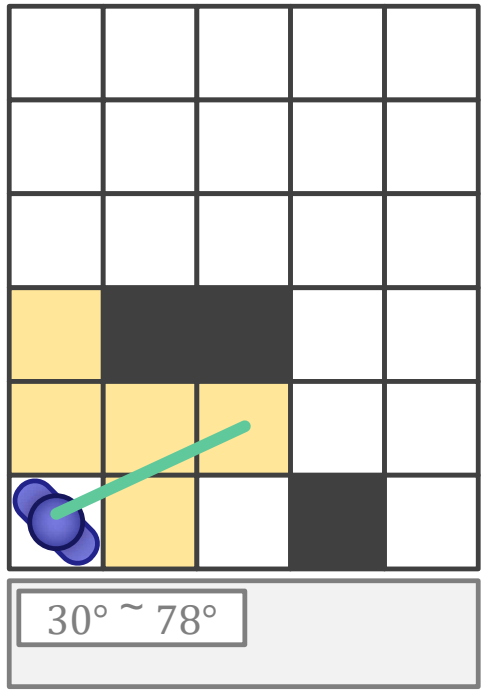
Lobstacle: player가 볼 수 없는 각도 범위의 리스트
① for distance = 1 to (지도상에서 player로부터 가장 먼 경계선까지의 거리)
② for each cell c1(x1, y1) in (player로부터 distance만큼 떨어진 정사각형)
③ if (cplayer → c1 까지 시야가 Lobstacle에 의해 가린다면) then c1을 음영처리
④ if (c1에 장애물 있음) then c1이 cplayer의 시야를 가리는 각도를 Lobstacle에 추가



<d = 2, 2번째 격자>

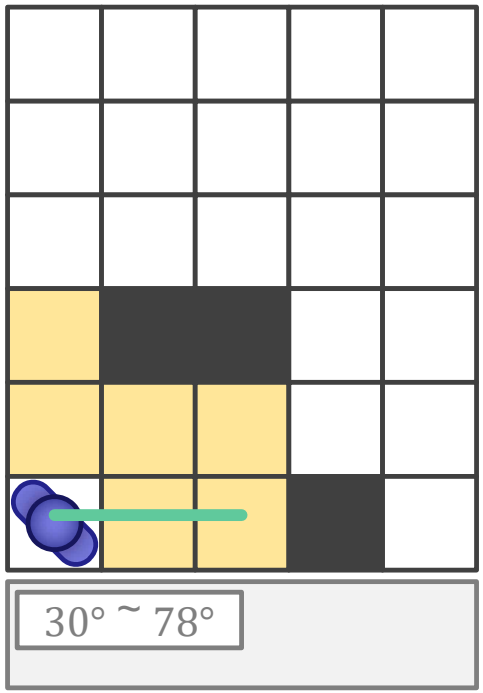


<d = 2, 3번째 격자>

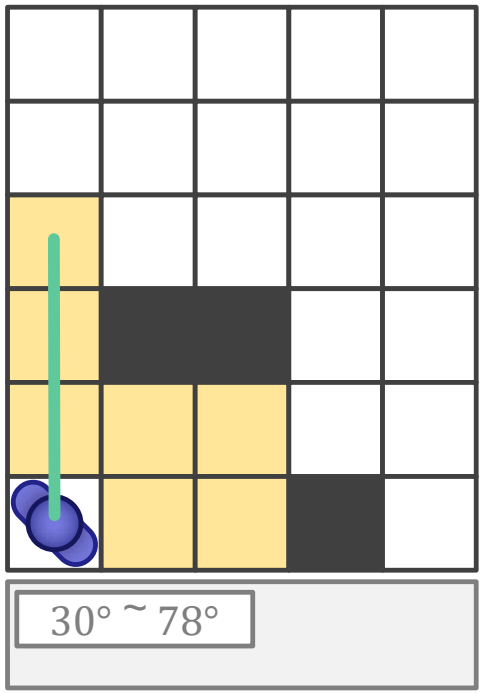


<d = 2, 4번째 격자>

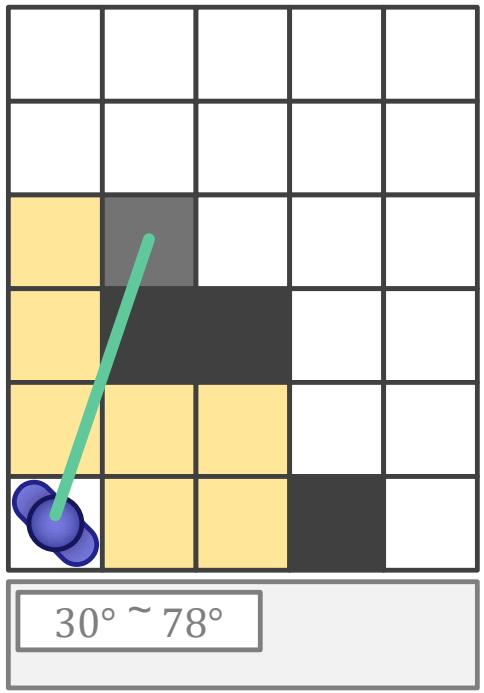
Lobstacle: player가 볼 수 없는 각도 범위의 리스트
① for distance = 1 to (지도상에서 player로부터 가장 먼 경계선까지의 거리)
② for each cell c1(x1, y1) in (player로부터 distance만큼 떨어진 정사각형)
③ if (cplayer → c1 까지 시야가 Lobstacle에 의해 가린다면) then c1을 음영처리
④ if (c1에 장애물 있음) then c1이 cplayer의 시야를 가리는 각도를 Lobstacle에 추가



