1. 라이브러리 모음

```
class Board:
    # 필요한 라이브러리 임포트
    import random
    import sys
    import pygame

    # 기본 생성자: 보드의 크기와 구덩이 확률을 초기화
    def __init__(self, size, pit_probability):
        self.size = size
        self.grid = [[None] * size for _ in range(size)]  # 세계를 나타내는 2D
그리드
        self.createWorld(pit_probability)  # 주어진 구덩이 확률로 세계 생성

        # Pygame 초기화
        self.pygame.init()

        # 디스플레이 설정
        self.cell_size = 100
        self.window_size = (self.size * self.cell_size, self.size *
self.cell_size)
        self.screen = self.pygame.display.set_mode(self.window_size)
        self.pygame.display.set_caption('Wumpus World')

        # 이미지 로드
        self.images = {
            'tile': self.pygame.image.load('graphics/tile.png'),
            'gold': self.pygame.image.load('graphics/gold.png'),
            'wumpus': self.pygame.image.load('graphics/wumpus.png'),
            'pit': self.pygame.image.load('graphics/pit.png'),
            'player': self.pygame.image.load('graphics/player.png')
        }
```

2. 보드 구성 (금, 늪, Wumpus)

```
3. # 금, 웜퍼스 및 구덩이로 세계를 생성하는 함수
    def createWorld(self, pit_probability):
        def randomCell(first_row=0, first_col=0):
            row, col = (0, 0)
            while (row, col) in {(0, 0), (0, 1), (1, 0)}:
                row, col = self.random.randint(first_row, self.size - 1),
    self.random.randint(first_col, self.size - 1)
            return row, col

        # 금을 배치
        gold_row, gold_col = randomCell()
        self.grid[gold_row][gold_col] = ['G']

        # 웜퍼스를 배치
        wumpus_row, wumpus_col = randomCell()
        if not self.grid[wumpus_row][wumpus_col]:
            self.grid[wumpus_row][wumpus_col] = ['W']
        else:
```

```
            self.grid[wumpus_row][wumpus_col].append('W')

        # 구덩이를 배치
        for row in range(self.size):
            for col in range(self.size):
                if (row, col) not in {(0, 0), (0, 1), (1, 0)} and not
    self.grid[row][col] and self.random.random() <= pit_probability:
                    self.grid[row][col] = ['P']

        # 세계 출력
        for row in reversed(self.grid):
            print(row)
```

3. GUI를 위해 image 삽입

```
# 보드에서 위치(x, y)가 유효한지 확인하는 함수
def checkLocation(self, x, y):
    return 0 <= x < self.size and 0 <= y < self.size

# 보드의 현재 상태를 표시하는 함수
def display(self, player):
    for event in self.pygame.event.get():
        if event.type == self.pygame.QUIT:
            self.pygame.quit()
            self.sys.exit()

    # 배경 타일을 그리기
    for y in range(self.size):
        for x in range(self.size):
            tile = self.pygame.transform.scale(self.images['tile'],
(self.cell_size, self.cell_size))
            self.screen.blit(tile, (x * self.cell_size, (self.size - y - 1) *
self.cell_size))

    # 배경 타일 위에 요소 그리기
    for y in range(self.size):
        for x in range(self.size):
            cell_content = self.grid[y][x]

            if cell_content:
                # 요소의 다른 조합을 확인하고 해당 이미지를 선택
                if 'G' in cell_content:
                    image = self.pygame.transform.scale(self.images['gold'],
(self.cell_size, self.cell_size))
                elif 'W' in cell_content:
                    image = self.pygame.transform.scale(self.images['wumpus'],
(self.cell_size, self.cell_size))
                elif 'P' in cell_content:
                    image = self.pygame.transform.scale(self.images['pit'],
(self.cell_size, self.cell_size))
                else:
