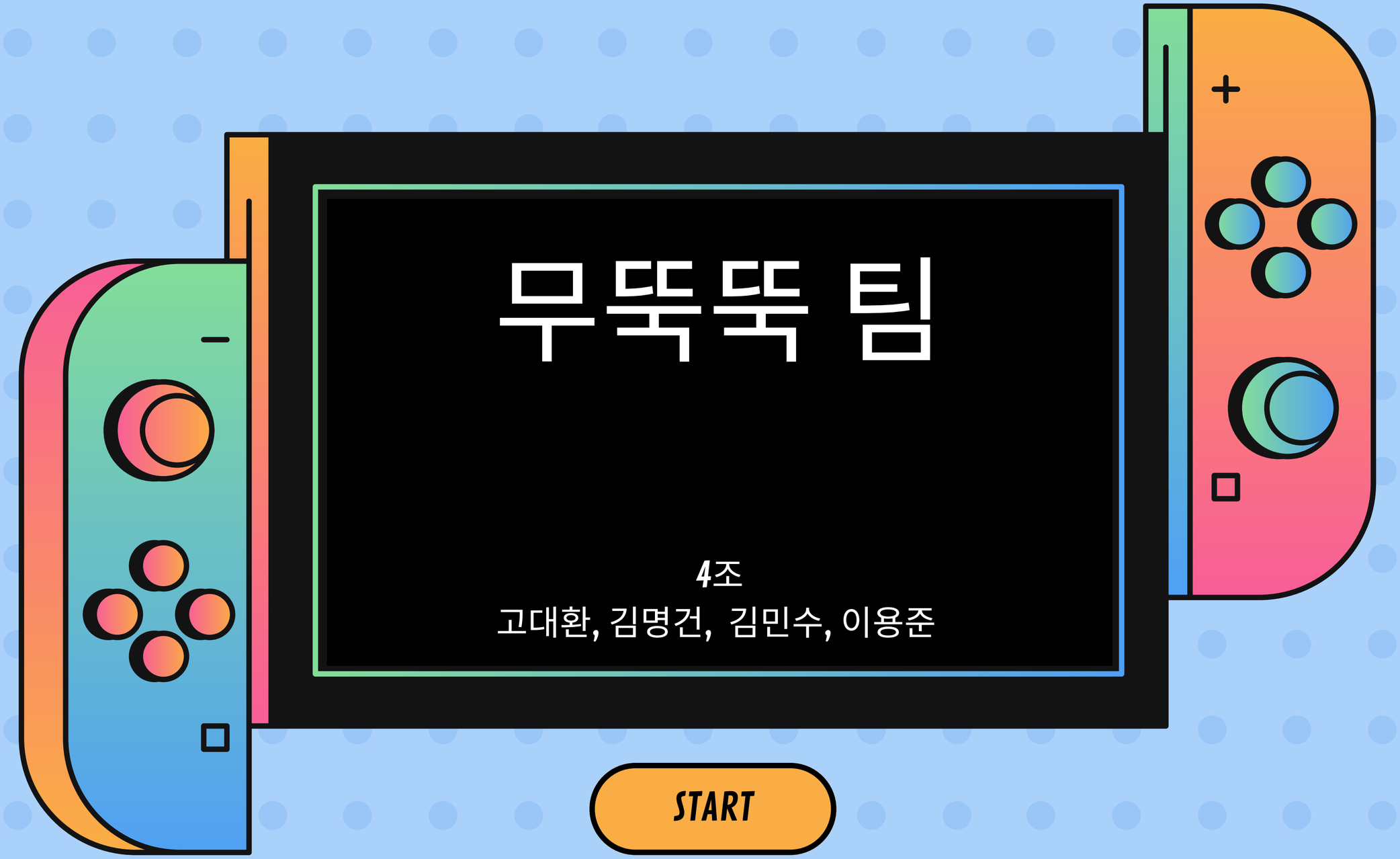
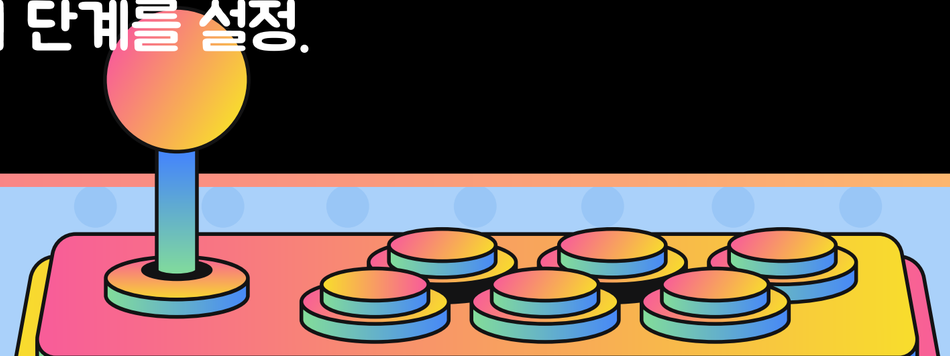

