2003년 한국산학기술학회 춘계 학술발표논문집

미로 자동생성 알고리즘 기법

이은아 · 박용범

단국대학교 대학원 전자계산학과

The technique of Automatic Generation Algorithm of Maze

Eun Ah Lee and Young B. Park

Computer Science, Dankook University

요 약

기존에 만들어진 미로 생성 알고리즘은 'Perfect 미로'를 지향하는 알고리즘이다. Perfect 미로는 두 점사이의 path가 유일하며, 순환되지 않고, 막힌공간이나, 접근할 수 없는 공간이 없는 미로를 말한다. 이런 미로 알고리즘을 사용하여 만든 미로는, 정형적인 형태를 띄게 된다. 이 알고리즘들을 아무런 수정 없이, RPG/ 액션/ 시뮬레이션/ 전략게임 등에 사용하게 되면, 게임 캐릭터는, 유일하면서 막힘이 빈번한 path 안에서 부자연스런 움직임을 가지게 될 것이다. 그래서 보다 넓은 면적을 탐험할 수 있고, 전진성을 높여주기 위하여 여러개의 path와 순환을 허용하는 알고리즘을 제안하게 되었다.

1. 서 론

캐릭터나 액션등과 함께 게임의 중요한 요소 중 하나는 게임이 플레이되는 우주-배경이다[6]. 그러나 그러한 배경이 게임이 진행되어도 새롭게 생성되지 않는다면 캐릭터의 활약이 아무리 뛰어나도 게이머의 흥미를 끌 수는 없다.

배경에서 texture나 graphic을 제거하면 '미로(Maze)'가 된다. 그래서 미로 자동생성 알고리즘은 게임제작에 있어서 간과할 수 없는 부분이 되었다[2].

이러한 배경 생성에 적합한 알고리즘은 다음과 같은 특질을 만족하여야 한다.

첫째 숨겨진 이벤트나 더 많은 아이템의 획득을 위해서 캐릭터가 가능한 한 넓은 면적을 탐색해야 한다.

둘째 미로 상에서 두 점사이의 path가 꼭 유일할 필요가 없기 때문에 순환이 허용되어야 한다. 캐릭터의 관점에서 배경은 움직임을 위한 현실공간이며 또한 게임의 생명은 생생한 움직임이다[5]. 그래서 RPG/ 액션/ 시뮬레이션/ 전략게임 등에 사용될 미로 생성 알고리즘은 위 조건을 만족하여야 한다.

순환이 허용되어 캐릭터가 자유롭게 이동할 수 있다면 미로를 넓게 탐험하는 것이 가능하기 때문에 본 논문에서는 두 번째 조건에 중점을 두어서 RPG/ 액션/ 시뮬레이션/ 전략게임 등에 사용될 순환 허용이 가능한 자동 미로생성 알고리즘을 제안한다.

2. 기존 미로생성 알고리즘과 특징

기존 미로 생성 알고리즘은 대부분 Perfect 미로 생성을 목적으로 하며 여러 가지 기법이 존재한다.

먼저 그 특징을 알아보고 그 종류를 살펴본다.

2.1 기존 미로생성 알고리즘들의 특징

기존 미로생성 알고리즘은 공통적으로 perfect 미로를 생성한다. perfect 미로는 다음의 네 가지 조건으로 정의할 수 있다[7].

① No circular path
- path의 순환을 허용하지 않는다.

② No open area
- n개의 벽이 완전한 공간을 허용하지 않는다.

③ No inaccessible area
- 장방형의 open area를 허용하지 않는다.

④ Unique path between two points
- 미로상의 두 점 사이의 path는 유일하다.

여기에 해당되는 알고리즘으로 Depth-First Search 알고리즘, Prim's 알고리즘, Kruskal's 알고리즘, Aldus-Broder 알고리즘 등이 있다.

