# PathFinding Method 연구

최원진 [1], 구본우 [1]
[1] 경민대학교 게임콘텐츠과
Dokebe1234@kyungmin.ac.kr, griogrio@ kyungmin.ac.kr

# PathFind Method Research

Won-Jin Choi[1], Bon-Woo Gu[1]
[1]Department of GameContents, KyungMin University

**요 약**

게임에서는 장애물이 가로 막고 있을 때 길 찾기 알고리즘이 요구된다. Path Finding Method 는 길과 장애물을 고려하여 목적지까지의 경로를 찾는 방법을 말한다. A* 알고리즘은 이런 복잡한 미로 찾기에 최적화된 Path Finding 알고리즘이다. 하지만, 모바일 같은 저비용 기기에서 A* 알고리즘을 사용하기엔 단순한 지형에서도 연산 부하가 발생할 수 있다. 본 논문에서는 가상의 공간에서 Grid 를 구축하여, 통행이 가능한 곳과 불가능한 곳을 나누어 최종 지점에 도달할 수 있도록 하는 방식을 제안한다. 본 논문에서 제시한 Path Finding Method 는 최종 지점이 막다른 길인 경우 가장 가까운 이동 가능한 경로로 길을 안내하도록 설계하여 예외 상황에 대처했다. 대표적인 길 찾기 알고리즘인 Dijkstra 알고리즘은 최소 비용을 고려해서 최단으로 가는 거리를 비교하여 길을 나타낼 수 있다. 하지만, Dijkstra 알고리즘 경우 비용이 양수가 아닌 음수의 경우 무한 루프에 빠지는 등 결과 값이 제대로 나오지 않을 수 있다. 본 논문에서 제안한 Path Finding Method 는 최소 비용을 노드별로 비교하는 방식이 아닌 초기 비용을 알 수 없는 분야에 쉽게 사용할 수 있다. 본 논문에서는 제안한 Path Finding Method 를 적용하여 Web 게임을 제작하는 것에 성공하였다. 향후, Path Finding Method 결과에 위치 정렬 알고리즘을 적용하여, 중복된 지역을 가는 확률을 최소화하면서 정리된 Path 가 돌출되도록 연구할 예정이다. 본 논문의 Path Finding Method 은 게임 개발 분야에 적극 기여되길 바란다.

## 1. 서론

게임에서는 구조물로 이루어진 미로를 탐색해야 한다. Map 의 복잡도에 따라 미로 탐색에 필요한 시간 복잡도와 공간 복잡도는 기하급수로 높아질 수밖에 없다.[1] A* 알고리즘은 이런 복잡한 미로 찾기에 최적화된 Path Finding 알고리즘이다.[2][3] 하지만, 모바일 같은 저비용 기기에서는 간단한 구조물 환경만으로도 A* 알고리즘을 사용하기엔 연산 부하가 발생할 수 있다.[4][5]

본 논문에서는 간단한 구조물 환경에서 사용할 수 있는 저비용(Low-cost) Path Finding Method 을 제안한다. 본 논문에서 제안된 Path Finding Method 은 모바일 저비용 기기에서도 연산 부하 없이 사용이 가능하다. 본 논문의 Path Finding Method 은 게임 개발 분야에 적극 기여되길 바란다.

## 2. Low-Cost Path Finding Method

본 논문에서 제안한 Path Finding Method 는 기본 입력으로 시작점과 목표 지점을 입력 받는다.

표 1 은 마우스 지정한 길을 찾고자 할 때 도출해내는 방법이다. *PlayerPos* 는 캐릭터의 위치를 의미한다. *MousePos* 는 마우스의 위치를 의미한다. *start* 는 *PlayerPos* 의 위치를 *GetEnalbePath* 를 통해 Grid 상의 위치로 나타낸다. *goal* 은 *MousePos* 의 위치를 *GetEnalbePath* 를 통해 Grid 상의 위치로 나타낸다.

이 때 가상의 좌표로 재선언해주며 지나갈 수 있는 길(Visit)을 탐색한다.

*Patharr* 는 *onPathSet* 을 통해 시작점과 목표 지점이 온전히 이어졌을 때 시작점과 중간 지점, 목표 지점 순으로 대입해준다.