무뚝뚝 팀
4조
고대환, 김명건, 김민수, 이용준
START

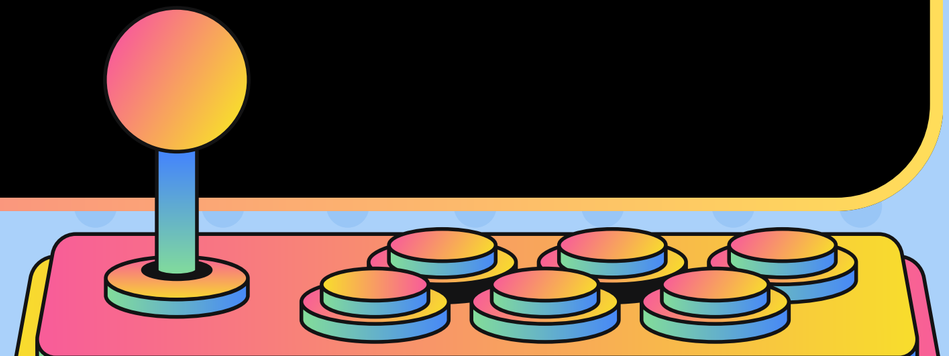
초기 제안서

1.플레이어와 COM(red)으로 나뉘는 두 개의 거북이가 존재.
2.플레이어의 거북이는 플레이어의 키보드 입력을 바탕으로 이동.
3.주어진 미로에서 플레이어는 뒤쫓아오는 COM을 피하여 제한시간 내에 탈출하는 것이 목적.
4.COM(red)는 플레이어를 뒤쫓아 둘이 충돌할 경우 게임 오버.
5.미로 내에서는 다양한 아이템 존재하고 이를 탈출에 적절히 활용.
6.미로의 막다른 부분에는 무작위 아이템이 반드시 존재.
7.아이템 종류 (COM을 멈춤, 목숨 증가, 속도 증가 등.)
8. 게임 오버 시 '게임 오버' 글자를 표시하고 새 게임을 시작 혹은 게임 종료 선택.
9.게임 시작 전에 아이템에 대한 설명, 게임 방법에 대해 설명.
10.COM의 속도를 통해 게임의 단계를 설정.



중간 제안서

1.플레이어와 COM(red)으로 나뉘는 두 개의 거북이가 존재.
2.플레이어의 거북이는 플레이어의 키보드 입력을 바탕으로 이동.
3.주어진 미로에서 플레이어는 뒤쫓아오는 COM을 피하여 제한시간 내에 탈출하는 것이 목적.
4.COM(red)는 플레이어를 뒤쫓아 둘이 충돌할 경우 게임 오버.
5.게임 오버 시 '게임 오버' 글자를 표시하고 새 게임을 시작 혹은 게임 종료 선택.
6.게임 시작 전에 게임 방법에 대해 설명.
7.COM의 속도를 통해 게임의 단계를 설정.

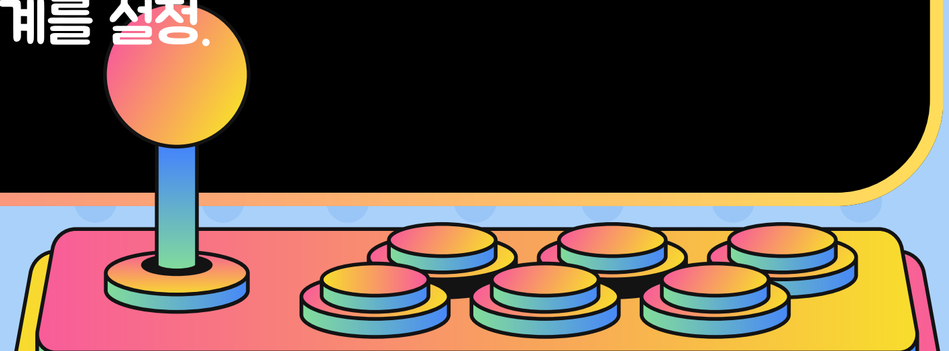


변화한 점

아이템 기능 삭제
게임방법 수정
(제한시간 삭제)

중간 제안서

1.플레이어와 COM(red)으로 나뉘는 두 개의 거북이가 존재.
2.플레이어의 거북이는 플레이어의 키보드 입력을 바탕으로 이동.
3.주어진 미로에서 플레이어는 뒤쫓아오는 COM을 피하여 제한시간 내에 탈출하는 것이 목적.
4.COM(red)는 플레이어를 뒤쫓아 둘이 충돌할 경우 게임 오버.
5.COM(red)가 벽에 끼는 현상을 해결하는 방법 고안
6.게임 오버 시 '게임 오버' 글자를 표시하고 새 게임을 시작 혹은 게임 종료 선택.
7.게임 시작 전에 게임 방법에 대해 설명.
8.적이 몇 명이 게임하기 좋을지에 대한 고민하기
9.게임 시작 전 적의 개수를 설정하는 화면을 띄워 적의 개수 설정 가능하게 만들기
10.COM의 속도를 통해 게임의 단계를 설정.

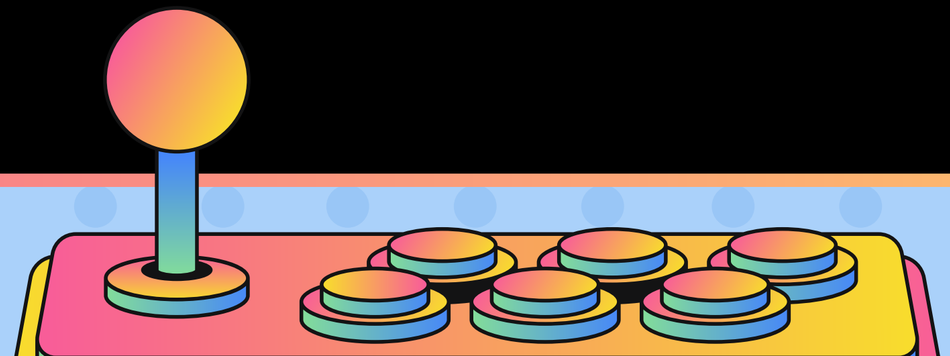


변화한 점

적 개수 설정기능
게임방법 수정
오류 해결

최종 제안서

1.Prim's Algorithm을 사용하여 무작위로 미로를 생성한다.
2.미로 내에 'S'(시작)와 'E'(종료)를 각각 초록색과 빨간색으로 표시한다.
3.파란색 원형의 플레이어 캐릭터와 빨간색 원형 적 캐릭터
4.적은 플레이어와의 최단 경로를 BFS(너비 우선 탐색)알고리즘을 사용하여 추격한다.
5.플레이어의 거북이는 플레이어의 키보드 입력을 바탕으로 이동.
6.플레이어가 새로운 위치를 방문할 때마다 점수가 증가하는 방식으로 점수 시스템이 작동.
7.COM(red)는 플레이어를 뒤쫓아 둘이 충돌할 경우 게임 오버.
8.게임 시작 전에 'Enter' 키를 눌러 게임을 시작할 수 있으며, 안내 텍스트를 중앙에 표시.



