한국지능시스템학회 논문지 2010, Vol. 20, No. 1, pp. 42-47

미로 탐색 알고리즘 테스트를 위한 플랫폼 개발

Platform Development for Maze Search Algorithms Testing

서효석*・박재민*・이상용**+

Hyo-Seok Seo*, Jae-Min Park* and Sang-Yong Lee**+

* 공주대학교 컴퓨터공학과

** 공주대학교 컴퓨터공학부

요 약

마이크로 마우스를 이용한 다수의 미로 경진대회가 개최되어 미로 탐색 알고리즘의 성능이 비교되고 있으며, 미로 탐색 알고리즘은 좌(우)수법, 구심법, 언덕오르기 등을 기본으로 하여 다양한 형태로 적용되어 사용되고 있다. 하지만 미로 탐색 알고리즘을 적용하여 테스트하기 위한 소프트웨어 플랫폼이 없어서 프로그램을 직접 개발하거나 하드웨어를 통해 알고리즘의 성능을 테스트해야 하는 불편함을 겪는다.

본 연구에서는 하드웨어로 구현이 어려운 다양한 형태의 미로 제작과 알고리즘의 손쉬운 적용이 가능하고, 스텝, 연산 횟수, 탐색 시간의 평가가 가능한 미로 탐색 알고리즘을 위한 플랫폼을 개발하였다. 플랫폼은 메인 레이어, 인터페이스 레이어, 사용자 레이어의 분리 구조로 되어 알고리즘을 쉽게 교체적용 할 수 있는 장점이 있다.

플랫폼의 실험을 통하여 미로 탐색 알고리즘들의 성능을 평가하고 분석하여 알고리즘의 개발 및 실험에도 적용할 수 있음을 확인하였다.

키워드 : 미로, 탐색, 테스트 플랫폼, 마이크로 마우스

Abstract

Many contests by micro mouse was celebrated of which maze search algorithms performance are compared. That is used in various forms based on left(right) weight method, euclidean algorithm method, hill climbing method. However we feel uncomfortable to test algorithms performance through direct development of programs or hardwares as no software platform to test in maze search algorithms.

In this research we develop of a platform for maze search algorithms that is easily to produce various forms of maze that are hard to be realized by hardware, to apply algorithms, and evaluate the seek time, operation count, steps and performance. The platform is consist of main layer, interface layer, user layer which has merit to apply and replace easily algorithms.

We verified that the maze search algorithm can be applied even in the development and experiment of algorithm by evaluating and analyzing its performance through the experiment of platform.

Key Words : Maze, Search, Test Platform, Micro Mouse

1. 서 론

마이크로 마우스를 이용한 다수의 미로 경진대회가 개최되어 탐색 알고리즘들의 성능이 평가되고 있으며, 최근에는 내비게이션, 웹 지도, 게임 등에서도 다양한 형태의 길을 찾는 탐색 알고리즘들이 사용되고 있다. 하지만 미로 탐색 알고리즘을 적용하고 테스트하기 위한 플랫폼이 없어서 성능 평가 프로그램을 직접 개발하거나 하드웨어를 통해 테스트해야 하는 불편함을 겪는다.

접수일자 : 2009년 11월 30일
완료일자 : 2010년 1월 29일
+ 교신저자
본 논문은 본 학회 2009년도 추계학술대회에서 선정된 우수논문입니다.

본 논문에서는 이러한 불편을 해소하기 위해서 미로 탐색 알고리즘 테스트를 위한 플랫폼을 설계하고 구현하였다. 플랫폼은 현실에서는 구현하기 어려운 장애물이나 미로의 출발점 및 도착점을 간단한 마우스 클릭만으로 자유롭게 설치할 수 있도록 도와주며, C++언어를 사용하여 알고리즘을 작성하거나 불러와 생성된 미로에 적용시킬 수 있다. 적용된 알고리즘은 경로 및 움직임을 애니메이션 효과로 확인할 수 있고 가중치가 적용된 탐색 시간, 연산 횟수, 스텝 등의 성능 비교 또한 가능하도록 제작되었다. 본 플랫폼에 기존에 알려진 다양한 미로 탐색 알고리즘을 적용하여 성능을 평가하고 분석함으로써, 미로 탐색 알고리즘을 연구하기 위한 통합 플랫폼으로써의 사용될 수 있을 것이다.