                    continue

                # 현재 셀에 이미지 그리기
                self.screen.blit(image, (x * self.cell_size, (self.size - y -
1) * self.cell_size))
```

```
    # 플레이어 그리기
    player_image = self.pygame.transform.scale(self.images['player'],
(self.cell_size, self.cell_size))
    self.screen.blit(player_image, (player.position[1] * self.cell_size,
(self.size - player.position[0] - 1) * self.cell_size))

    self.pygame.display.flip()
```

4. Pygame library를 이용해 인식기(percept) 만들기

```
class Player:
    # 필요한 라이브러리 임포트
    import random
    import pygame

    # 기본 생성자: 플레이어 속성 초기화
    def __init__(self):
        self.position = (0, 0)
        self.orientation = 'right'
        self.arrow = 3
        self.score = 0
        self.action = ''
        self.visited = {}  # 방문한 위치와 그 지각을 저장할 딕셔너리
        self.attempted = set()  # 시도한 위치를 저장할 집합
        self.visited_repeat = []  # 반복 방문한 위치를 저장할 리스트
        self.has_gold = False

    # 현재 방향을 기준으로 플레이어를 왼쪽으로 회전하는 함수
    def turnLeft(self, board):
        directions = ['up', 'left', 'down', 'right']
        self.orientation =
directions[(directions.index(self.orientation) - 1) % 4]
        self.score -= 1

    # 현재 방향을 기준으로 플레이어를 오른쪽으로 회전하는 함수
    def turnRight(self, board):
        directions = ['up', 'right', 'down', 'left']
        self.orientation =
directions[(directions.index(self.orientation) + 1) % 4]
        self.score -= 1

    # 현재 위치의 지각을 얻는 함수
    def getPerceptions(self, board):
        perceptions = []
        y, x = self.position

        # 인접한 셀에서 웜퍼스, 구덩이 또는 금을 확인하고 해당 지각을 추가
        for i, j in [(y+1, x), (y-1, x), (y, x+1), (y, x-1)]:
            if board.checkLocation(i, j) and board.grid[i][j] is not
None:
                for p in board.grid[i][j]:
                    if p == 'W':
                        perceptions.append('Stench')
                    elif p == 'P':
                        perceptions.append('Breeze')
                    elif p == 'G':
                        perceptions.append('Glitter')
```

```
        return perceptions

    # 현재 방향으로 플레이어를 앞으로 이동하는 함수
    def moveForward(self, board):
        y, x = self.position
        if self.orientation == 'right':
            x += 1
        elif self.orientation == 'left':
            x -= 1
        elif self.orientation == 'up':
            y += 1
        elif self.orientation == 'down':
            y -= 1

        # 새로운 위치가 유효한 보드 범위 내에 있는지 확인
        if board.checkLocation(y, x):
            self.position = (y, x)
        else:
            # 에이전트가 벽에 부딪힘
            print('\tBump...')

        y, x = self.position
        current_pos = board.grid[y][x]

        # 플레이어가 구덩이 또는 웜퍼스를 만나는지 또는 게임에서 승리하는지 확인
        if current_pos and any(element in ["P", "W"] for element in
current_pos):
            print('------- GAME OVER -------')
            self.score -= 1000
        elif current_pos and 'G' in current_pos:
            print("\t Grab Gold Found!!!")
            self.grabGold(board)

    # 금을 잡고 보드에서 제거하는 함수
    def grabGold(self, board):
        y, x = self.position
        board.grid[y][x] = None
        self.score += 1000
        self.has_gold = True

    # 지정된 방향으로 화살을 쏘는 함수
    def shootArrow(self, board, y, x, orientation):
        target = board.grid[y][x]  # 목표 셀의 내용을 가져옴
        self.orientation = orientation  # 쏘기 전에 방향을 설정
        self.arrow -= 1

        if target and 'W' in target:
            # 웜퍼스가 죽음
            print("\t!!! S C R E A M !!!")