게임 소개

적을 피해 탈출하는 미로게임

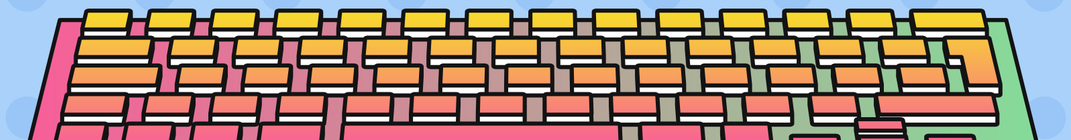
OLIVIA WILSON

HOW TO PLAY

- 무작위로 미로를 생성
- 미로 내에 'S'(시작)와 'E'(종료)를 각각 초록색과 빨간색으로 표시
- 적은 플레이어와의 최단 경로를 사용하여 추격
- 플레이어의 거북이는 플레이어의 키보드 입력을 바탕으로 이동
- 플레이어가 새로운 위치를 방문할 때마다 점수가 증가
- 적과 충돌할 경우 게임 오버, 적을 피해 탈출

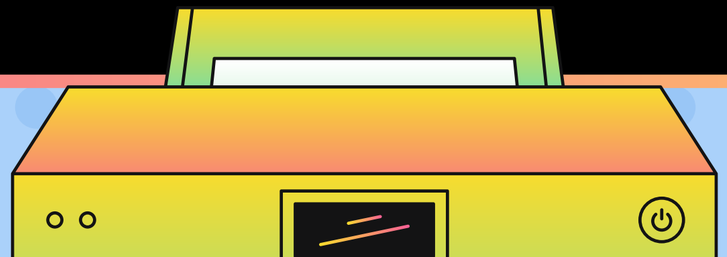
핵심 기능

- 미로 생성
- 적 생성
- 적이 이동
- 플레이어 이동



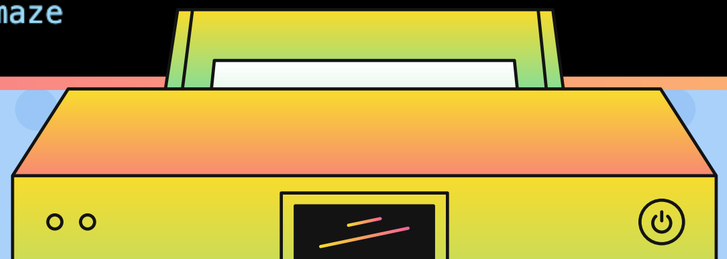
미로생성

```python
# Prim's Algorithm을 이용해 미로 생성
def generate_maze(size):
    # 지정된 크기로 미로 생성 (초기에는 모든 셀을 벽으로 설정)
    maze = [[1 for _ in range(size)] for _ in range(size)]

    def get_neighbors(x, y):
        # 현재 위치에서 상하좌우 이웃 셀을 구함
        neighbors = []
        if x > 1: neighbors.append((x - 2, y))
        if x < size - 2: neighbors.append((x + 2, y))
        if y > 1: neighbors.append((x, y - 2))
        if y < size - 2: neighbors.append((x, y + 2))
        return neighbors

    start_x, start_y = 1, 1
    maze[start_y][start_x] = 0
    walls = [(start_x + dx, start_y + dy) for dx, dy in [(0, 2), (2, 0)]
             if start_x + dx < size and start_y + dy < size]
```



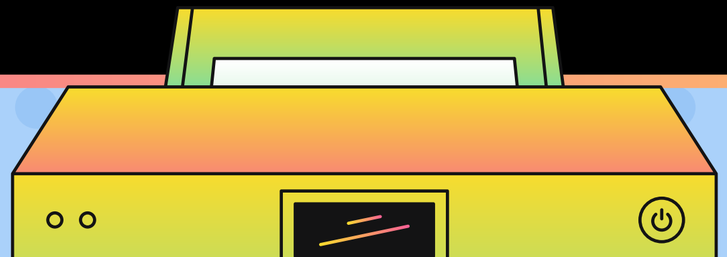
미로생성

```
    # 미로를 생성하면서 벽을 랜덤하게 제거하여 길을 만든다
    while walls:
        x, y = random.choice(walls)
        walls.remove((x, y))

        neighbors = get_neighbors(x, y)
        open_neighbors = [(nx, ny) for nx, ny in neighbors
                          if maze[ny][nx] == 0]

        if open_neighbors:
            nx, ny = random.choice(open_neighbors)
            if x == nx:
                maze[(y + ny) // 2][x] = 0
            else:
                maze[y][(x + nx) // 2] = 0
            maze[y][x] = 0

            for neighbor in get_neighbors(x, y):
                if maze[neighbor[1]][neighbor[0]] == 1:
                    walls.append(neighbor)

    maze[0][1] = 'S'  # 시작점 설정
    maze[size - 1][size - 2] = 'E'  # 종료점 설정
    return maze
```