수도 코드 1: Path Click

**INPUT:** $PlayerPos, GoalPos$
**OUTPUT:** $Patharr$

**SET** $start \leftarrow GetEnalbePath(PlayerPos)$
**SET** $goal \leftarrow GetEnalbePath(GoalPos)$

**SET** $Visit \leftarrow \{0\}$
**SET** $Patharr \leftarrow \emptyset$

**SET** $Visit[start.y][start.x] \leftarrow 1$
**SET** $Visit[goal.y][goal.x] \leftarrow 1$

**SET** $Patharr \leftarrow Patharr \cup start$
**CALL** $onPathSet(start, goal, Patharr)$
**SET** $Patharr \leftarrow Patharr \cup goal$

<표 1> Path Click

수도 코드 2: PathSet

**INPUT:** $start, goal, Patharr$
**OUTPUT:** $Patharr$

**SET** $len \leftarrow Length(start - goal)$
**SET** $center \leftarrow (start + goal)\ /\ 2$

**if** $len \leq 1.5$ **then**:
  **return**
**end if**

**if** $Pos[center.y][center.x]$ **is** 0 **then**:
  **SET** $center \leftarrow GetEnalbePath(center, start, goal)$
**end if**

**if** $center$ **is** $goal$ **then**:
  **return**
**end if**

**SET** $Visit[center.y][center.x] \leftarrow 1$
**CALL** $onPathSet(start, center\ , Patharr)$
**SET** $Patharr \leftarrow Patharr \cup center$
**CALL** $onPathSet(center\ , goal, Patharr)$

<표 2> PathSet

표 2 는 2 개의 지점을 인자로 받아와 두 지점 사이 거리가 점점 좁혀져 한 지점에 가까워졌을 때 끝나는 방식이다. 그전 까지는 재귀 함수를 통해 원하는 결과 값이 나오도록 재차 반복 실행시켜준다.

만약 길을 찾는 도중 막다른 길에 다다랐을 때 “GetEnalbePath”를 통해 다른 길을 도출해낸다.

표 3 의 경우 좌표가 못 지나가는 지점을 도달했을 때 해당 지점에서 가장 인접한 이동 가능 결로를 새로 지정해준다.

수도 코드 3: GetEnablePath

**INPUT:** $center, start, goal$
**OUTPUT:** $center$

**SET** $dis \leftarrow \{\infty\}$

**for** $y := 0$ *to* $H$ **do**:
  **for** $x := 0$ *to* $W$ **do**:

    **if** $Pos[y][x]$ **is** 1 **and** $Visit[y][x]$ **is** 0
      $dis[x][y] \leftarrow Length([x, y] - center)$
    **end if**

    **if** $[x, y]$ ***is*** $start$ **or** $[x, y]$ **is** $goal$
      $dis[x][y] \leftarrow \infty$
    **end if**

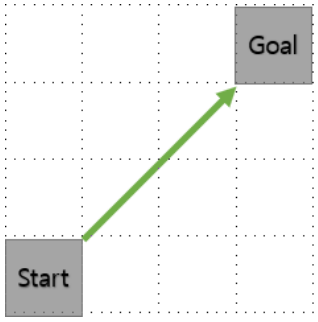
  **end for**

**end for**

**return** $argmin_{x,y}(dis)$

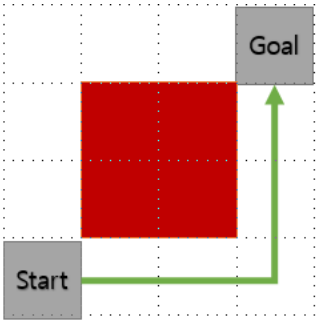
<표 3> Get Enable Path

### 3. Low-Cost Path Finding Method 실험 결과



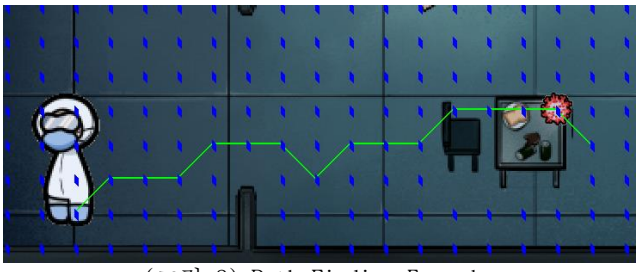

(그림 1) None Obstruction

그림 1 과 그림 2 는 Path Finding Method 을 이용하여 제작한 길 찾기 화면을 보여준다. 본 실험에서 길을 막고 있는 장애물을 피해서 올바른 경로를 보여준다.