또한 대부분이 별도의 저장공간을 필요로 한다. 복귀 Cell을 저장할 Stack을 위한 저장공간, 어떤 Cell의 미로 생성 여부, 또는 그 Cell과 이웃한 Cell인지 여부를 가리기 위한 저장공간, 무작위로 선택된 미로의 Cell들 사이의 각 간선이나 벽을 열거할 저장공간 등을 위하여 저장공간이 사용된다[1]. 이러한 별도의 저장공간은 미로 생성시 시스템에 약간의 부담을 주게되므로 지양해야 할 것이다.

2.2 기존의 미로생성 알고리즘

2.2.1 Depth-First Search 알고리즘

```
CellStack(LIFO) 생성  // Cell 위치 저장용
set TotalCell                    // 미로의 총 셀 수
CurrentCell 선택              // at random
VisitedCells - 1
while VisitedCells < TotalCells
    CurrentCell의 이웃 Cell 찾기
    if one or more found
        Cell 선택                 // at random
        Cell과 CurrentCell 사이의 벽 제거
        VisitedCells += 1
    else
        CellStack에서 newCell을 pop한다
        CurrentCell - newCell
    endif
endwhile
```

Algorithm-1 : Depth First Search 알고리즘

Depth First Search 알고리즘은 전형적인 perfect 미로를 생성한다 (Algorithm-1 참조)[8].
방문된 Cell수를 나타내는 VisitedCell을 Total- Cell만큼 증가시켜야 알고리즘이 종료되므로 결코 open area가 생성되지 않는다. 또한, 알고리즘의 특성상 각 Cell 최소한 한번의 방문을 받거나, 하게 되므로 어느 한쪽은 반드시 뚫려있게 된다. 이것은 장방형의 dead-end는 생길 수 있지만 직사각형의 inaccessible area는 생길 수 없음을 말해준다. 또한 곳곳에 dead- ends를 만들기 때문에 path의 순환도 허용되지 않는다.

2.2.2 Prim's 알고리즘

```
set TotalCell                    // 미로의 총 셀 수
In, Frontier, Out생성   // TotalCell 크기의 Vector
Out = 모든 Cell로 초기화
startingCell 선택          // at random
CurrentCell = startingCell
In = CurrentCell
Frontier = startingCell의 이웃 Cell
while Frontier에 원소가 있을 때
   Cell을 Frontier에서 In으로 옮긴다 // at random
   CurrentCell = Cell
   CurrentCell의 이웃을 Out에서 찾는다
   if one or more found
      CurrentCell의 이웃 Cell을 Out에서 지우고
      Frontier에 옮긴다.
      CurrentCell의 모든 이웃을 In에서 찾는다.
      if one or more found
         Cell을 선택한다// at random
         CurrentCell과 선택된 Cell사이 벽 제거
      endif
   endif
endwhile
```

Algorithm-2 : Prim's 알고리즘

Prim's 알고리즘은, 미로가 생성되는 동안 각 Cell은 다음중 한가지 형태이다 (Algorithm-2 참조)[3][8].

1) "In" : 미로의 한 부분인 셀들이 들어 있으며, 이미 통로가 새겨져 있다.
2) "Frontier" : 아직 미로의 한 부분이 아닌 셀들이고 통로가 새겨지지 않았다. 그러나 이미 "In"에 들어있는 셀의 다음(이웃) 셀들이다.
3) "Out" : 아직 미로의 한 부분이 아닌 셀들의 집합이다. 그리고 그 셀들의 이웃은 "In"에 들어있지 않다.

Depth-First Search 알고리즘처럼, 별도의 저장공간을 사용하며, perfect 미로를 생성한다.

이 알고리즘은 Frontier에 아무 Cell도 남아있지 않을 때 종료된다.