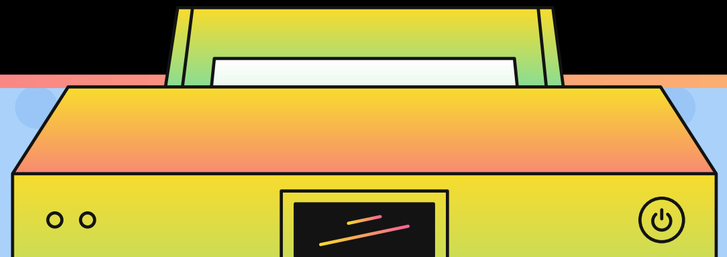
화면 설정

```
# 미로 사이즈 설정
maze_size = 25
maze = generate_maze(maze_size)

# 화면 설정
screen = turtle.Screen()
screen.title("Game Start Screen")
screen.setup(width=800, height=800)
screen.bgcolor("black")
screen.tracer(0)

# 텍스트 표시 설정
start_text = turtle.Turtle()
start_text.hideturtle()
```



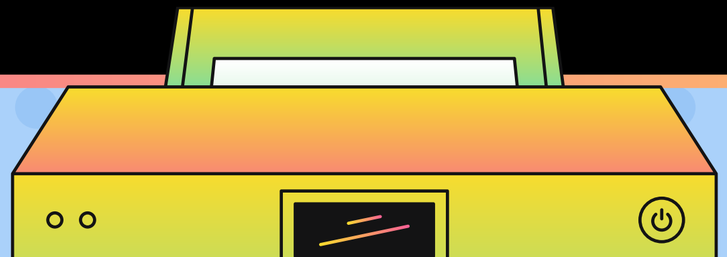
시작 화면 표시

```
# 시작 화면 표시 함수
def display_text_with_border(x, y, message):
    start_text.color("black")
    start_text.penup()
    start_text.goto(0, 0)

    # 테두리 그리기
    offsets = [(0, 2), (2, 0), (0, -2), (-2, 0), (2, 2),
               (-2, -2), (2, -2), (-2, 2)]  # 8방향
    for dx, dy in offsets:
        start_text.goto(0 + dx, 0 + dy)
        start_text.write(message, align="center", font=("Arial", 24, "bold"))
    # 본래 텍스트 그리기
    start_text.color("white")
    start_text.penup()
    start_text.goto(0, 0)
    start_text.write(message, align="center", font=("Arial", 24, "bold"))
```



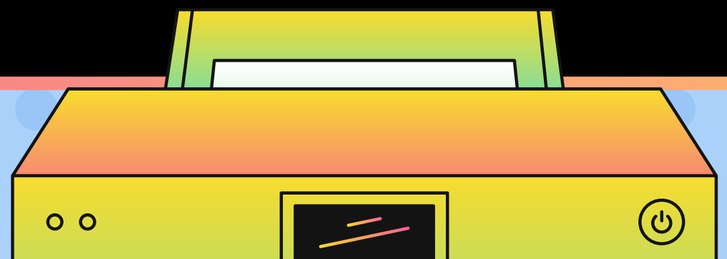
플레이어 설정

```
# 미로와 게임 캐릭터 관련 설정
t = turtle.Turtle()
t.speed('fastest')  # 터틀 이동 속도를 빠르게 설정
t.hideturtle()

# 플레이어 생성
player = turtle.Turtle()
player.shape('circle')
player.color('blue')
player.penup()

# 적 캐릭터 리스트 생성
enemies = []

# 플레이어가 방문한 칸을 추적하는 세트
visited_cells = set()
```



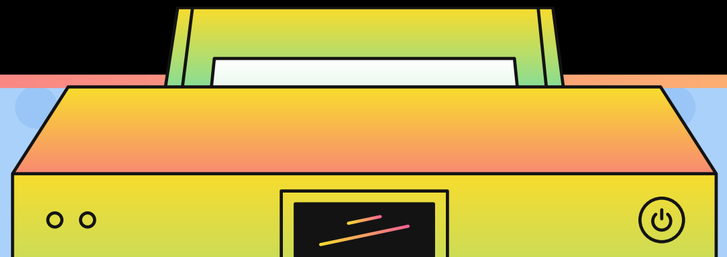
터틀 설정

```
# 점수 표시를 위한 터틀 설정
score_turtle = turtle.Turtle()
score_turtle.hideturtle()
score_turtle.penup()
score_turtle.goto(-maze_size * 10, maze_size * 10 + 20)
score_turtle.color("blue")
score_turtle.write(f"Score: {score}", align="left",
                   font=("Arial", 16, "bold"))
```



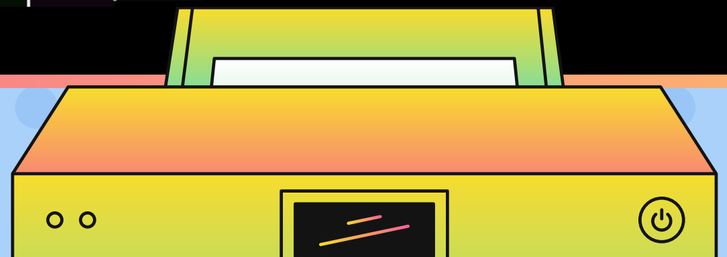
시작 화면 표시

```
# 게임 시작 함수
def start_game():
    # Enter 키를 누르면 게임 시작
    global game_running
    if not game_running:
        return

    screen.bgcolor("white")
    place_player()  # 플레이어 위치를 설정
    place_enemies() # 적 위치를 설정
    key_ctrl()  # 키보드 컨트롤 설정
    start_text.clear()  # 시작 텍스트 삭제
    drawmaze(maze)  # 미로 그리기
```