<d = 2, 5번째 격자>

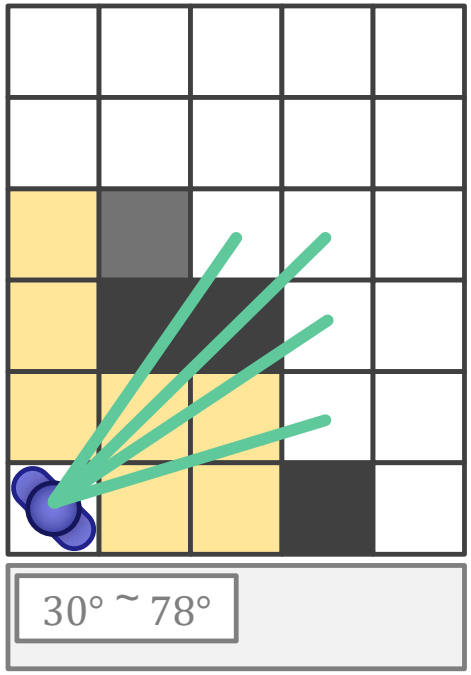


<d = 3, 1번째 격자>

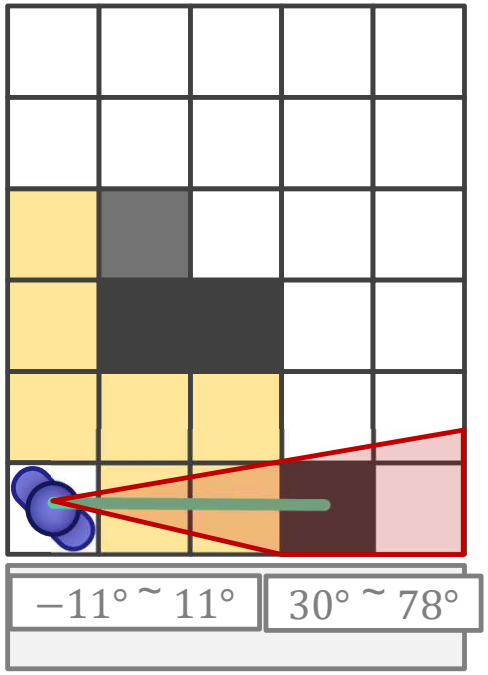


<d = 3, 2번째 격자>

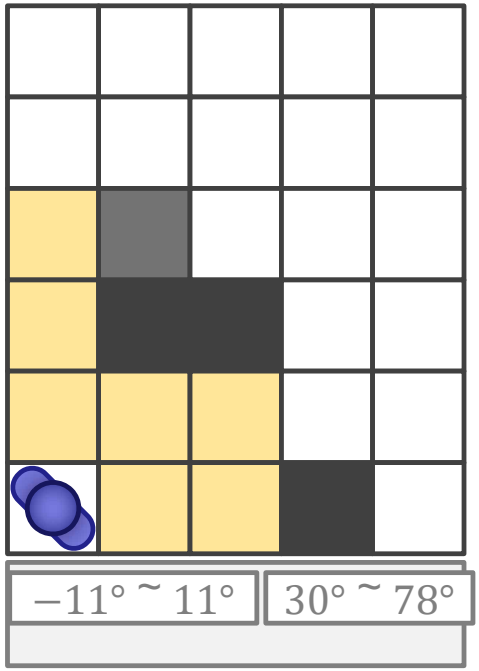
Lobstacle: player가 볼 수 없는 각도 범위의 리스트
① for distance = 1 to (지도상에서 player로부터 가장 먼 경계선까지의 거리)
② for each cell c1(x1, y1) in (player로부터 distance만큼 떨어진 정사각형)
③ if (cplayer → c1 까지 시야가 Lobstacle에 의해 가린다면) then c1을 음영처리
④ if (c1에 장애물 있음) then c1이 cplayer의 시야를 가리는 각도를 Lobstacle에 추가



<d = 3, 3 ~ 6번째 격자>



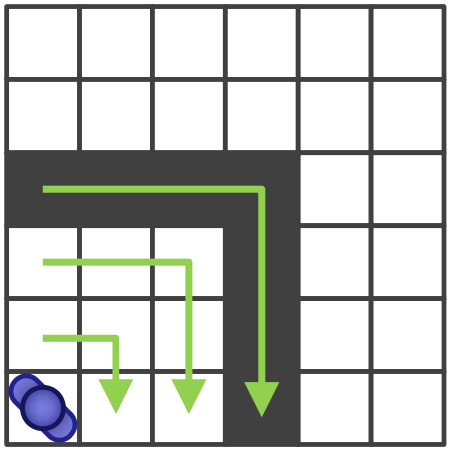
<d = 3, 7번째 격자>



(Q) Lobstacle 목록 참조해
d = 4 ~ 5 격자에 대해 음영처리해 보자.

Lobstacle: player가 볼 수 없는 각도 범위의 리스트
① for **distance = 1 to (지도상에서 player로부터 가장 먼 경계선까지의 거리)**
② for each cell c1(x1, y1) in (player로부터 distance만큼 떨어진 정사각형)
③ if (cplayer → c1 까지 시야가 Lobstacle에 의해 가린다면) then c1을 음영처리
④ if (c1에 장애물 있음) then c1이 cplayer의 시야를 가리는 각도를 Lobstacle에 추가

(Q) 위 방법에서 사용자로부터 **가까운 격자부터 검사를 시작해 점차 먼 격자를 검사**해가는 이유는 무엇인가? 만약 먼 격자부터 검사하도록 검사 순서를 변경한다면 어떤 문제가 발생하는가?



기하 활용 문제 해결 2

기하 관련 지식 활용해 SW 문제 풀이하는 방법 익히기

01. 문제에서 사용하는 정의 이해: 정수 좌표 가지는 직각삼각형

02. 첫 번째 알고리즘

03. 두 번째 알고리즘

04. 세 번째 알고리즘

05. 지금까지 본 알고리즘의 구현 방법 이해

06. 실습 문제 풀이 & 질의 응답

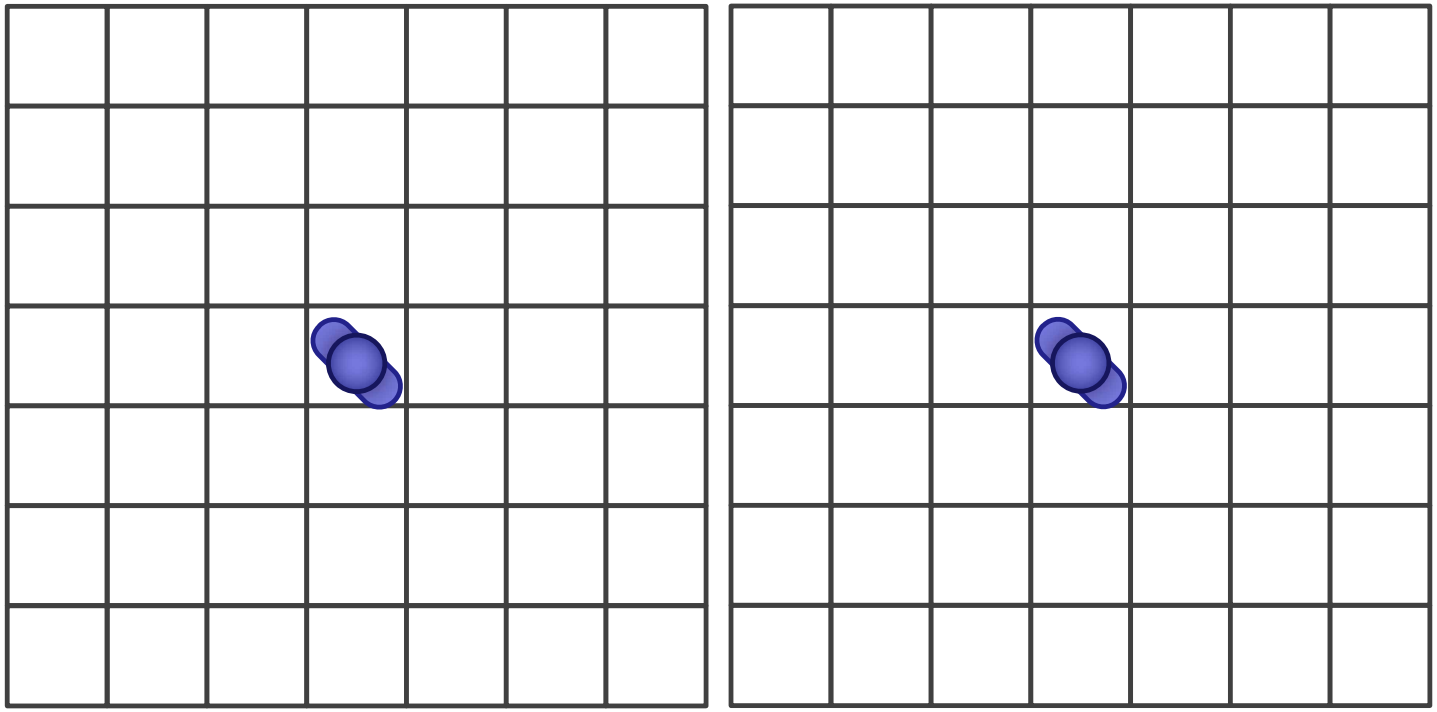
최종 방법의 구현방법 detail 이해 +
실습문제 풀이에 제공하는 코드 이해

<왜 기하/수학의 활용 예를 보는가? >

- **SW/HW 시스템 설계** 시 **수학 활용**하여 문제 해결되는 경우 많음
- 적절한 수학 활용 (기하 포함) → (i) **좋은 성능의 시스템** 설계 + (ii) **설계 시간 단축**
- 기하 관련 지식은 컴퓨터 그래픽 분야 및 수학 활용 SW 개발에 많이 활용됨 (예: Octave, Matlab)