2.2.3 Aldus-Broder 알고리즘

```
각 셀의 최대 벽의 개수 = n
TotalCell                    // 미로의 총 셀수
CurrentCell = startingCell   // at random
newCell = CurrentCell의 이웃 Cell // at random
VisitedCell = 1
while VisitedCell < TotalCell
    if newCell이 n개의 벽을 가지고 있으면
        newCell과 CurrentCell 사이의 벽 제거
        VisitedCell += 1
        CurrentCell = newCell
        newCell = CurrentCell의 이웃 Cell
    else
        CurrentCell = newCell
        newCell = CurrentCell의 이웃 Cell
    endif
endwhile
```

Algorithm-3 : Aldus-Broder 알고리즘

Aldus-Broder 알고리즘은 여분의 저장공간이나 스택을 필요로 하지 않는다. 한 셀을 선택하고 무작위로 선택된 이웃 셀로 이동한다. 만약 통로가 열리지 않은 셀에 들어가게 되면 이전의 셀과 조각된다. 모든 셀이 통로가 조각될 때까지 이웃 셀로의 이동을 계속한다 (Algorithm-3 참조)[7].

2.2.4 Kruskal's 알고리즘

```
Initialize the union-find structures
Create set Walls
                // containing all the interior walls
Set COUNT    // number of cells
while COUNT>1
   Remove a wall from the Walls  //at random
   if (the two cells already connected)
                // wall separates the two cells
      do nothing
   else
      connect the two cells
   endif
endwhile
```

Algorithm-4 : Kruskal's 알고리즘

Kruskal's 알고리즘은 랜덤하게 선택된 미로의 셀들 사이의 각 간선이나 벽을 열거하고, 미로의 크기에 비례하는 저장 공간이 있어야 한다. 각 셀에 유일한 id를 붙이고 랜덤하게 순서된 모든 간선을 순환시킨다 (Algorithm-3 참조)[4][8].

각 간선에서 만약 그것의 양쪽 중 한쪽의 셀이 다른 id를 가지고 있다면 벽을 지운다. 그리고 다른 쪽처럼 같은 id를 가진 한쪽의 모든 셀을 Set한다. 만약 벽의 양쪽 중 한쪽의 셀이 이미 같은 id를 가지고 있다면 이미 두 셀 사이에 어떤 경로가 존재한다는 것이다.

3 제안하는 미로생성 알고리즘; Circular-path 알고리즘

```
1   각 셀의 최대 벽의 개수 = n
2   TotalCell                    // 미로의 총 셀수
3   count = 1
4   startCell                    // at random
5   CurrentCell = startCell

6   while count < TotalCell
7       newCell로 이동 // CurrentCell이 이웃셀
8       if newCell의 벽이 n개
9           CurrentCell과 newCell 사이 벽 제거
10          count += 1
11      else if newCell의 벽이 n-1개
12          CurrentCell과 newCell사이 벽제거
13      endif
14  endwhile
```

Algorithm-5 : Circular path 알고리즘

기존의 알고리즘은 공통적으로 perfect 미로를 생성한다. 하나의 게임 안에서도 배경생성에 사용되는 알고리즘은 다양할 필요성이 있다. 캐릭터가 Dungeon 등에서 활약하는 게임이라면 기존의 알고리즘만으로도 충분할 것이다. Dungeon에서는 여러 개의 dead-ends에 막히더라도 실제 세계의 지형과 견주어 부자연스러울 것이 없기 때문이다. 그러나 캐릭터가 움직이는 곳이 ground 또는 성곽주변이라면 dead-ends는 존재하되, 순환 역시 허용하여 자연스러운 움직임을 유도할 필요가 있을 것이다.

알고리즘은 각 셀을 첫 방문한 개수인 count가 TotalCell수와 같아질 때 종료된다. 또한 별도의 저장공간을 필요로하지 않기 때문에, 미로 생성시 시스템의 부담을 덜어준다.

또한, 이 알고리즘의 특징은 순환을 허용하는데 있다. open area와 inaccessible area를 생성하지 않는 조건은 perfect 미로와 같지만 순환을 허용함으로써 두 점사이의 유일한 path가 존재한다는 조건에서 벗어나게 된다.