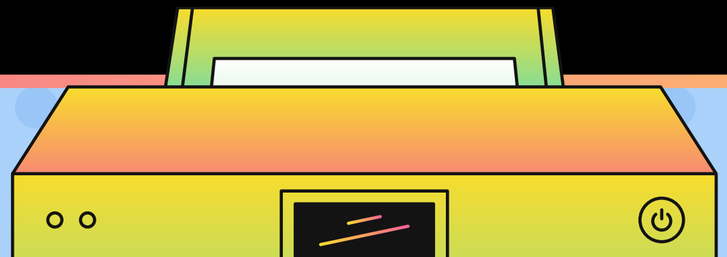
미로 그리기

```python
# 미로 그리기 함수
def drawmaze(maze):
    if not game_running:  # 게임이 종료된 경우 실행하지 않음
        return
    t.penup()
    start_x, start_y = -maze_size * 10, maze_size * 10
    for y in range(len(maze)):
        for x in range(len(maze[y])):
            t.goto(start_x + x * 20, start_y - y * 20)
            if maze[y][x] == 1:
                # 벽을 그리기
                t.pendown()
                t.fillcolor('black')
                t.begin_fill()
                for _ in range(4):
                    t.forward(20)
                    t.right(90)
                t.end_fill()
                t.penup()
```



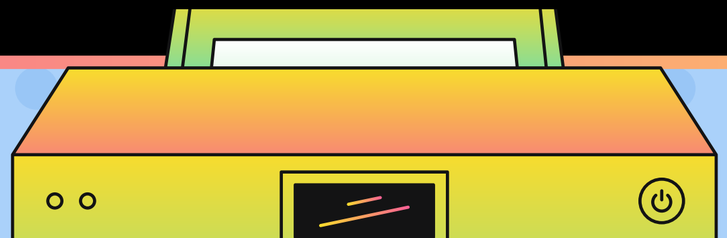
미로 그리기

```
            elif maze[y][x] == 'S':
                # 시작점 표시
                t.pendown()
                t.fillcolor('green')
                t.begin_fill()
                for _ in range(4):
                    t.forward(20)
                    t.right(90)
                t.end_fill()
                t.penup()
            elif maze[y][x] == 'E':
                # 종료점 표시
                t.pendown()
                t.fillcolor('red')
                t.begin_fill()
                for _ in range(4):
                    t.forward(20)
                    t.right(90)
                t.end_fill()
                t.penup()
    screen.update()
```



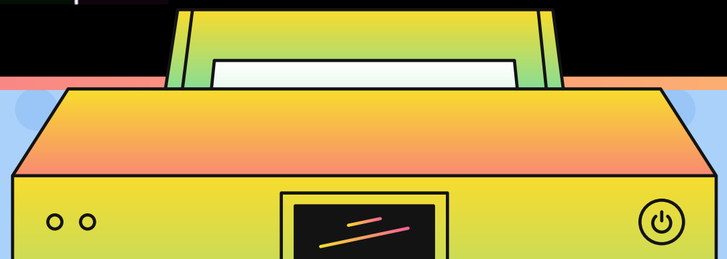
플레이어 위치 설정

```
# 플레이어 위치 설정 함수
def place_player():
    for y in range(len(maze)):
        for x in range(len(maze[y])):
            if maze[y][x] == 'S':
                # 시작점 위치에 플레이어 위치 설정
                start_x = -maze_size * 10 + x * 20 + 10
                start_y = maze_size * 10 - y * 20 - 10
                player.goto(start_x, start_y)
                visited_cells.add((x, y))
                return

drawmaze(maze)
screen.tracer(1)  # 화면 갱신 자동 복원
```



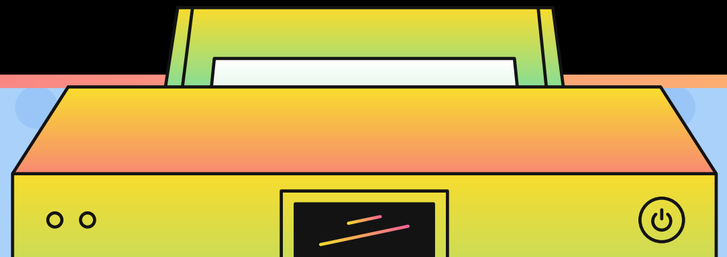
최단경로 탐색

```python
# 적의 현재 위치에서 플레이어 위치까지의 최단 경로를 찾는 함수
def find_path(start_x, start_y, goal_x, goal_y):
    queue = deque([(start_x, start_y)])
    visited = set()
    visited.add((start_x, start_y))
    parent_map = {}

    while queue:
        x, y = queue.popleft()

        # 플레이어 위치에 도달하면 경로 반환
        if (x, y) == (goal_x, goal_y):
            path = []
            while (x, y) != (start_x, start_y):
                path.append((x, y))
                x, y = parent_map[(x, y)]
            path.reverse()
            return path  # 경로 반환
```



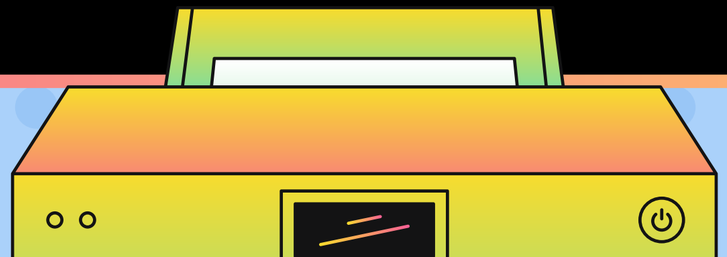
최단경로 탐색

```
        # 상하좌우 이동 가능한지 체크
        for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
            nx, ny = x + dx, y + dy
            if 0 <= nx < len(maze[0]) and 0 <= ny < len(maze) and maze[ny][nx] != 1 and (nx, ny) not in visited:
                queue.append((nx, ny))
                visited.add((nx, ny))
                parent_map[(nx, ny)] = (x, y)

    return []  # 플레이어에게 도달 불가능할 경우 빈 리스트 반환
```