표 1. 미로탐색 알고리즘의 특징
Table 1. Features of maze searching algorithms

특징 / 알고리즘		일반적 미로 탐색				최단거리 경로 탐색		최단비용 경로 탐색	
		좌(우)수법	확장 좌(우)수법	구심법	확장 구심법	언덕오르기	A* 알고리즘	다익스트라 알고리즘	벨만-포드 알고리즘
출발점 사전 제공		필요	필요	필요	필요	필요	필요	필요	필요
도착점 사전 제공		불필요	불필요	필요	필요	필요	필요	필요	필요
벽 정보 사전 제공		불필요	불필요	불필요	불필요	필요	불필요	필요	불필요
탐색 가능성		▲	가능	가능	가능	가능	가능	가능	가능
최단경로 탐색 가능성		불가능	불가능	불가능	◆	가능	가능	●	가능
연산량		적음	적음	보통	매우 많음	보통	많음	매우 많음	매우 많음
장애물 인식	고정	○	○	○	○	○	○	○	○
	가변	○	○	×	○	×	×	○	○
경로 변환		가능	가능	불가능	가능	불가능	불가능	가능	가능

▲ 아일랜드 형 미로에서 탐색 불가 ◆ 여러 번 수행 시 가능 ● 음수 가중치가 있으면 불가능

2. 미로 탐색 알고리즘

미로 탐색 알고리즘은 출발점에서 도착점까지의 경로를 찾아내기 위한 방법을 말한다[1]. 미로 탐색 알고리즘은 일반적인 미로 탐색 방법, 최단 경로를 찾는 방법, 최소 비용을 찾는 방법으로 나뉜다.

일반적인 미로 탐색 방법은 좌(우)수법, 구심법이 있으며 이것들의 단점을 보완한 확장좌(우)수법, 확장구심법이 있다[1]. 일반적인 알고리즘들은 미로에 대해 많은 정보를 가지고 있지 않은 상황에서 적은 연산만으로도 미로의 특징을 파악 할 수 있고, 상황에 따라 경로를 변환하며 도착점을 찾아 갈 수 있다. 하지만 최단 경로를 찾기란 쉽지 않다.

최단 경로를 찾는 알고리즘에는 인공지능의 휴리스틱 탐색 기법을 사용한 언덕오르기와 A* 알고리즘 등이 있다[2][3][4]. 최단 경로 탐색 알고리즘들은 미로에 대한 사전 정보가 주어진 상황에서 평가함수를 사용하여 많은 연산을 통해 최단 경로를 찾는 것이 가능하지만, 한번 정해진 경로를 바꿀 수 없기 때문에 미로가 유동적으로 바뀌는 상황에서도 정해진 길만 따라가게 된다.

최소 비용 탐색 알고리즘인 다익스트라 알고리즘과 벨만-포드 알고리즘은 가중치가 있는 미로에서 최단 비용 경로를 찾는 알고리즘으로 미로에 대한 사전 정보가 주어진 상황에서 매우 많은 연산을 필요로 하지만, 가변적인 상황을 맞게 되면 계산한 경로를 수정하여 이동하는 것이 가능하기 때문에 유동적인 미로에서도 강한 모습을 보인다[5][6].

미로 탐색 알고리즘들은 표 1과 같이 최단 경로 탐색 가능성, 연산 량, 장애물 인식, 경로변환 가능성 등이 다르기 때문에 특정 알고리즘의 좋거나 나쁨을 쉽게 판단할 수 없으며, 미로의 특성에 따라 적용 방법과 구현 방법이 다르기 때문에 미로의 특성에 맞게 적절한 알고리즘을 선택하거나 혼합 사용하여 더 나은 알고리즘을 구현하는 연구가 진행 되고 있다.

3. 플랫폼 설계 및 구현

3.1 플랫폼 설계

3.1.1 플랫폼 구조

본 논문에서는 다양한 알고리즘을 적용하고 성능을 평가하기 위한 플랫폼을 설계하고 구현하였다.