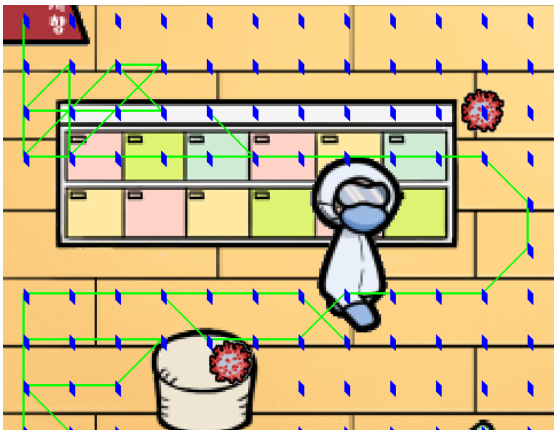



(그림 2) Obstruction

그림 3 는 Python 을 기반으로 제작한 Path Finding Method 을 실제 게임에 적용한 화면을 보여준다. 본 실험에서 지나갈 수 있는 지형과 이동경로의 시각화를 보여준다.



(그림 3) Path Finding Example



(그림 4) Overlapping Regions

그림 4 에서처럼 구조물들로 둘러 쌓여 있을 때 Path Finding Method 에서 중복된 지역을 가는 경우가 생긴다. 이는 맵의 복잡도가 낮을 경우, 중복된 지역을 가는 확률이 낮아지지만, 맵의 복잡도가 높을 경우, 더 많은 경우의 수를 찾아야 하므로, 중복된 지역을 가는 확률이 높아진다.

## 4. 결론

본 논문에서 저비용 기기에서도 사용할 수 있는 Low-Cost Path Finding Method 를 제안하였으며, 길 찾기와 경로 찾기를 이용하여 시작점에서 목표 지점까지의 경로를 얻는 실험을 진행하였다. Low-cost Path Finding Method 는 시작점과 목표 지점 사이의 장애물이 있다면, 가장 가까운 갈 수 있는 지점을 빠르게 찾는다. 이 단계를 노드 사이 거리가 1.5PX 이하가 될 때까지 찾는 것을 재귀로 반복한다. 1.5PX 보다 더 큰 범위를 적용한다면, Path Finding 결과는 단순하며 물체를 통과할 수도 있다. 반면, 1.5PX 보다 작다면, Path Finding 결과는 매우 부드러우며 물체의 경계를 부드럽게 지나가는 경로가 생성된다. 본 실험은 Python 을 기반으로 제작하였고, 모바일 게임에 접목시켜 보았다. 향후, Path Finding Method 결과에 위치 정렬 알고리즘을 적용하여, 중복된 지역을 가는 확률을 최소화하면서 정리된 Path 가 돌출되도록 연구할 예정이다. 본 논문은 게임 분야에 기여하길 바란다.

## 참고문헌

[1] 김윤성 외, “임의형태의 장애물 경계정보를 이용한 최소거리 우회경로 탐색 알고리즘”, 한국시뮬레이션 학회 논문지, vol.19, no.4, pp.129-137, 2010

[2] Loong ET, “A star path following mobile robot.”, 2011 4th International conference on mechatronics, Vancouver, 2011, pp.1-7

[3] Cui ET, “A*-based pathfinding in modern computer games”, International Journal of Computer Science and Network Securiy, 11, 1, pp.125-130, 2011

[4] Bryan Stout, “The Basics of A* for Path Planning”, Game Programming Gems, 1, pp. 254-263, 2000

[5] 정병두 외, “A* 알고리즘 평가함수의 추정 부하량 변경에 관한 연구”, 한국 ITS 학회 논문지, vol.14, no.3, pp.1-8, 2015