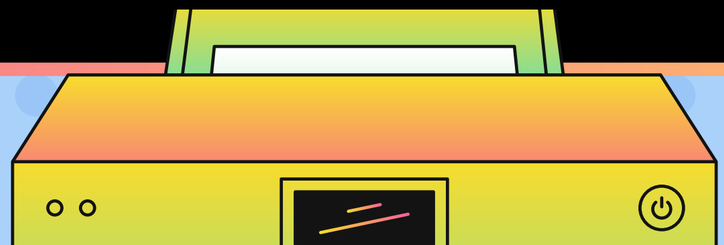
시작 화면 표시

```python
# 적이 플레이어를 따라 이동
def move_enemies():
    global game_running
    if not game_running:  # 게임이 종료된 경우 실행하지 않음
        return
    for enemy in enemies:
        player_x, player_y = player.pos()
        enemy_x, enemy_y = enemy.pos()

        # 적 위치와 플레이어 위치를 비교해서 이동
        enemy_grid_x = int((enemy_x + maze_size * 10) // 20)
        enemy_grid_y = int((maze_size * 10 - enemy_y) // 20)
        player_grid_x = int((player_x + maze_size * 10) // 20)
        player_grid_y = int((maze_size * 10 - player_y) // 20)

        # 최단 경로 찾기
        path = find_path(enemy_grid_x, enemy_grid_y, player_grid_x, player_grid_y)
```



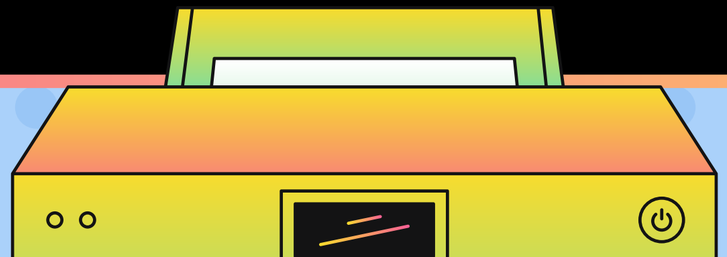
시작 화면 표시

```
        # 경로가 있다면 다음 좌표로 이동
        if path:
            next_x, next_y = path[0]
            new_enemy_x = -maze_size * 10 + next_x * 20 + 10
            new_enemy_y = maze_size * 10 - next_y * 20 - 10
            enemy.goto(new_enemy_x, new_enemy_y)

        # 적이 플레이어와 충돌하면 게임 오버
        if enemy.distance(player) < 20:
            game_running = False
            print("게임 오버!")
            turtle.bye()
            return

    screen.ontimer(move_enemies, 500)  # 적의 이동 속도 설정
```



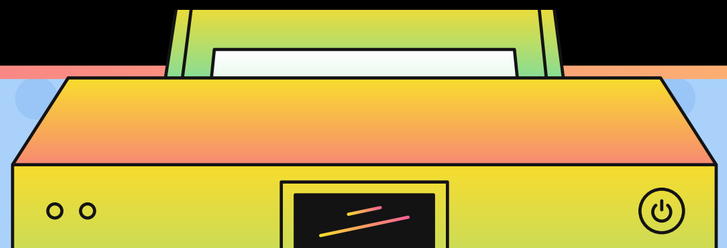
적 생성

```python
# 적 캐릭터 생성
def place_enemies():
    num_enemies = 5

    for _ in range(num_enemies):
        enemy = turtle.Turtle()
        enemy.shape('circle')
        enemy.color('red')
        enemy.penup()
        enemy.speed(0)  # 최대 속도로 설정
        # 미로의 길인 셀 중 무작위로 선택하여 적 배치
        while True:
            ex = random.randint(0, maze_size - 1)
            ey = random.randint(0, maze_size - 1)
            if maze[ey][ex] == 0 and (ex, ey) != (1, 1) and (ex, ey) != (maze_size - 2, maze_size - 1):
                break
        new_enemy_x = -maze_size * 10 + ex * 20 + 10
        new_enemy_y = maze_size * 10 - ey * 20 - 10
        enemy.goto(new_enemy_x, new_enemy_y)
        enemies.append(enemy)

    move_enemies()
```



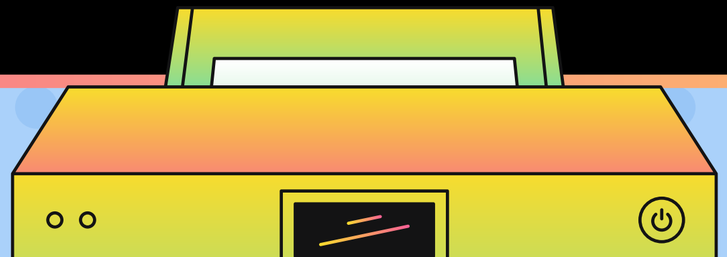
플레이어 이동

```python
# 플레이어가 이동할 위치 계산
def move_to_grid(x, y):
    new_x = -maze_size * 10 + x * 20 + 10
    new_y = maze_size * 10 - y * 20 - 10
    player.goto(new_x, new_y)
    if (x, y) not in visited_cells:
        visited_cells.add((x, y))
        update_score()

# 플레이어 상하좌우 이동 함수
def move(dx, dy):
    if not game_running:
        return

    x, y = player.pos()
    grid_x = int((x + maze_size * 10) // 20)
    grid_y = int((maze_size * 10 - y) // 20)
    new_x, new_y = grid_x + dx, grid_y + dy
    if 0 <= new_x < len(maze[0]) and 0 <= new_y < len(maze) and maze[new_y][new_x] != 1:
        move_to_grid(new_x, new_y)
        check_win()
```