미로 탐색 알고리즘을 연구하기 위해서 플랫폼을 직접 제작하거나 하드웨어로 성능을 테스트해야 하는 불편함을 해결하고, 실험에서 동일한 조건을 제공하기 위해 미로 탐색 알고리즘 테스트를 위한 플랫폼은 그림 1과 같이 메인 레이어, 인터페이스 레이어, 사용자 레이어의 3개 레이어 구조로 설계했다. 각 레이어의 분할을 통해 사용자는 쉽게 사용자 레이어에 찾아가 알고리즘을 교체하거나 적용할 수 있고 적용된 알고리즘이 메인 레이어의 미로 정보 모듈에 영향을 끼치는 것을 미연에 방지하는 것도 가능하게 되었다.

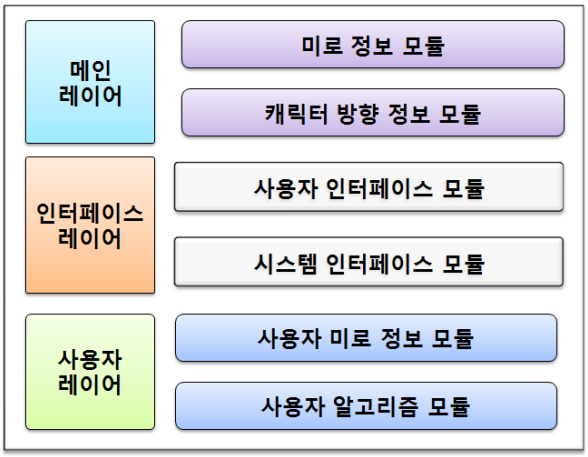


그림 1. 플랫폼 구조
Fig. 1. Structure of platform

3.1.2 플랫폼 처리 흐름

플랫폼 처리 흐름은 그림 2와 같다. 먼저 사용자가 미로, 캐릭터, 장애물을 생성하며, 출발점과 도착점이 하나씩 생성되어 있지 않을 경우 알고리즘은 실행되지 않는다. 미로생성이 끝나고 시작 혹은 알고리즘 호출이 실행되면 캐릭터의 현재 좌표를 사용자 알고리즘에 전송한다. 사용자 알고리즘 모듈은 미로 정보 모듈에서 제공된 좌표에 따라 도착점까지 이동하기 위한 다음 위치를 계산하여 리턴하고, 이 과정을 반복함으로써 캐릭터가 이동하며 도착점까지의 경로를 생

성한다. 최종적으로 캐릭터가 도착점에 도달하면 진행 시간과 스텝이 표시되어 알고리즘을 평가할 수 있다. 만약 캐릭터의 비정상적 이동 등이 감지 됐을 경우에는 경고 메시지 출력과 함께 알고리즘 실행은 종료된다. 이때 플랫폼을 종료시키거나 다시 초기화하여 진행할 수 있다.

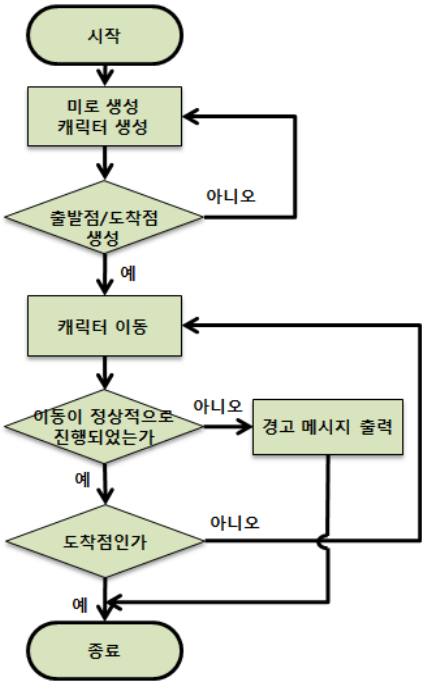


그림 2. 플랫폼 처리 흐름도
Fig. 2. Flow chart of platform processing

3.2 플랫폼 구현

3.2.1 메인 레이어

메인 레이어는 플랫폼의 기본 정보를 저장하고 제공하는 계층으로 플랫폼에서 만들어진 미로의 정보를 가지고 있다.