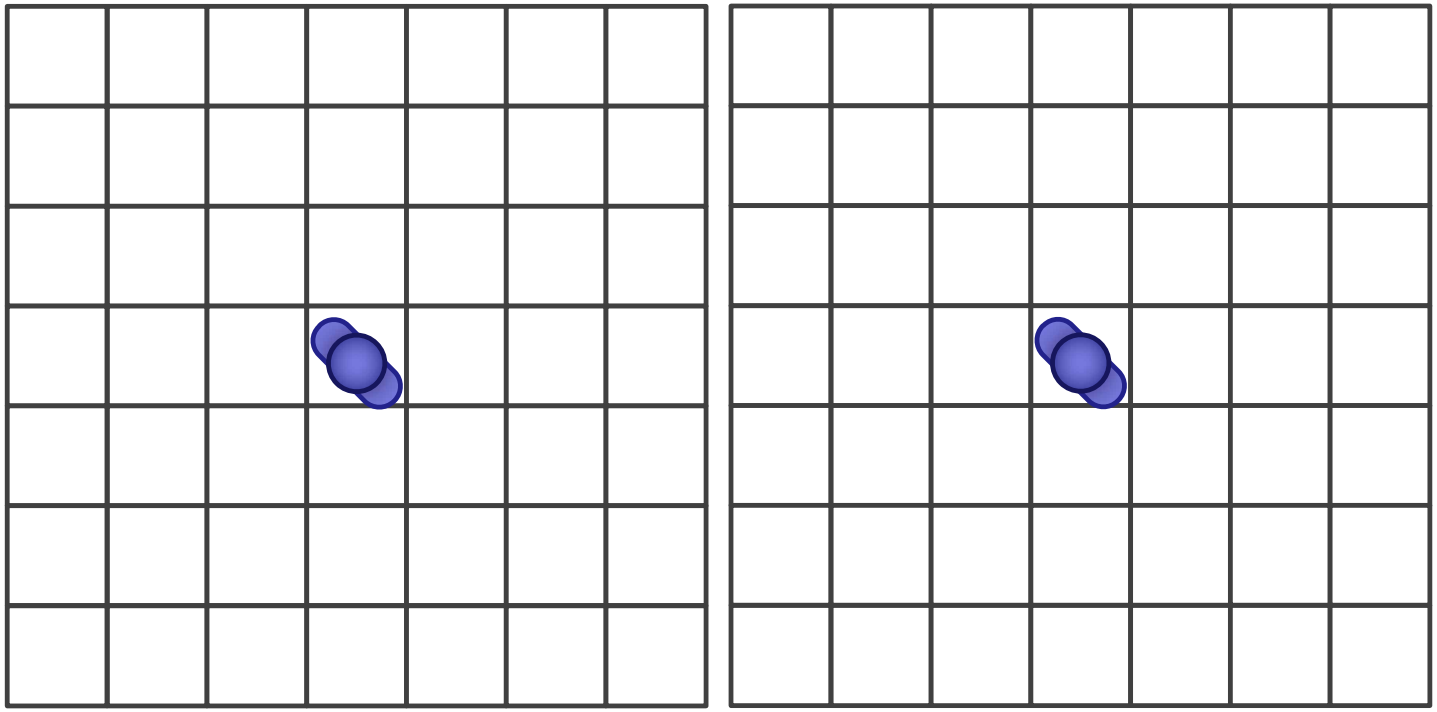
Lobstacle: player가 볼 수 없는 각도 범위의 리스트
① for distance = 1 to (지도상에서 player로부터 가장 먼 경계선까지의 거리)
② for each cell c1(x1, y1) in (player로부터 distance만큼 떨어진 정사각형)
③ if (cplayer → c1 까지 시야가 Lobstacle에 의해 가린다면) then c1을 음영처리
④ if (c1에 장애물 있음) then c1이 cplayer의 시야를 가리는 각도를 Lobstacle에 추가

(Q) 위 방법의 ④에 따르면 **장애물이 있는** 격자 c1을 발견하였으면 '**c1이 cplayer의 시야를 가리는 각도**를 Lobstacle에 추가'하여야 한다. 이처럼 가리는 각도의 범위를 어떻게 결정할 것인지 생각해보자. 이 범위는 player에 대한 c의 상대 위치에 따라 다르게 결정되어야 한다.



Lobstacle: player가 볼 수 없는 각도 범위의 리스트
① for distance = 1 to (지도상에서 player로부터 가장 먼 경계선까지의 거리)
② for each cell c1(x1, y1) in (player로부터 distance만큼 떨어진 정사각형)
③ if (**cplayer → c1 까지 시야**가 Lobstacle에 의해 가린다면) then c1을 음영처리
④ if (c1에 장애물 있음) then c1이 cplayer의 시야를 가리는 각도를 Lobstacle에 추가

(Q) 위 방법의 ③에 따르면 검사하는 각 격자 c1에 대해 '**cplayer → c1 까지 시야**가 Lobstacle에 의해 가린다면 c1을 음영 처리'해야 한다. cplayer → c1 까지 **c1 전체가 잘 보이는 시야**는 어떤 각도로 표현하는 것이 적절할까?



Player에서 격자 (x, y) 전체를 볼 수 있는 각도 범위 or 격자 (x, y)가 player의 시야를 가리는 각도 범위 구하는 코드

```
def angleRangeToCell(self, x, y):
  if y < self.yPlayer:
    if x < self.xPlayer: x1, y1, x2, y2 = x, y+1, x+1, y
    elif x == self.xPlayer: x1, y1, x2, y2 = x, y+1, x+1, y+1
    else: x1, y1, x2, y2 = x, y, x+1, y+1
  elif y == self.yPlayer:
    if x < self.xPlayer: x1, y1, x2, y2 = x+1, y, x+1, y+1
    elif self.xPlayer == x: pass
    else: x1, y1, x2, y2 =  x, y, x, y+1
  else:
    if x < self.xPlayer: x1, y1, x2, y2 = x, y, x+1, y+1
    elif self.xPlayer == x: x1, y1, x2, y2 = x, y, x+1, y
    else: x1, y1, x2, y2 = x, y+1, x+1, y

  return AngleRange(self.xPlayer + 0.5, self.yPlayer + 0.5, x1, y1, x2, y2)
```

Shade.py 내 Board class의 멤버 함수

Lobstacle: player가 볼 수 없는 각도 범위의 리스트

① for distance = 1 to (지도상에서 player로부터 가장 먼 경계선까지의 거리)

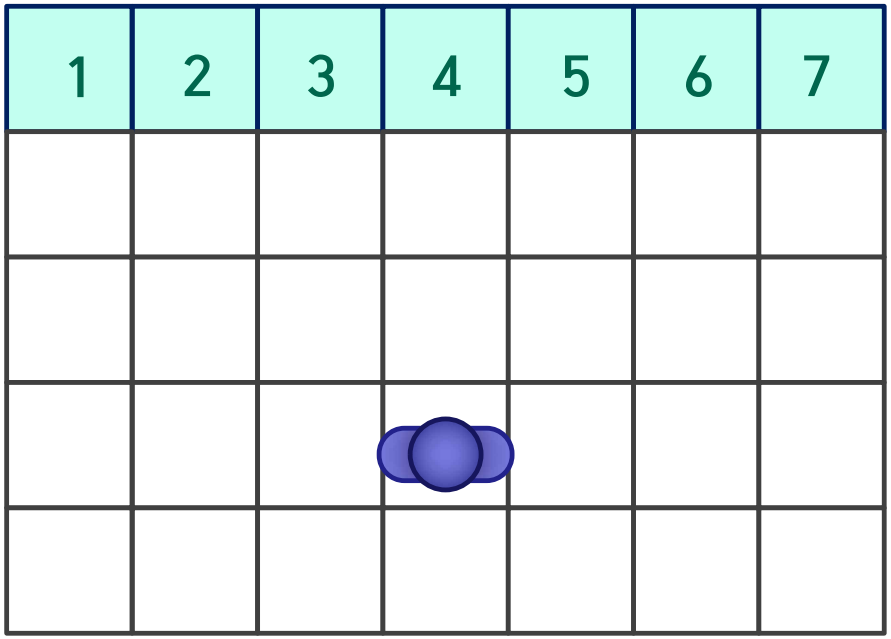
② **for each cell c1(x1, y1) in (player로부터 distance만큼 떨어진 정사각형)**