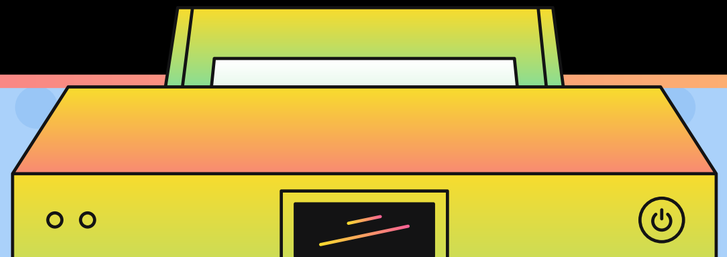
기타 함수

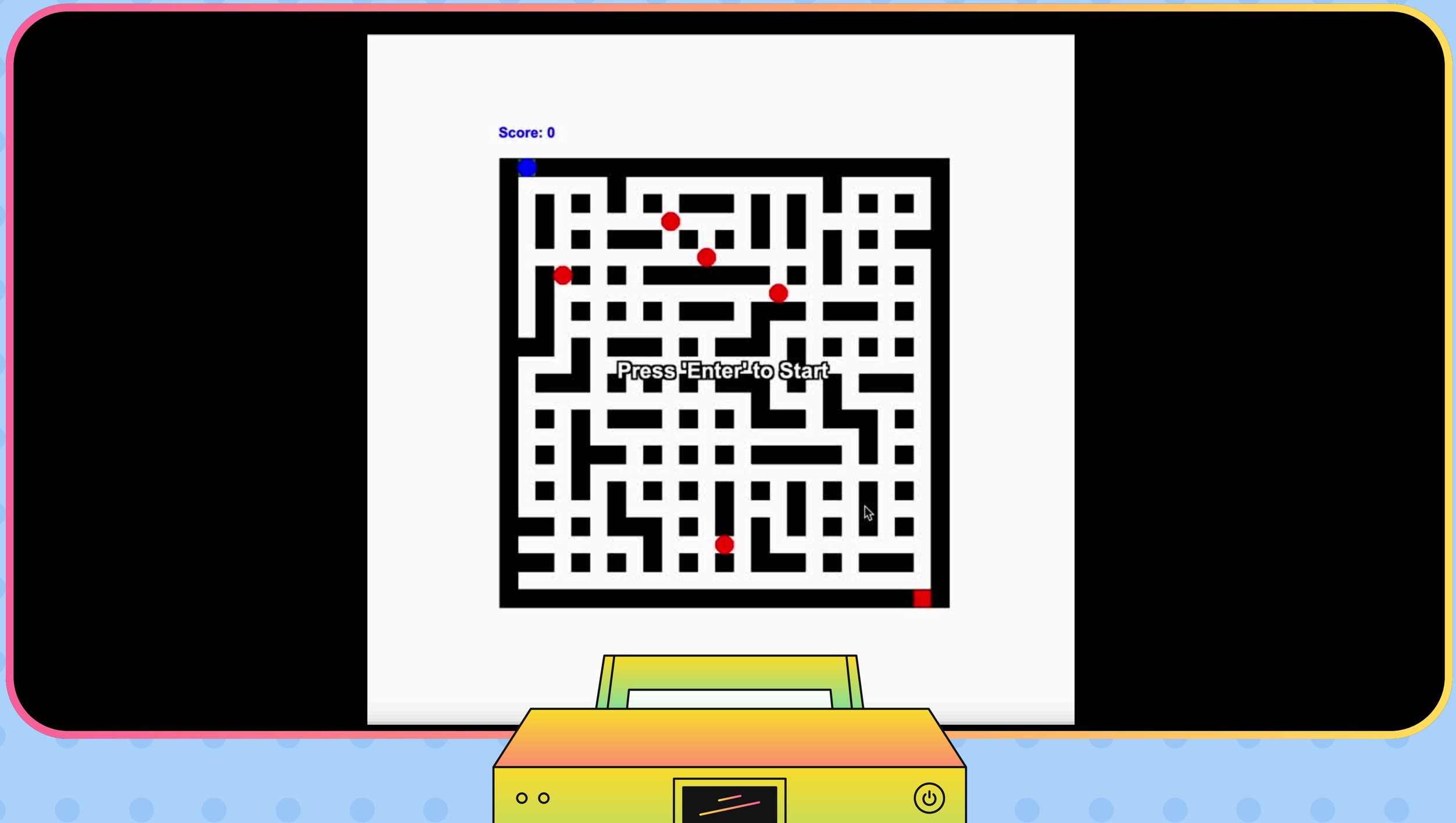
```python
# 플레이어가 종료점에 도달했는지 확인
def check_win():
    global game_running
    x, y = player.pos()
    grid_x = int((x + maze_size * 10) // 20)
    grid_y = int((maze_size * 10 - y) // 20)
    if maze[grid_y][grid_x] == 'E':
        game_running = False
        print("미로 탈출 성공!")
        turtle.bye()

# 키보드 이벤트 설정 함수
def key_ctrl():
    screen.onkey(lambda: move(0, -1), "w")
    screen.onkey(lambda: move(0, 1), "s")
    screen.onkey(lambda: move(-1, 0), "a")
    screen.onkey(lambda: move(1, 0), "d")
    screen.onkey(lambda: move(0, -1), "Up")
    screen.onkey(lambda: move(0, 1), "Down")
    screen.onkey(lambda: move(-1, 0), "Left")
    screen.onkey(lambda: move(1, 0), "Right")
```