메인 레이어는 미로 정보 모듈과 캐릭터 방향 정보 모듈로 구성된다. 미로 정보 모듈은 실제 X, Y좌표와 타일의 속성 정보를 가지는 Map 클래스로 2차원 배열의 형태이며, 사용자 알고리즘 모듈에서의 접근을 막기 위해 미로 속성 정보는 상속의 형태로 사용자 미로 정보 모듈에 전달한다. 미로의 속성은 미로를 만들고 배치하는 과정에서만 수정이 가능하고, 알고리즘이 실행되면 수정이 불가능하다.

캐릭터 방향 정보 모듈은 캐릭터의 진행 방향에 대한 정보를 '1 = 북, 2 = 동, 3 = 남, 4 = 서'의 숫자형으로 저장한다. 각 방향은 캐릭터의 현재 위치와 이동할 위치를 비교하여 전환하고 인터페이스 레이어의 캐릭터 이동 제어 기능에 전달한다. 그림 3은 메인 레이어의 흐름을 나타낸다.

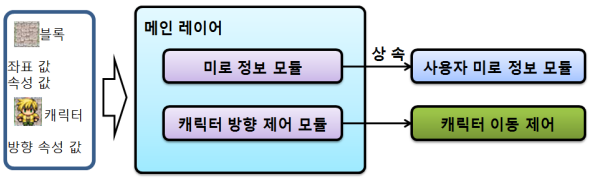


그림 3. 메인 레이어 흐름도
Fig. 3. Flow chart of main layer

3.2.2 인터페이스 레이어

인터페이스 레이어는 미로 제작과 편집 기능을 제공하는 사용자 인터페이스 모듈과 시스템에서 발생하는 이벤트를 처리하는 시스템 인터페이스 모듈로 구성된다.

사용자 인터페이스 모듈은 미로 생성/제어 기능, 캐릭터 생성/제어 기능, 장애물 생성/제어 기능 등이 있다. 미로 생성/제어 기능은 미로의 벽, 길, 도착점을 지정하고, 캐릭터 생성/제어 기능은 캐릭터의 출발점을 지정하고, 장애물 생성/제어 기능은 늪, 가변 문등의 장애물의 위치를 지정한다. 미로 및 캐릭터, 장애물의 생성은 마우스 클릭만으로 손쉽게 제어가 가능하며 좌 버튼을 클릭 할 때마다 '벽, 목적지, 문(장애물), 캐릭터, 늪(장애물), 길'의 순서로 타일이 변화한다. 각 미로에서 캐릭터와 도착지는 한 개씩 생성이 가능하다. 그림 4와 같이 사용자 인터페이스 모듈에서 생성된 미로 정보는 메인 레이어의 미로 정보 모듈에 저장된다. 표 2는 사용자 인터페이스 모듈의 일부 코드로, if 문에서 알고리즘의 시작이 확인된 경우 생성된 타일의 좌표와 속성 값을 미로 정보 모듈로 보내주는 코드이다.

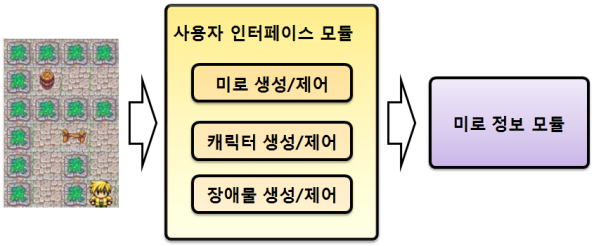


그림 4. 사용자 인터페이스 모듈
Fig. 4. User interface module

표 2. 사용자 인터페이스 모듈 소스
Table 2. User interface module source

```
                ⋮
Map[LOWORD(lParam)/tile][HIWORD(lParam)/tile].status++;
                ⋮
if(startcount == 1)
startcount--;
startNode.x = LOWORD(lParam)/tile;
startNode.y = HIWORD(lParam)/tile;
mouse = startNode;
                ⋮
```