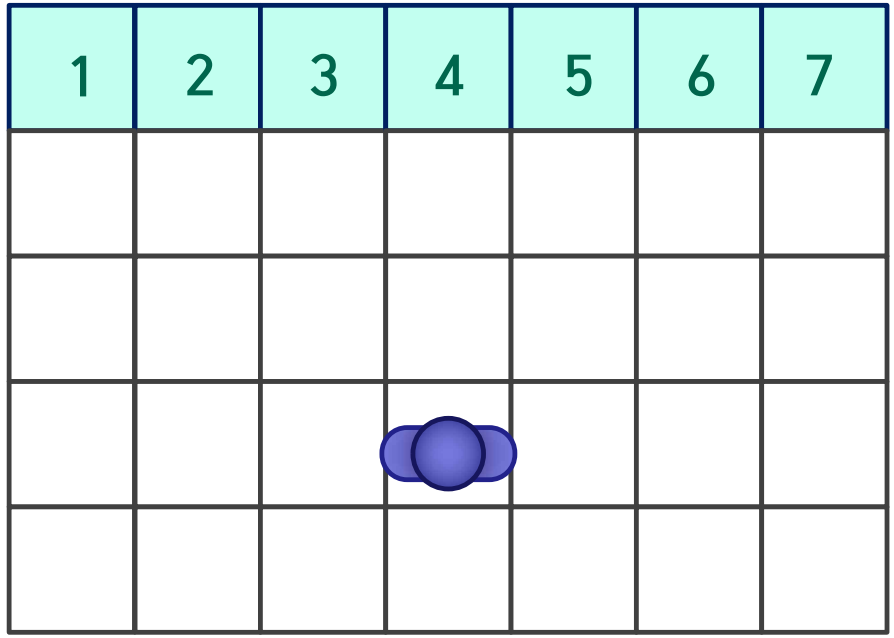
③ if (cplayer → c1 까지 시야가 Lobstacle에 의해 가린다면) then c1을 음영처리

④ if (c1에 장애물 있음) then c1이 cplayer의 시야를 가리는 각도를 Lobstacle에 추가

Distance 같은 격자 간 검사 순서 정하기

(Q) 앞에서 사용자로부터 가까운 격자에서 먼 격자 순으로 검사하는 이유를 보았다. 이는 distance 값이 같은 격자 간에도 마찬가지로 적용되어야 한다. distance=3인 사각형의 한 변에 있는 격자 1~7에 대해서 생각해보자 (다른 3개 변 각각에는 앞으로 볼 특성이 동일하게 적용된다). 이 중 장애물이 있는 경우 인접한 격자를 가리는 격자는 어느 것인가? 어느 격자가 어느 격자에 이러한 영향을 주는지 표시해 보자.





Lobstacle: player가 볼 수 없는 각도 범위의 리스트

① for distance = 1 to (지도상에서 player로부터 가장 먼 경계선까지의 거리)

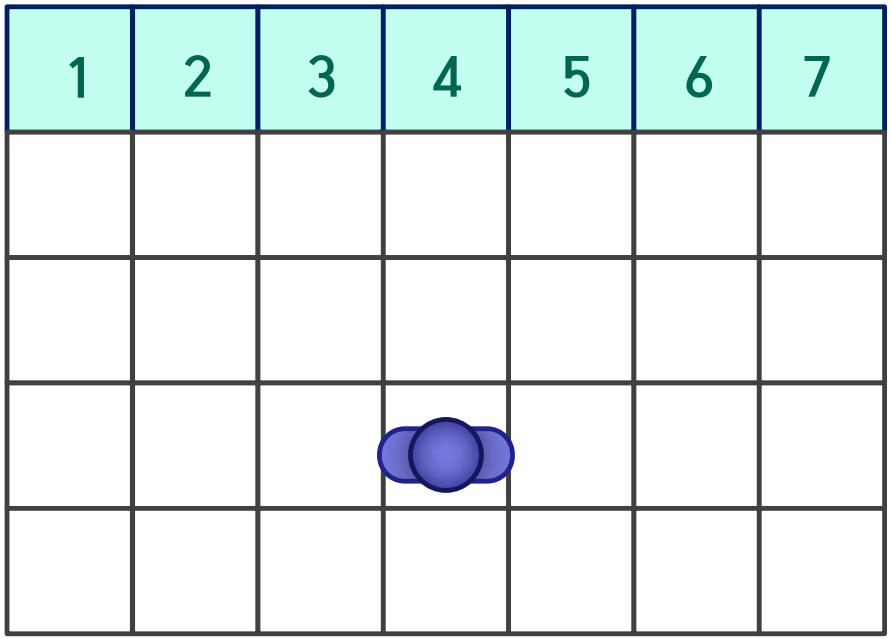
② **for each cell c1(x1, y1) in (player로부터 distance만큼 떨어진 정사각형)**

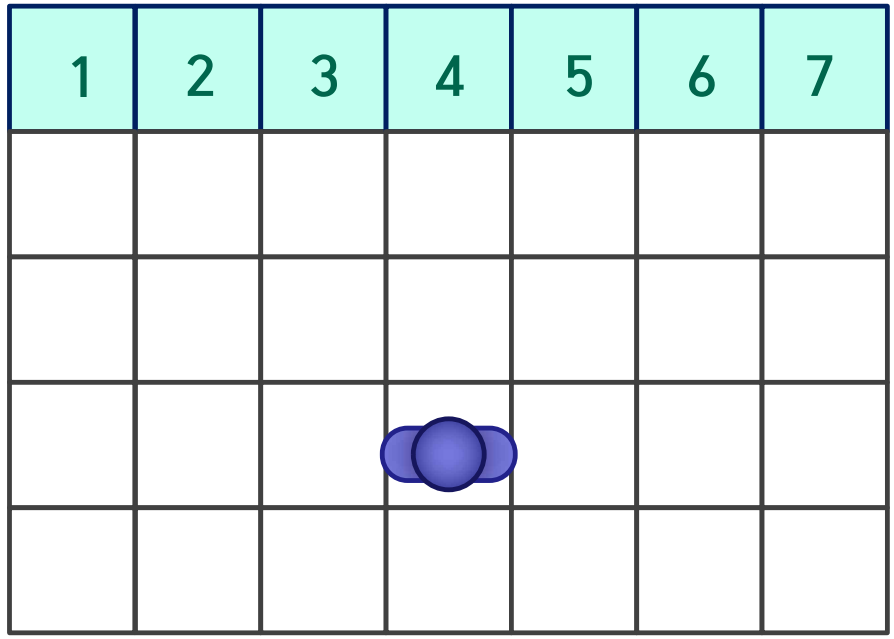
③ if (cplayer → c1 까지 시야가 Lobstacle에 의해 가린다면) then c1을 음영처리

④ if (c1에 장애물 있음) then c1이 cplayer의 시야를 가리는 각도를 Lobstacle에 추가

Distance 같은 격자 간 검사 순서 정하기

(Q) 앞 문제의 답을 고려해 볼 때, 같은 distance 상에 있는 격자들은 어떤 순서로 검사해야 조금이라도 가려지는 격자를 빠뜨리지 않고 정확하게 확인할 수 있을까?

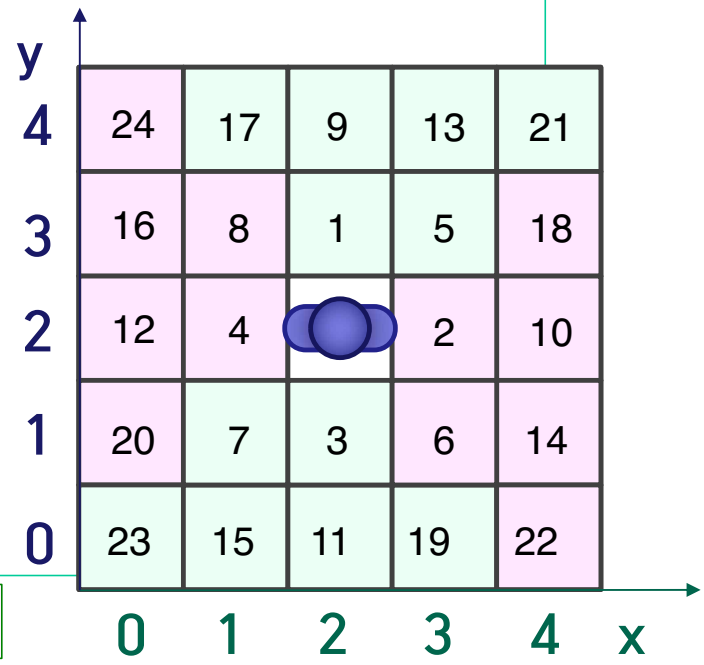




같은 거리의 격자를 검사하는 순서를 정하는 방법

Lobstacle: player가 볼 수 없는 각도 범위의 리스트
checkCell(x, y): 격자 (x, y)를 검사하며 아래를 수행하는 함수
③ if (player → (x, y) 까지 시야가 Lobstacle에 의해 가린다면) then (x, y)를 음영처리
④ if ((x, y)에 장애물 있음) then (x, y)가 player의 시야를 가리는 각도를 Lobstacle에 추가

```
for distance = 1 to (지도상에서 player로부터 가장 먼 경계선까지 거리):
   i = 0
   while i < distance:
     checkCell(xPlayer - i, yPlayer + distance) # up
     checkCell(xPlayer + distance, yPlayer + i) # right
     checkCell(xPlayer + i, yPlayer - distance) # down
     checkCell(xPlayer - distance, yPlayer - i) # left
     i++;
     checkCell(xPlayer + i, yPlayer + distance) # up
     checkCell(xPlayer + distance, yPlayer - i) # right
     checkCell(xPlayer - i, yPlayer - distance) # down
     checkCell(xPlayer - distance, yPlayer + i) # left
```



(Q) 위 방법에 따라 격자를 검사하는 순서대로 1부터 번호를 써 보시오.