기타 함수

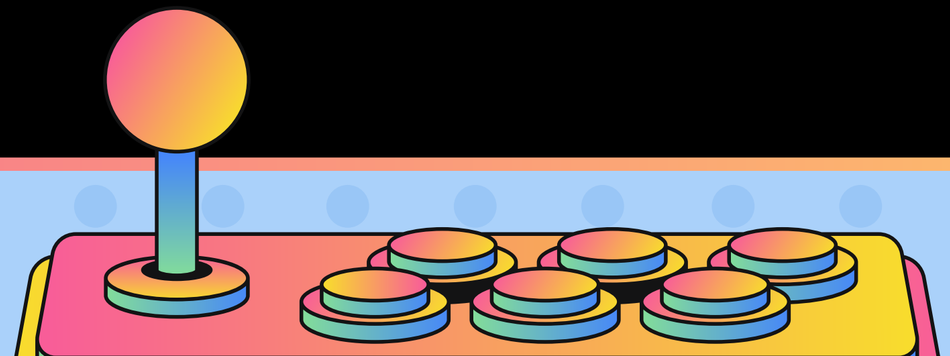
```
screen.listen()
screen.onkey(start_game, "Return")  # Enter 키를 눌러 게임 시작

# 시작 화면 표시
display_text_with_border(0, 0, "Press 'Enter' to Start")


# 메인 루프 실행
turtle.mainloop()
```

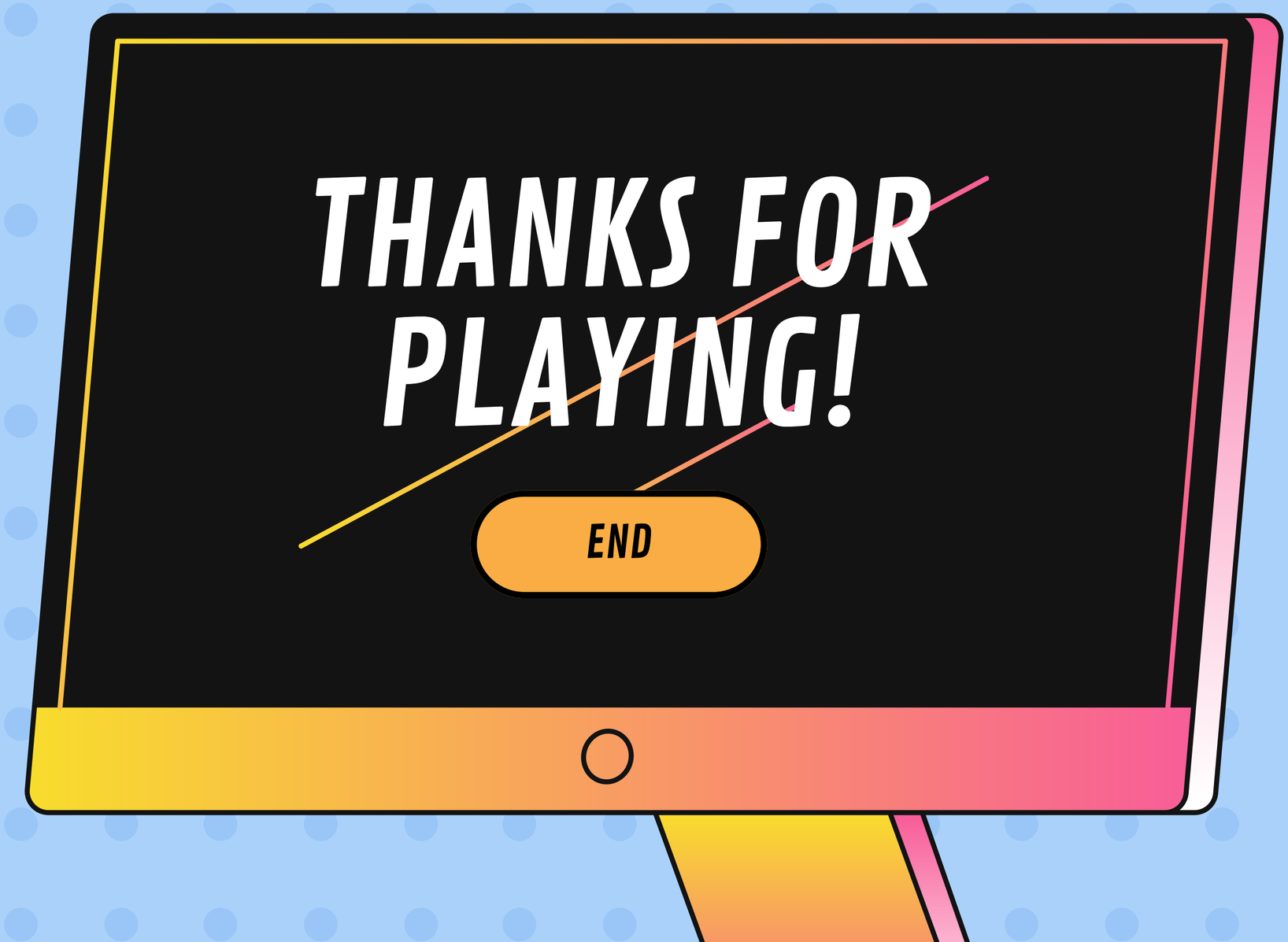



Score: 0
Press 'Enter' to Start

소감

이 과제를 통해 복잡한 문제를 단계적으로 해결하는 알고리즘을 설계하고,
그 과정에서 실질적인 게임 콘텐츠로 구현하는 경험을 얻을 수 있었습니다.
미로 탐색, 최단 경로 찾기, 적의 이동 및 플레이어와의 상호작용 같은 요소를
알고리즘으로 구체화하면서 문제 해결 능력을, 마주친 여러 문제를 팀원과 해결하며
유연한 사고와 협동성을 기를 수 있었습니다.




THANKS FOR PLAYING!
END