시스템 인터페이스 모듈은 그림 5와 같이 캐릭터 이동 제어 기능과 이상 상태 감지 기능으로 구성된다. 캐릭터 이동 제어 기능은 사용자 알고리즘 모듈에서 생성된 함수를 호출하여 리턴되는 정보에 따라 캐릭터를 이동시키는 기능으로 메뉴의 시작버튼의 자동 모드와 알고리즘 호출로 사용되는 수동 모드가 있다. 자동 모드는 캐릭터가 출발점에서 도착적에 도달할 때까지 멈추지 않고 일정 시간에 따라 진행되며, 수동 모드는 한 칸씩 이동이 진행된다. 캐릭터가 이동되면 이동한 방향에 따라 미로에 발자국 형태로 이동경로가 표시된다. 이상 상태 감지 기능은 리턴받은 캐릭터의 위치를 백업 위치 및 미로 데이터에 비교하여 한 번에 여러 칸을 이동한 경우, 벽 등의 이동할 수 없는 지형으로 이동

하는 이상을 감지할 경우, 알고리즘의 오류를 감지할 경우에 경고 메시지를 출력한다.

표 3은 시스템 인터페이스 모듈의 코드로, switch-case 문을 사용하여 캐릭터 이동제어 모듈과 사용자 알고리즘 모듈 사이에서 일어나는 캐릭터 이동과, 이상 이동 등의 오류를 감지하는 코드이다.

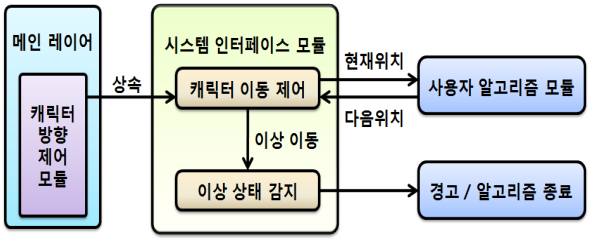


그림 5. 시스템 인터페이스 모듈
Fig. 5. System interface module

표 3. 시스템 인터페이스 모듈 소스
Table 3. System interface module source

```
                ⋮
case WM_TIMER:
case VK_RETURN:
                ⋮
switch(mousehead)
case North:
  Map[mouse.x][mouse.y].status = NorthStep;
                ⋮
mousetemp = mouse;
mouse = LWM(mouse);
                ⋮
if(!((mouse.x == mousetemp.x && mouse.y ==
                ⋮
if(Map[mouse.x][mouse.y].status == Wall ||
                ⋮
```

3.2.3 사용자 레이어

사용자 레이어는 그림 6과 같이 미로 정보 모듈이 제공하는 정보를 받아 사용자에게 알려주는 사용자 미로 정보 모듈과 알고리즘을 자유롭게 적용할 수 있는 사용자 알고리즘 모듈로 구성된다.

사용자 미로 정보 모듈은 메인 레이어의 미로 정보 모듈에서 미로의 정보를 상속 받아 사용자가 알고리즘의 구현에 필요한 정보들을 허용된 범위에서 추가하거나 수정하여 사용자 알고리즘의 완성도를 높일 수 있는 정보를 제공한다. 미로의 정보는 구성된 미로의 형태에는 영향을 주지 않으며, 사용자가 알고리즘을 구성할 때 사용자에 맞게 정보를 가공할 수 있다.

사용자 알고리즘 모듈은 알고리즘을 생성하거나 적용하는 부분으로 사용자가 임의로 편집할 수 있다. 이 모듈은 시스템 인터페이스 모듈에서 받은 현재 위치 정보를 알고리즘에 적용하여 다음 위치를 계산해 리턴하게 된다.

표 4는 구현된 확장좌수법의 일부로 switch-case을 사용하여 캐릭터 방향 정보 모듈에서 얻어온 값과 캐릭터 왼쪽의 타일을 비교하여 방향 전환 및 좌표 이동을 처리한다. 캐릭터의 왼쪽이 이동 가능한 타일이면 캐릭터는 왼쪽으로 방향 전환 후 이동하고, 벽이거나 지나온 길이라면 방향 전환만 한 후 다시 계산한다. 계산이 끝나면 알고리즘은 캐릭터의 새로운 좌표를 저장하고 시스템 인터페이스 모듈로 다음 위치를 리턴한다.