            self.score -= 10
            self.visited[(self.position[0], self.position[1])] =
self.getPerceptions(board)

            # 보드에서 웜퍼스를 제거
            if len(target) > 1:
                board.grid[y][x].remove('W')
            else:
                board.grid[y][x] = None
```

```
    # 특정 목적지로 이동할 방향을 결정하는 함수
    def returnToDirection(self, destination_y, destination_x, board):
        y, x = self.position

        # 목적지가 인접한 셀에 있는지 확인하고 해당 방향 반환
        if (y+1, x) == (destination_y, destination_x):
            return 'up'
        elif (y-1, x) == (destination_y, destination_x):
            return 'down'
        elif (y, x+1) == (destination_y, destination_x):
            return 'right'
        elif (y, x-1) == (destination_y, destination_x):
            return 'left'
        else:
            # 좌표가 일치하지 않으면 방문한 위치를 기준으로 방향 선택
            possible_directions = [(i, j, orientation) for i, j,
orientation in [(y+1, x, 'up'), (y-1, x, 'down'), (y, x+1, 'right'),
(y, x-1, 'left')] if board.checkLocation(i, j)]
            directions_visited = [(i, j, orientation) for i, j,
orientation in possible_directions if (i, j) in self.visited.keys()]
            direction = directions_visited[self.random.randint(0,
len(directions_visited) - 1)][2] if directions_visited else
possible_directions[self.random.randint(0, len(possible_directions) -
1)][2]

            return direction

    # 방문하지 않은 방향을 무작위로 선택하는 함수
    def randomDirection(self, board):
        y, x = self.position
        possible_directions = [(i, j, orientation) for i, j, orientation
in [(y+1, x, 'up'), (y-1, x, 'down'), (y, x+1, 'right'), (y, x-1,
'left')] if board.checkLocation(i, j)]
        directions_not_visited = [(i, j, orientation) for i, j,
orientation in possible_directions if (i, j) not in
self.visited.keys()]
        direction = directions_not_visited[self.random.randint(0,
len(directions_not_visited) - 1)] if directions_not_visited else
possible_directions[self.random.randint(0, len(possible_directions) -
1)]

        return direction

    # 지정된 방향으로 무작위로 회전하거나 무작위로 방향을 선택하는 함수
    def turnRandom(self, board, destination=None, return_safe=False):
        if return_safe:
            y, x = destination
            direction = self.returnToDirection(y, x, board)
        else:
            direction = self.randomDirection(board)[2]

        # 선택된 방향으로 회전
        if self.orientation != direction:
            directions = ['up', 'right', 'down', 'left']
            clockwise = (directions.index(direction) -
directions.index(self.orientation)) % 4
            counterclockwise = (directions.index(self.orientation) -
directions.index(direction)) % 4
```

```
            turn = 'right' if clockwise <= counterclockwise else 'left'

            # 시계 방향 또는 반시계 방향으로 회전하여 방향을 맞춤
            while self.orientation != direction:
                if turn == 'right':
                    self.turnRight(board)
                else:
                    self.turnLeft(board)

    # 다음 안전한 목적지로 이동하는 함수
    def goToSafe(self, board, clock):
        coincidence = False

        # 방문한 위치를 반복하면서 다음 안전한 목적지를 찾음
        for key in reversed(self.visited_repeat):
            if (not self.visited[key] or 'Glitter' in self.visited[key])
and key not in self.attempted:
                self.attempted.add(key)
                destination = key
                coincidence = True
                break

        if coincidence:
            # 나쁜 지각이 없는 타일로 이동
            current_y, current_x = self.position

            while (current_y, current_x) != destination:
                # 무작위로 이미 방문한 위치로 회전
                self.turnRandom(board, destination, return_safe=True)

                # 게임 보드를 표시하고 지연 시간 설정
                self.pygame.time.delay(1000)
                clock.tick(30)
                self.moveForward(board)
                board.display(self)

                current_y, current_x = self.position

    # 시작점으로 안전하게 돌아가는 함수
    def climb(self, board, clock):
        self.goToSafe(board, clock)
        while self.position != (0, 0):
            self.turnRandom(board, (0, 0), return_safe=True)
            self.moveForward(board)
            board.display(self)
            self.pygame.time.delay(1000)
            clock.tick(30)
        print("금과 함께 시작점으로 안전하게 돌아왔습니다!")
        print("게임 종료. 승리했습니다!")