실습문제 풀이에 제공되는 API 이해: **AngleRange 클래스** 멤버 변수와 함수


```
class AngleRange: # 각도의 range들을 저장하는 클래스
  # 멤버 변수
  self.ranges = []  # 각도 범위 저장하는 리스트. 각 범위는 2-tuple (rFrom, rTo)
                    # 예: [(0.1, 0.2), (0.3, 0.4)]는 0.1 ~ 0.2와 0.3 ~ 0.4의 두 범위 저장

  # 멤버 함수
  def __init__(self, x0, y0, x1, y1, x2, y2):  # 생성자 함수 (constructor)
    # (x0, y0)와 (x1, y1)를 잇는 직선과  (x0, y0)와 (x2, y2) 잇는 직선 간
    # 각도 범위 저장하는 객체 생성
    # 만약 x0 ~ y2가 None이라면

  def __str__(self): # 이 클래스 객체를 문자열 형식으로 반환하며, print()로 출력할 때 호출됨

  def intersectWith(self, ag):
    # 현재 객체가 나타내는 range들과 ag가 나타내는 range가 일부라도 겹치면 True를 반환하고
    # 전혀 겹치지 않으면 False 반환

  def mergeWith(self, ag):
    # 현재 객체가 나타내는 range들과 ag가 나타내는 range들을 병합하여
    # 그 결과를 현재 객체에 저장
```

실습문제 풀이에 제공되는 API 이해: AngleRange 클래스 활용 예


```
ag1 = AngleRange(None, None, None, None, None, None)  # 비어 있는 객체 생성
ag2 = AngleRange(0, 0, 1, 0, 0, 1)  # (0, 0)과 (1, 0), (0, 1)이 이루는 각도 범위 저장 객체
ag1.mergeWith(ag2) # ag1에 ag2의 범위가 병합됨
print(ag1)

ag3 = AngleRange(0, 0, 1, 0, -1, 0)  # (0, 0)과 (1, 0), (-1, 0)이 이루는 각도 범위 저장 객체
ag1.mergeWith(ag3) # ag1에 ag3의 범위가 병합됨
print(ag1)
```

```
<출력 결과>
(0.0, 1.5707963267948966)
# (0, 0)과 (1, 0), (0, 1)이 이루는 각도 범위는 0 ~ 90도이며
# 이를 라디안 단위로 표현하면 (0.0, 1.5707963267948966)

(0.0, 3.141592653589793)
# (0, 0)과 (1, 0), (0, 1)이 이루는 각도 범위는 0 ~ 90도이며
# (0, 0)과 (1, 0), (-1, 0)이 이루는 각도 범위는 0 ~ 180도 이므로
# 이들이 모두 병합되면 0 ~ 180도 하나의 범위가 됨
# 이를 라디안 단위로 표현하면 (0.0, 3.141592653589793)
```

실습문제 풀이에 제공되는 API 이해: Board 클래스 멤버 변수와 함수

Map은 Python에서 사용하는 키워드라 이름을 Board로 정함

```
class Board: # 지도를 저장하는 클래스
  # 멤버 변수
  self.board # 지도 정보를 저장하는 2차원 배열로
             # board[y][x]가 0이면 (x, y)에 장애물이 없고, 1이면 장애물이 있음을 나타냄

  self.xPlayer, self.yPlayer # 사용자의 x, y 좌표를 저장하는 변수

  self.shade # 음영 정보를 저장하는 2차원 배열로
             # shade[y][x]가 0이면 장애물에 가려지지 않음을, 1이면 가려짐을 나타냄
             # 객체가 처음 생성될 때는 모두 0으로 초기화

  # 멤버 함수
  def __init__(self, fileName):  # 생성자 함수 (constructor)
    # Board 객체를 생성하되, 특히 파일 fileName으로부터 지도 정보를 읽어 멤버 변수에 저장
    # 만약 fileName이 정수라면, N = fileName인 임의의 지도 객체를 생성

  def __str__(self): # 이 클래스 객체를 문자열 형식으로 반환하며, print()로 출력할 때 호출됨

  # 다음 페이지에 계속 이어짐
```

실습문제 풀이에 제공되는 API 이해: Board 클래스 사용 예


```
b1 = Board("map1.txt")   # map1.txt 파일의 정보 읽어 Board 객체 생성
print(b1)                # 생성한 객체 출력

b2 = Board(5)            # 가로 세로 길이 N = 5인 Board 객체를 랜덤하게 생성
print(b2)                # 생성한 객체 출력
```

<출력 결과>

```
00010
00010
00p00
01000
01000

00011
10010
0p000
01000
00001
```

map1.txt

```
00010
00010
00p00
01000
01000
```

실습문제 풀이에 제공되는 API 이해: **Board 클래스** 멤버 변수와 함수


```
class Board: # 지도를 저장하는 클래스
  # 멤버 함수: 앞 페이지에서 이어짐
  def createShade1(self): # 방법 1에 따라 음영을 결정해 self.shade에 저장하는 함수
  def createShade2(self): # 방법 2에 따라 음영을 결정해 self.shade에 저장하는 함수
  def createShade3(self): # 방법 3에 따라 음영을 결정해 self.shade에 저장하는 함수로
                          # 실습 과제로 구현해야 함

  def shadeToString(self): # self.shade에 저장된 음영 정보를 문자열 형태로 반환하며
                           # createShade 함수 사용해 음영 결정 후 결과 확인 위해 사용

  def angleRangeToCell(self, x, y):
    # 사용자 (self.xPlayer, self.yPlayer)와
    # 셀 (x, y)가 이루는 각도 범위를 AngleRange 객체로 만들어 반환
    # 내용은 수업자료 참조
```

실습문제 풀이에 제공되는 API 이해: Board 클래스 사용 예


```
b1 = Board("map1.txt")    # map1.txt에서 지도 정보 읽어와 Board 객체 생성
b1.createShade1()         # 방법 1을 사용해 음영 정보 결정
print(b1.shadeToString()) # 결정한 결과 출력

ag = b1.angleRangeToCell(3, 2)    # Board 객체 b1에서 사용자와 셀 (3, 2)가 이루는 각도 범위를
                                  # AngleRange 객체로 만들어 ag에 저장
print(ag)                         # ag에 저장한 객체 출력
```