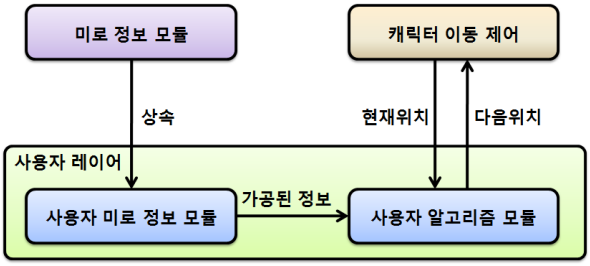


그림 6. 사용자 레이어
Fig. 6. User layer

표 4. 확장좌수법을 적용한 사용자 알고리즘 모듈 소스
Table 4. User algorithm module source applying enhanced left weight method

```
                ⋮
Map LWM(Map mouse)
                ⋮
switch(mousehead)
case West:
  switch(UserMap[mouse.x][mouse.y+1].status)
  case Wall:
    switch(UserMap[mouse.x-1][mouse.y].status)
    case Wall:
      mousehead = North;
    default:
      mouse.x--;
                ⋮
return mouse;
                ⋮
```

4. 실험 및 평가

본 플랫폼은 Windows XP 환경에서 Visual Studio 2008, C++, Windows API를 이용하여 개발하였으며, 테스트 미로는 그림 7과 같이 마이크로 마우스 대회에서 사용하는 미로와 유사하게 작성하였다. 본 연구에서는 좌수법, 확장좌수법, A* 알고리즘 등을 대상으로 장애물이 있는 경우와 없는 경우에 대하여 시간과 스텝 수를 평가하였다. 시간은 캐릭터가 도착점에 도달할 때까지 소요된 시간(초), 스텝 수는 캐릭터가 이동하거나 제자리에서 방향을 변경할 때 마다 1씩 증가하여 얻어진 수치이다. 그리고 장애물로는 늪지대와 문을 사용하였고, 늪지대를 통과할 경우 2초의 시간 가중치가 더해지고, 문은 5초마다 열리고 닫히는 걸 반복하도록 설계하였다. 실험에서는 시간에 우선순위를 두어 평가 결과를 판단하였다.

그림 7은 테스트 미로의 출발점, 도착점, 장애물과 A* 알고리즘을 적용한 캐릭터의 이동 경로를 보여준다. 이 테스트 미로에 언덕오르기 알고리즘을 적용했을 때에도 A* 알고리즘과 동일한 이동 경로를 보여주었고, 두 알고리즘 모두 장애물을 회피하여 경로를 찾았다.

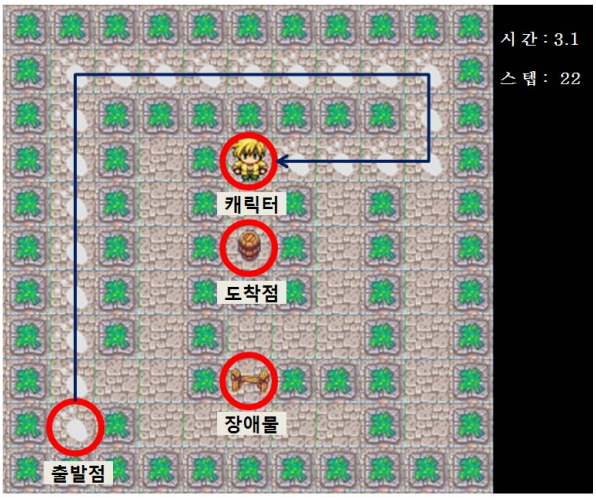


그림 7. A* 알고리즘을 적용한 테스트 미로
Fig. 7. Test maze applying A* algorithm

A* 알고리즘과 언덕오르기는 출발점과 도착점의 좌표 정보를 가지고 먼저 캐릭터의 이동 없이 최상의 경로를 찾기 위한 탐색을 수행한다. 그 다음 도착점을 찾으면 다시 출발점에서부터 캐릭터를 한 칸씩 이동하게 된다. 이때 메모리에 저장된 타일에 대한 평가값에 따라 이동하고, 장애물의 유무에 따라 이동 경로를 달리하여 최단 경로를 탐색하였다. 하지만 문과 같은 가변형 장애물이 있는 경우에는 문을 벽으로 인식하여 반드시 최단 경로를 탐색할 수는 없었다.

