2024年 시스템프로그래밍

Team Project

202111040 김 지 현
202111057 문 서 흔
202111179 주 성 민
202211025 최 은 서

담당교수 : 김 예 성

DGIST 융복합대학기초학부
GOAL

2024 System Programming : Strategy Report

202111040 김지현, 202111057 문서흔,
202111179 주성민, 202211205 최은서

QR & Line tracking

Line tracking의 line 인식 후 처리 시간에 비해, QR 인식 및 처리 시간이 오래 걸려 두 기능을 동시에 실행했을 때 line tracking의 주기가 길어지면서 line을 자주 벗어나는 모습을 보였다. 따라서 line tracking의 안정성을 높이고 효율적인 QR 인식 시스템을 구축하기 위해 4초마다 라즈봇을 멈춰 세워 QR 스캔을 진행한 뒤, 다시 line tracking을 하도록 구현하였다.

How to input trap

상대방의 움직임이 알고리즘에 따라 매우 다양해 예측이 어려운 만큼, 상대에 따라 상이한 전략을 사용하는 것이 아닌 trap을 설치할 target location을 고정하는 전략을 선택하였다. 이에 알고리즘대로 움직이면서 처음에 방문하는 4개의 지점에 연속적으로 trap을 설치하도록 구현하였다.

Algorithm: Greedy Algorithm

실시간으로 위치마다의 item 정보가 업데이트 되기 때문에, 최대한 빠른 시간 내에 이동하는 것이 중요하다고 생각하였다. 이에 목표 지점을 따로 설정하지 않고, 선택의 순간마다 당장 눈앞에 보이는 최적의 상황만을 쫓아 최대한 많은 점수를 얻고자 greedy algorithm을 사용하였다. 이는 선택이 이루어지는 단계 상호 의존성이 적다고 판단하였기에 적절할 것으로 판단하였다.

트랩을 최대한 피하기 위하여 앞서 설명하였던 greedy algorithm을 응용하여 trap이 있는 노드를 방문하지 않도록 구현하였다. 만약 이동할 있는 한 칸 이내의 인접한 노드에 모두 trap이 있는 경우, 더 이상의 점수 손실을 막기 위하여 정지한다.

[프로세스]

- 가장 인접한 노드들의 Item/Trap 여부 조사 -> Item이 있는 경우 비교를 통해 가장 큰 점수를 지닌 item이 있는 쪽으로 이동
- Trap 이 있는 노드는 무조건 피하며 이동 진행
- 이동할 수 있는 모든 경로에 Trap이 있는 경우, random으로 한 Trap을 밟음

```
bool findSafeMove(int* targetRow, int* targetCol, Node board[MAP_ROW][MAP_COL]) {
    int directions[4][2] = { {0, 1}, {1, 0}, {0, -1}, {-1, 0} }; // 동, 남, 서, 북
    for (int i = 0; i < 4; i++) {
        int newRow = player_me.row + directions[i][0];
        int newCol = player_me.col + directions[i][1];
        if (newRow >= 0 && newRow < MAP_ROW && newCol >= 0 && newCol < MAP_COL && !isTrap(newRow, newCol, board)) {
            *targetRow = newRow;
            *targetCol = newCol;
            return true;
        }
    }
    return false;
}
```

그림1. 현재 위치의 상하좌우 중 이동 가능하며, trap이 없는 위치를 찾아 이동가능한 경로를 확보

```
void algorithm(Node board[MAP_ROW][MAP_COL]) {
    int targetRow = -1, targetCol = -1;
    bool moveToItem = false;

    // 먼저 한 칸 이내에 아이템이 있는 곳을 찾음
    int directions[4][2] = { {0, 1}, {1, 0}, {0, -1}, {-1, 0} }; // 동, 남, 서, 북
    for (int i = 0; i < 4; i++) {
        int newRow = player_me.row + directions[i][0];
        int newCol = player_me.col + directions[i][1];
        if (newRow >= 0 && newRow < MAP_ROW && newCol >= 0 && newCol < MAP_COL && hasItem(newRow, newCol, board)) {
            targetRow = newRow;
            targetCol = newCol;
            moveToItem = true;
            break;
        }
    }

    // 한 칸 이내에 아이템이 없으면 트랩이 없는 안전한 곳을 찾음
    if (!moveToItem && !findSafeMove(&targetRow, &targetCol, board)) {
        // 이동할 수 있는 모든 곳에 트랩이 있는 경우 정지
        next_action = -1;
        return;
    }

    // 안전한 경로로 이동
    safeDrive(targetRow, targetCol);
}
```

그림2. 구상한 Greedy Algorithm Code

서버로부터 불러온 정보를 바탕으로 구현한 알고리즘 대로 이동하도록 하려고 하였으나. Line tracking, QR인식, 서버통신 모두를 연결하는 과정에서 서버에서 불러온 정보를 바탕으로 로봇을 움직이는 부분까지 완성하지는 못하여 실제 상황에서 위의 알고리즘 대로 움직이도록 완벽하게 구현하지는 못하였다.