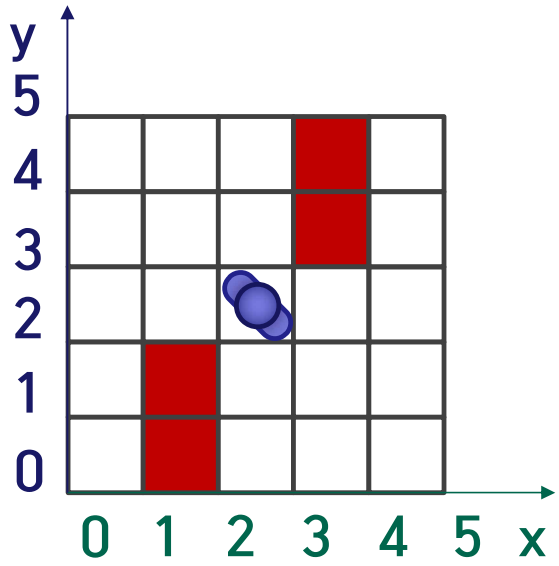
<출력 결과>

```
00011
00001
00000
10000
11000
```

map1.txt

```
00010
00010
00p00
01000
01000
```

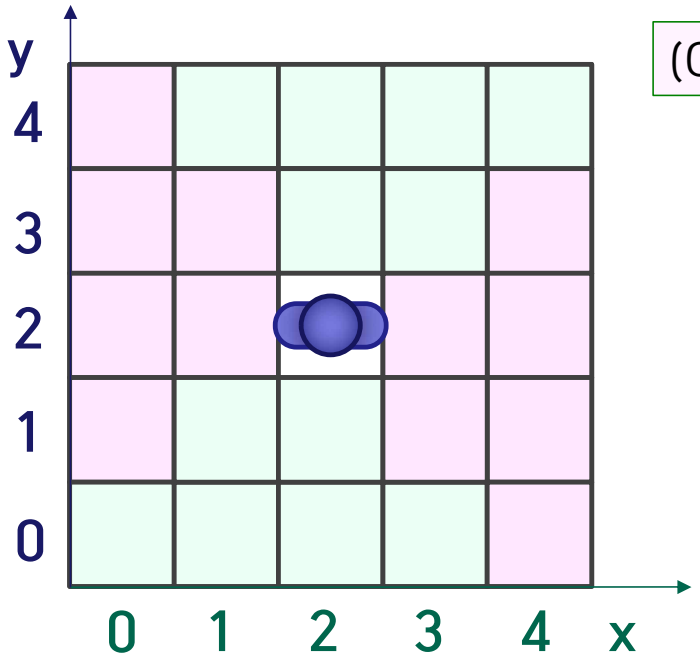
```
(-0.7853981633974483, 0.7853981633974483)
# 셀 (3, 2)는 사용자 바로 오른쪽의 격자이며
# 이 격자와 이루는 각도는 라디안 단위로 위와 같음
```



<정리>

이번 시간 문제의 목표: 장애물을 포함한 지도와 지도에서 사용자의 위치가 주어졌을 때 사용자가 볼 수 있는 영역과 볼 수 없는 영역을 구분하여 다른 색으로 표기하는 효율적인 방법 제시

Lobstacle: player가 볼 수 없는 각도 범위의 리스트
① for distance = 1 to (지도상에서 player로부터 가장 먼 경계선까지의 거리)
② for each cell c1(x1, y1) in (player로부터 distance만큼 떨어진 정사각형)
③ if (cplayer → c1 까지 시야가 Lobstacle에 의해 가린다면) then c1을 음영처리
④ if (c1에 장애물 있음) then c1이 cplayer의 시야를 가리는 각도를 Lobstacle에 추가



(Q) 최종적으로 얻은 방법의 수행 속도는 무엇에 비례하는가?

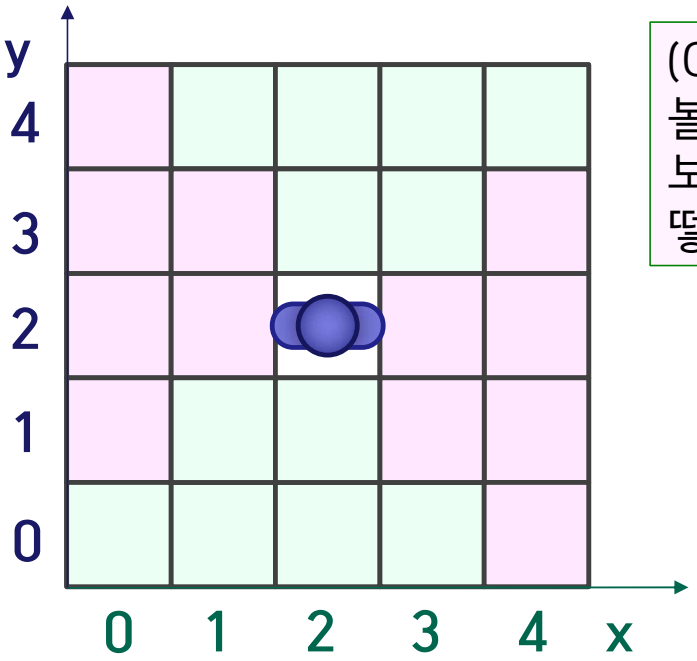
장애물 각도 리스트의 ($L_{obstacle}$) 길이 변화 관찰 예

10 x 10 random map:
2 3 3 4 4 3 4 2

50 x 50 random map
1 2 4 4 4 5 5 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

100 x 100 random map
4 6 5 4 4 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

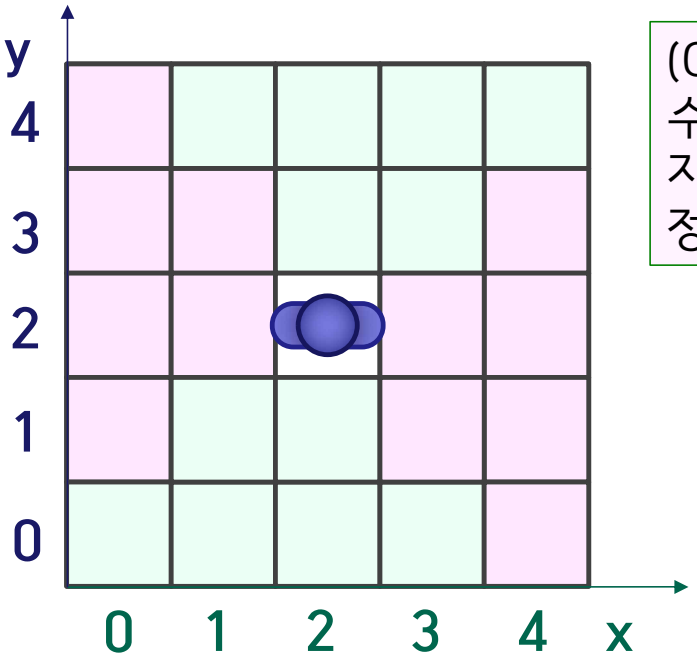
Lobstacle: player가 볼 수 없는 각도 범위의 리스트
① for distance = 1 to (지도상에서 player로부터 가장 먼 경계선까지의 거리)
② for each cell c1(x1, y1) in (player로부터 distance만큼 떨어진 정사각형)
③ if (cplayer → c1 까지 시야가 Lobstacle에 의해 가린다면) then c1을 음영처리
④ if (c1에 장애물 있음) then c1이 cplayer의 시야를 가리는 각도를 Lobstacle에 추가



(Q) 현재 제시한 방법은 사용자가 360° 전 방향을 돌아본다고 가정하고 볼 수 있는 곳과 볼 수 없는 곳 출력한다. 이를 수정하여 사용자가 현재 보는 방향 쪽의 격자만 볼 수 있는 것으로 처리하려면 방법과 코드를 어떻게 수정하여야 할까?