        self.pygame.time.delay(2000)
        self.pygame.quit()
        self.sys.exit()

    # 방문한 위치를 유지하면서 플레이어 상태를 리셋하는 함수
    def reset(self):
        self.position = (0, 0)
        self.orientation = 'right'
```

```
        self.arrow = 3
        self.action = ''
        self.has_gold = False
```

5. Agent가 금을 찾기 위한 최대한 안전한 상황을 만드는 과정

```
class WumpusWorld:
    # 필요한 라이브러리 임포트
    import pygame

    # 기본 생성자: 보드와 플레이어로 게임 초기화
    def __init__(self, size=4, pit_probability=0.1):
        self.board = Board(size, pit_probability)
        self.player = Player()

    # 위험한 상황(Stench 또는 Breeze)을 처리하는 함수
    def handleDanger(self, perceptions, clock):
        if 'Stench' in perceptions:
            # Stench가 감지되면 플레이어는 화살을 쏠지 이동할지 결정
            y, x, orientation = self.player.randomDirection(self.board)
            if self.player.arrow > 0:
                # 화살이 있으면 웜퍼스를 쏘기
                self.player.action = 'Shoot'
                print("Action:", self.player.action)
                self.player.shootArrow(self.board, y, x, orientation)
            else:
                # 화살이 없으면 안전한 위치로 이동 시도
                if len(self.player.visited) > 1:
                    self.player.goToSafe(self.board, clock)
                self.player.turnRandom(self.board)
                self.player.action = 'MoveForward'
                print("Action:", self.player.action)
                self.player.moveForward(self.board)
        elif 'Breeze' in perceptions:
            # Breeze가 감지되면 안전한 위치로 이동 시도
            if len(self.player.visited) > 1:
                self.player.goToSafe(self.board, clock)
            self.player.turnRandom(self.board)
            self.player.action = 'MoveForward'
            print("Action:", self.player.action)
            self.player.moveForward(self.board)
```

6. Wumpus game 실행 함수

```
# Wumpus World 게임을 실행하는 함수
    def play(self):
        # 프레임 속도를 제어하기 위해 Pygame 시계 초기화
        clock = self.pygame.time.Clock()

        # 메인 게임 루프
        while True:
            # 현재 위치를 기반으로 플레이어의 지각을 가져옴
            perceptions = self.player.getPerceptions(self.board)

            # 게임의 현재 상태를 표시
```

```
            self.board.display(self.player)

            # 방문한 타일과 지각 정보를 업데이트
            current_tile = (self.player.position[0],
self.player.position[1])
            self.player.visited[current_tile] = perceptions
            self.player.visited_repeat.append(current_tile)

            # 게임의 현재 상태를 출력
            print("\n-----------------\n")
            print("\tCurrent State:")
            print("Player Position:", self.player.position)
            print("Perceptions:", perceptions)

            if 'Glitter' in perceptions:
                # Glitter가 감지되면 금을 잡기 전에 잠재적 위험 확인
                if 'Breeze' in perceptions or 'Stench' in perceptions:
                    self.handleDanger(perceptions, clock)
                # 안전하다면 무작위로 회전하고 앞으로 이동하여 금을 잡기
                else:
                    self.player.turnRandom(self.board)
                    self.player.action = 'MoveForward'
                    print("Action:", self.player.action)
                    self.player.moveForward(self.board)
                    if self.player.has_gold:
                        self.player.climb(self.board, clock)

            # 위험한 상황(Pit 또는 Wumpus)을 확인
            elif 'Breeze' in perceptions or 'Stench' in perceptions:
                self.handleDanger(perceptions, clock)
            # 위험이 없으면 무작위로 이동
            else:
                self.player.turnRandom(self.board)
                self.player.action = 'MoveForward'
                print("Action:", self.player.action)
                self.player.moveForward(self.board)

            # 플레이어의 현재 방향과 점수를 출력
            print('Orientation:', self.player.orientation)
            print("Score:", self.player.score)

            # 게임을 시각화하기 위해 지연을 추가하고 프레임 속도를 초당 30 프레임으로
제한
            self.pygame.time.delay(1000)
            clock.tick(30)

            # 게임의 종료 조건(승리 또는 패배)을 확인
            if self.player.score <= -1000:
                print("Restarting from (0, 0)")
                self.player.reset()

# 게임을 초기화하고 실행
world = WumpusWorld()
world.play()
```