# 2024 System Programming: Strategy Report

202111040 김지현, 202111057 문서흔,
202111179 주성민, 202211205 최은서

## How to input trap

상대방의 움직임이 알고리즘에 따라 매우 다양해 예측이 어려운 만큼, 상대에 따라 상이한 전략을 사용하는 게 아닌 trap을 설치할 target location을 고정하는 전략을 선택한다. 시작위치에서 인접한 다음 column에 연속적으로 트랩을 설치한다. 상대방이 트랩을 심는 시간을 고려하여 트랩을 밟지 않고 최대한 빠른 시간 내에 트랩을 설치하고자 함. 이는 트랩을 설치하는 동안은 상대방의 트랩을 밟을 확률이 가장 적은 방법이라고 판단하여 해당 전략을 선택하였다.

- (0, 0) 시작한 경우, (0, 1), (1, 1), (2, 1), (3, 1)에 설치
- (4, 4) 시작한 경우 (4, 3), (3, 3), (2, 3), (1, 3)에 설치

```
200     bool findSafeMove(int* targetRow, int* targetCol, Node board[MAP_ROW][MAP_COL]) {
201         int directions[4][2] = { {0, 1}, {1, 0}, {0, -1}, {-1, 0} }; // 동, 남, 서, 북
202         for (int i = 0; i < 4; i++) {
203             int newRow = player_me.row + directions[i][0];
204             int newCol = player_me.col + directions[i][1];
205             if (newRow >= 0 && newRow < MAP_ROW && newCol >= 0 && newCol < MAP_COL && !isTrap(newRow, newCol, board)) {
206                 *targetRow = newRow;
207                 *targetCol = newCol;
208                 return true;
209             }
210         }
211         return false;
212     }
```

## Algorithm: Greedy Algorithm

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

트랩을 최대한 피하기 위하여 앞서 설명하였던 greedy algorithm을 응용하여 trap이 있는 노드를 방문하지 않도록 구현. 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 피할 것. Trap이 있는 경우, 이동할 수 있는 모든 노드들에 trap이 있는 경우 stop. 이를 주어진 시간 내에서 무한으로 반복 진행.

```
219     void algorithm(Node board[MAP_ROW][MAP_COL]) {
220         int targetRow = -1, targetCol = -1;
221         bool moveToItem = false;
222         int directions[4][2] = { {0, 1}, {1, 0}, {0, -1}, {-1, 0} }; // 동, 남, 서, 북 // 먼저 한 칸 이내에 아이템이 있는 곳을 찾음
223         for (int i = 0; i < 4; i++) {
224             int newRow = player_me.row + directions[i][0];
225             int newCol = player_me.col + directions[i][1];
226             if (newRow >= 0 && newRow < MAP_ROW && newCol >= 0 && newCol < MAP_COL && hasItem(newRow, newCol, board)) {
227                 targetRow = newRow;
228                 targetCol = newCol;
229                 moveToItem = true;
230                 break;
231             }
232         }
233         if (!moveToItem && !findSafeMove(&targetRow, &targetCol, board)) { // 한 칸 이내에 아이템이 없으면 트랩이 없는 안전한 곳을 찾음
234             // 이동할 수 있는 모든 곳에 트랩이 있는 경우 정지
235             next_action = -1;
236             return;
237         }
238         safeDrive(targetRow, targetCol);  // 안전한 경로로 이동
```