이런 결과를 만드는 것이 바로 라인 11~12이다.

기존의 미로들이 공통적으로 현재 Cell과 n개의 벽이 완전하게 남아있는 Cell 사이를 조각하는 방법을 택하지만, 이 알고리즘에서는 n-1개의 벽을 가지고 있는 Cell과의 사이에 있는 벽도 제거할 수 있게 하였다. 물론 이 경우에는 count가 증가되지 않기 때문에 모든 셀을 방문해야 하는 규칙에는 변함이 없게된다. 벽의 개수를 n-1로 제한한 이유는 최소한의 dead-ends를 남겨두기 위해서이다.
(Algorithm-5 참조)

4. Circular-path 알고리즘의 구현 및 결과

Figure 1과 Figure 2는 Circular-path 제안한 알고리즘에서 11~12 라인을 제거하여 순환을 허용하지 않은 알고리즘과 제거하지 않아서 순환을 허용한 알고리즘을 각각 구현한 결과이다.

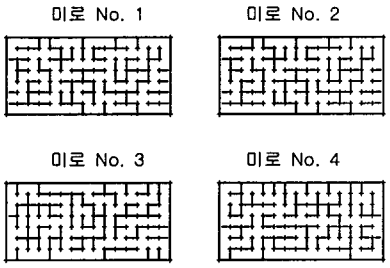
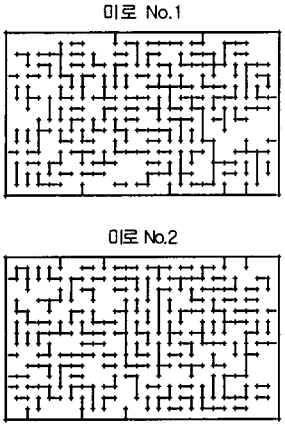


Figure 1. 순환이 허용되지 않는 Circular-path 알고리즘 구현 결과



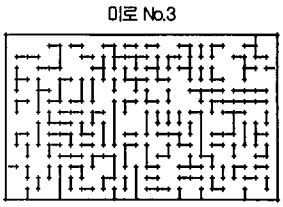


Figure 2. 순환이 허용되는 Circular-path 알고리즘 구현 결과

5. 결 론

이제까지 게임제작에 필요한 미로생성 알고리즘을 살펴보았다. 기존에 쓰이고 있는 여러 가지 알고리즘은 성능면에서 훌륭한 알고리즘임에 틀림이 없다. 다만 저자가 기존 알고리즘의 문제점이라고 짚은 것은 상황에 따른 적합성에 대한 문제일 뿐이지 결코 그 알고리즘들을 폄하하려는 의도는 갖고 있지 않다.

또한 보완을 위해 제안한 Circular-path 알고리즘은 앞에서 제시한 문제점들에 대한 보완일 뿐 결코 완전한 알고리즘이 아니다.

그래서 앞으로 해야 할 일은 Circular-path 알고리즘에 대하여 현실적으로, 또는 이론적으로 수정을 가하여 좀더 충실하고 타당한 알고리즘으로 거듭나게 해야할 것이다.

참고자료

[1] Ellis Horowitz외 (이석호 역), C로 쓴 자료구조론, 사이텍미디어 1993
[2] Mickey kawick(서형준 역), 실시간 전략 게임 프로그래밍, 정보문화사 2000
[3] Neopolitan외 (도경구 역), 알고리즘, 사이텍 미디어, 1999
[4] 이재규, C로 배우는 알고리즘, 세화 2001
[5] David M. Bourg (황혁기 역), 생생한 게임 개발에 꼭 필요한 기본 물리, 한빛미디어, 2002
[6] Lecky Thompson (장원석 역) 게임 개발 수학적 테크닉, 정보문화사, 2002
[7] http://Astrolog.org/labyrnth/algorithm.htm
[8] http://www.mazeworks.com/mazegen