못보는 각도를 미리 Lobstacle에 합쳐놓고 진행한다

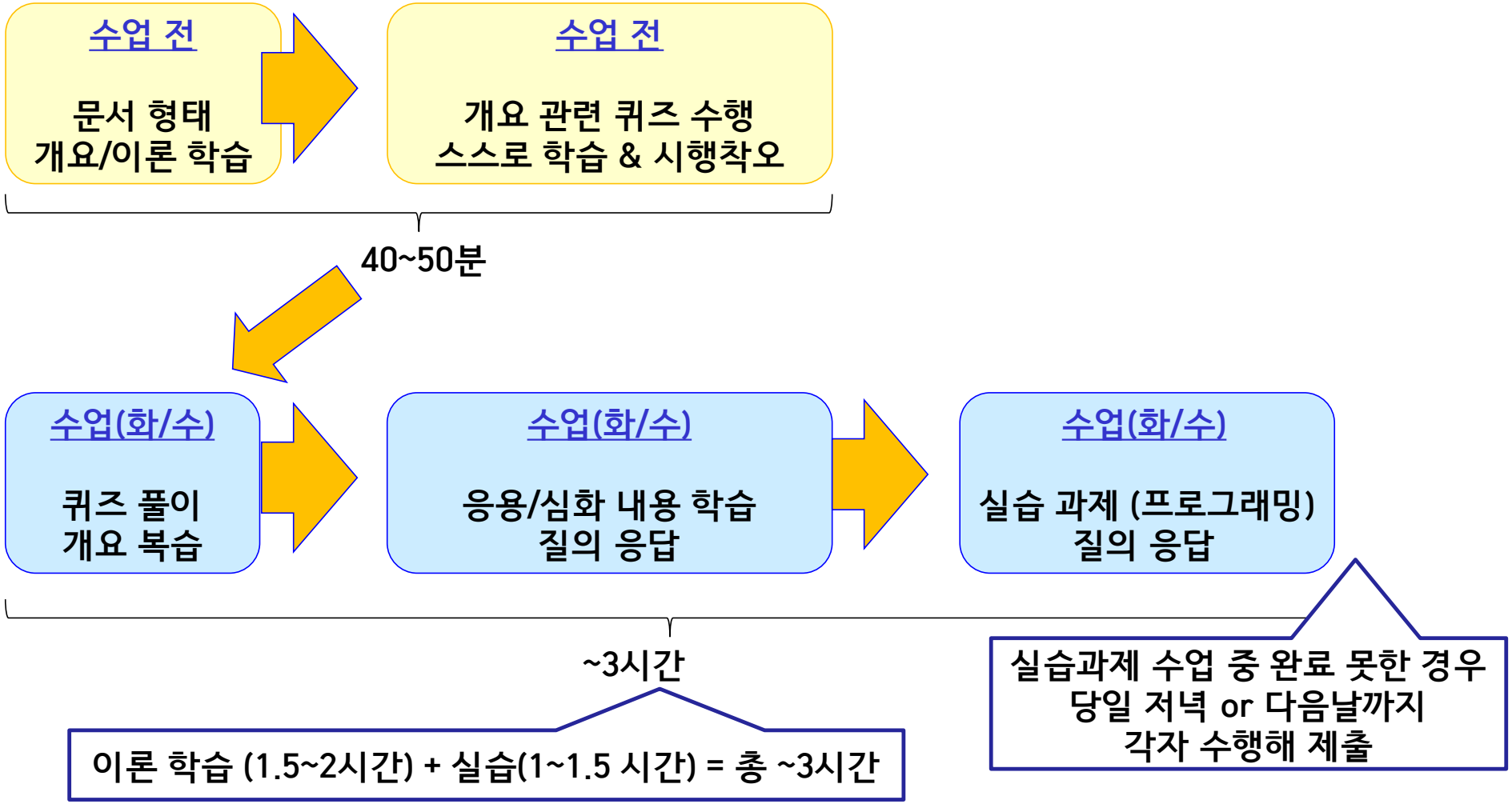
Lobstacle: player가 볼 수 없는 각도 범위의 리스트
① for distance = 1 to (지도상에서 player로부터 가장 먼 경계선까지의 거리)
② for each cell c1(x1, y1) in (player로부터 distance만큼 떨어진 정사각형)
③ if (cplayer → c1 까지 시야가 Lobstacle에 의해 가린다면) then c1을 음영처리
④ if (c1에 장애물 있음) then c1이 cplayer의 시야를 가리는 각도를 Lobstacle에 추가



(Q) 현재 제시한 방법은 장애물에 의해 조금이라도 가려지는 격자는 볼 수 없는 격자로 처리한다. 이를 수정하여 각 격자가 장애물에 의해 가려지는 정도에 따라 다른 음영 값을 부여하려면 방법과 코드를 어떻게 수정하여야 할까?

상수시간이 걸린다
계산식 추가하면 됨.
면에 어디를 통과하는지 알면 될듯?

스마트 출결



05. 실습문제풀이 & 질의응답

- **이번 시간에 배운 내용에 대한 실습 문제 풀이 & 질의 응답**
- **채점 방식은 지난 시간 문제 풀이때와 유사함**

- **왜 중요한가?**
- 이번 주 배운 내용을 총괄하는 문제 풀이 통해 배운 내용 활용 & 복습
- 문제 풀이 점수는 이번 주 과제 점수에 포함됨

createShade3(N) 함수 구현 조건:
이번 시간에 배운 최종 방법 사용해 지도 객체의 음영 결정하는 코드 작성

- 지도를 나타내는 Board 객체가 주어졌을 때 (지도 크기 $1 \leq N \leq 200$)
- 최종 방법에 따라 음영 결정해 멤버 변수 shade에 저장하는 함수 구현

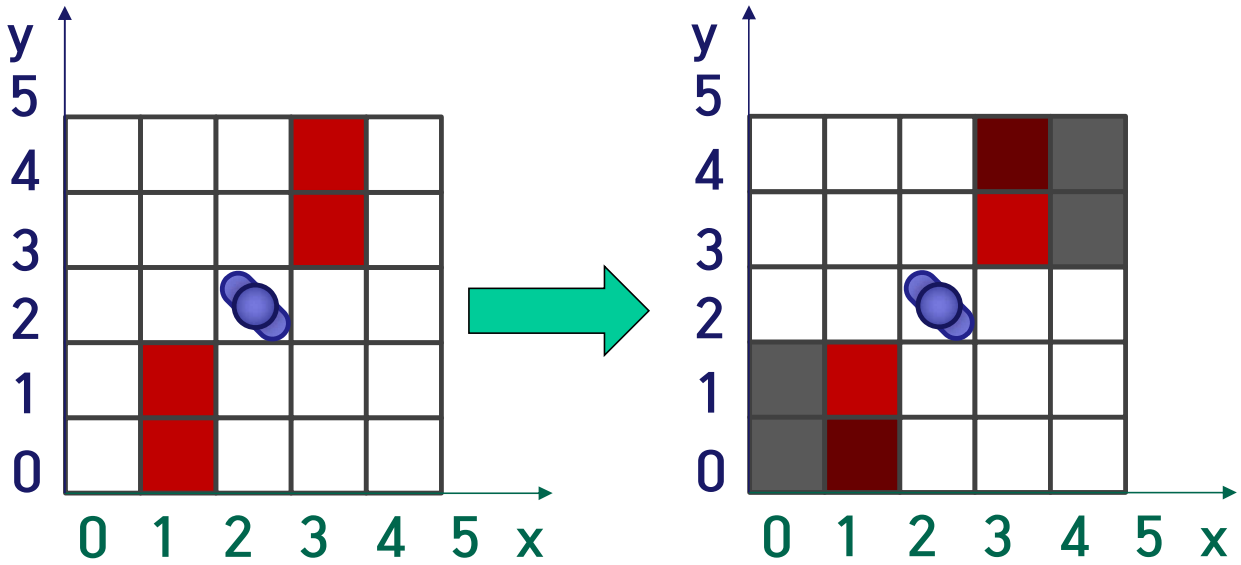
  def createShade3 (N):

- 입력 지도: $1 \leq N \leq 200$ 크기의 지도
- 반환 값: 반환 값은 없으며, 각 격자 (x, y)에 대한 음영 확인 결과를 멤버변수 shade[y][x]에 0 (가려지지 않음) 또는 1 (가려짐) 으로 저장
- 이번 시간에 제공한 코드 Shade.py의 createShade3 () 함수 내에 코드 작성해 제출

입출력 예: cell의 음영 결정해 self.shade에 저장

```
b = Board("map1.txt")
b.createShade3()
print(b.shadeToString())
```

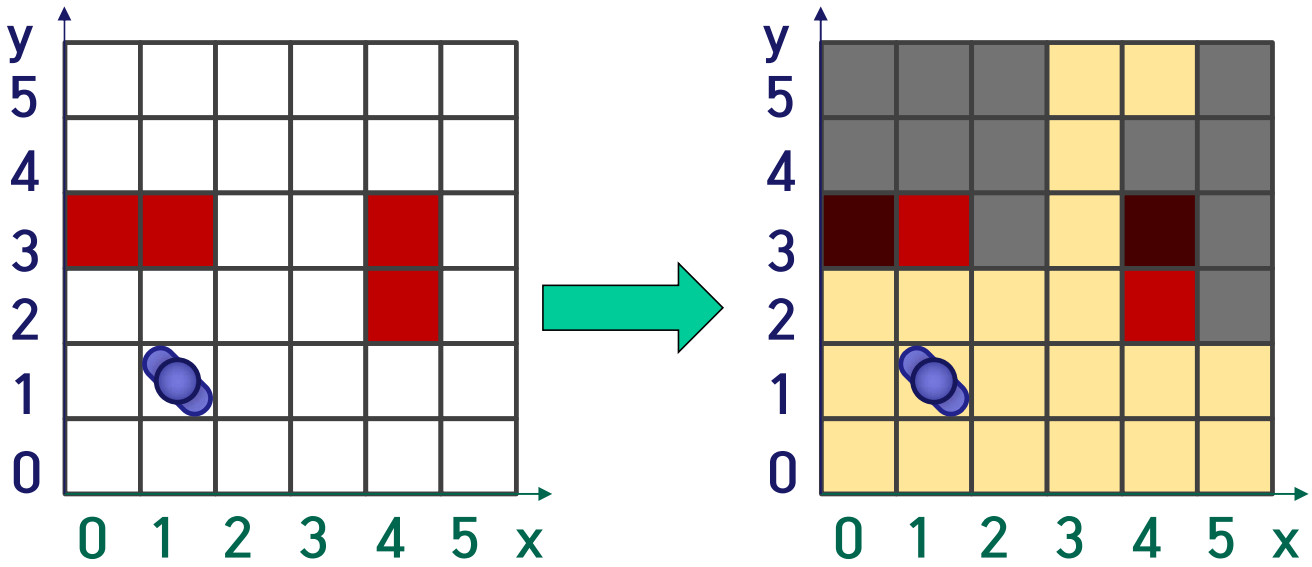
```
00011
00001
00000
10000
11000
```



입출력 예: cell의 음영 결정해 self.shade에 저장

```
b = Board("map2.txt")
b.createShade3()
print(b.shadeToString())
```

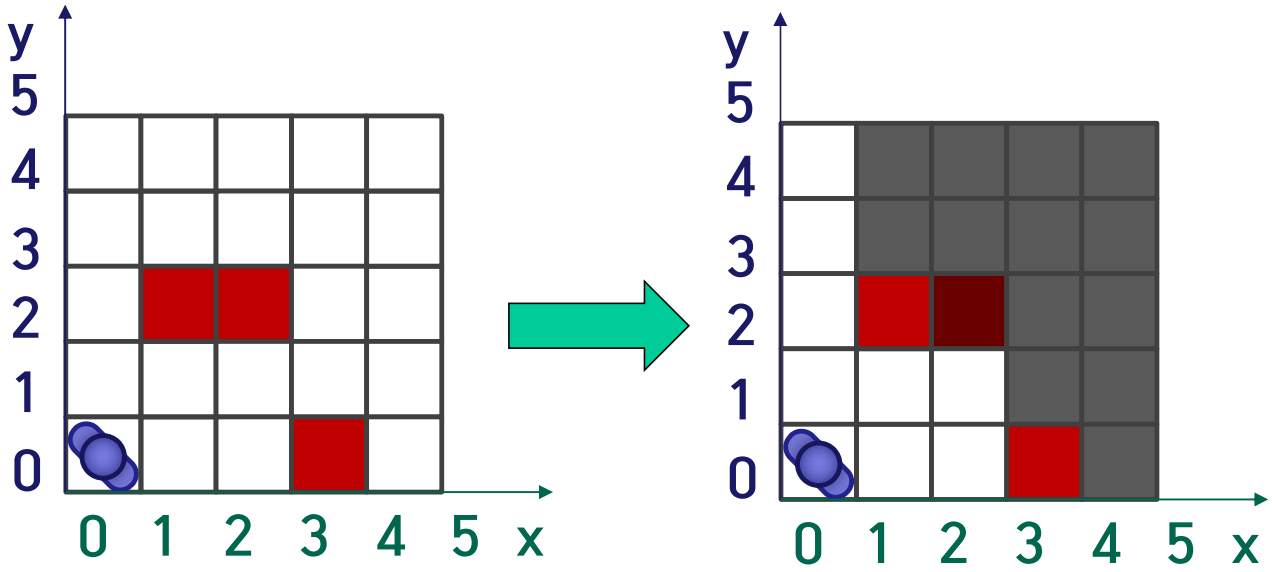
```
111001
111011
101011
000001
000000
000000
```



입출력 예: cell의 음영 결정해 self.shade에 저장

```
b = Board("map3.txt")
b.createShade3()
print(b.shadeToString())
```

```
01111
01111
00111
00011
00001
```



그 외 예제는 __main__ 아래 테스트 코드를 참조하세요.

구현할 함수에 대한 수도 코드: 아래는 하나의 예이며, 반드시 이 로직을 따를 필요는 없음

```
# 구현해야 할 함수. Board 객체에 대해 이 함수를 호출하면 최종 방법에 따라 각 격자의 음영을 결정해 shade에 저장
def createShade3(self)
    def checkCell(x, y): # 격자 (x, y)의 음영을 결정하고, 장애물 각도 목록을 업데이트하는 함수
        # x 혹은 y가 지도 범위를 벗어나면 return
        # ag = 사용자와 격자 (x, y)가 이루는 각도 범위
        # ag가 agObtacle과 겹치면 격자 (x, y)가 장애물에 가림을 의미하므로 self.shade[y][x]에 1 저장
        # self.board[y][x]가 1이면 격자 (x, y)에 장애물이 있는 것이므로 agObstacle에 ag 추가

    # 장애물 각도 목록 나타내는 agObstacle을 비어 있는 AngleRange 객체로 초기화
    # distance가 가질 수 있는 최댓값인 distanceMax 결정
    # 1 ~ distanceMax 범위의 각 distance에 대해 아래 수행
    #    i = 0
    #    while i < distance:
    #       checkCell(self.xPlayer - i, self.yPlayer + distance) # up
    #       checkCell(self.xPlayer + distance, self.yPlayer + i) # right
    #       checkCell(self.xPlayer + i, self.yPlayer - distance) # down
    #       checkCell(self.xPlayer - distance, self.yPlayer - i) # left
    #       i += 1
    #       checkCell(self.xPlayer + i, self.yPlayer + distance) # up
    #       checkCell(self.xPlayer + distance, self.yPlayer - i) # right
    #       checkCell(self.xPlayer - i, self.yPlayer - distance) # down
    #       checkCell(self.xPlayer - distance, self.yPlayer + i) # left
```

그 외 프로그램 구현 조건

- 최종 결과물로 Shade.py 파일 하나만 제출하며, 이 파일만으로 코드가 동작해야 함

- import는 사용할 수 없음

- __main__ 아래의 코드는 작성한 함수가 올바름을 확인하는 코드로
- 코드 작성 후 스스로 채점해 보는데 활용하세요.
- 각 테스트 케이스에 대해 P(or Pass) 혹은 F(or Fail)이 출력됩니다.
- **map?.txt는 채점에 사용되는 지도 파일이며 Shade.py와 같은 폴더에 두고 실행**하세요.
- 코드를 제출할 때는 __main__ 아래 코드는 제거하거나 수정하지 말고 제출합니다. (채점에 사용되기 때문)

- **속도 테스트는 거의 항상 pass 해야만 통과**입니다.

이번 시간 제공 코드 함께 보기

- Shade.py

실습 문제 풀이 & 질의 응답

- 종료 시간(17:00) 이전에 **일찍 모든 문제를 통과**한 경우 각자 **퇴실 가능**
- 실습 문제에 대한 코드는 lms 과제함에 **다음 날 23:59까지 제출** 가능합니다.
- 마감 시간까지 작성을 다 못한 경우는 (제출하지 않으면 0점이므로) 그때까지 작성한 코드를 꼭 제출해 부분점수를 받으세요.

- 실습 문제는 **개별 평가**입니다. 제출한 코드에 대한 유사도 검사를 실습 문제마다 진행하며, 코드를 건내 준 사례가 발견되면 0점 처리됩니다.
- 로직에 대해 서로 의견을 나누는 것은 괜찮지만, 코드를 직접 건내 주지는 마세요.
- 본인 실력 향상을 위해서도 **코드는 꼭 각자 직접 작성**해 주세요.