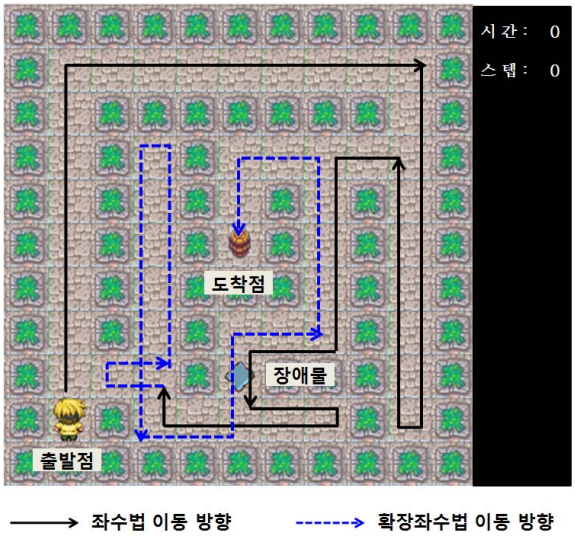


그림 8. 확장좌수법을 적용한 테스트 미로
Fig. 8. Test maze applying enhanced left weight method

그림 8은 테스트 미로에 확장좌수법을 적용할 경우의 캐릭터 이동 경로를 표현한 것이다. 그림에서 실선은 캐릭터가 좌수법과 같이 이동한 경로를 나타내고, 점선은 확장좌수법에서 단일 폐곡선의 무한루프를 방지하기 위해 다른 경로로 이동하는 것을 나타낸다.

확장좌수법은 테스트 미로가 아일랜드 형 미로이기 때문에 단일 폐곡선을 한번 이동하고 다른 길을 탐색하여 도착점을 찾았다. 또한 장애물의 인식 여부와 관계없이 장애물을 피하지 못하고 통과하기 때문에 시간이 더욱 증가하게 된다.

표 5. 아일랜드 형 미로에서 알고리즘 성능
Table 5. Algorithms performance in Island-type maze

알고리즘 \ 평가		시간(초)	스텝	장애물 인식	장애물 통과
A* 알고리즘, 언덕오르기	장애물 없음	2.5	18	-	-
	장애물 문	3.4	24	X	회피
	장애물 늪	3.4	24	O	회피
확장 좌수법	장애물 없음	12.4	87	-	-
	장애물 문	16.4	87	O	통과
	장애물 늪	16.5	116	O	통과

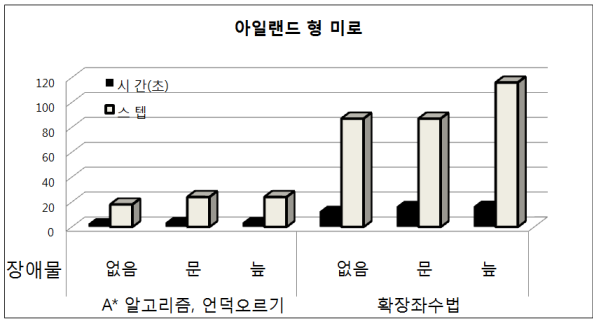


그림 9. 아일랜드 형 미로에서 성능 평가 그래프
Fig. 9. Performance assessment graph in Island-type maze

표 5 알고리즘 평가 결과에서 보듯이 장애물이 있는 경우와 없는 경우 모두 A* 알고리즘과 언덕오르기 알고리즘이 확장좌수법보다 빠른 결과를 보였다.

그림 9는 알고리즘의 성능을 그래프로 표현한 것이다. 그래프에서 왼쪽은 A*알고리즘과 언덕오르기 알고리즘의 시간과 스텝, 오른쪽은 확장 좌수법의 시간과 스텝을 나타낸다. 테스트 미로에서 각 알고리즘의 성능은 그래프와 같이 큰 차이를 보였다.

하지만 좌수법의 경우 그림 10과 같이 아일랜드 형 미로에서 미로의 외곽을 계속해서 돌게 되므로 도착점에 도달하지 못하여 시간과 스텝 수를 측정할 수 없었다.

그림 10. 좌수법을 적용한 테스트 미로
Fig. 10. Test maze applying left weight method

5. 결 론

본 연구에서는 하드웨어로 구현이 어려운 다양한 형태의 미로 제작과 알고리즘의 손쉬운 적용이 가능하고, 스텝, 연산 횟수, 탐색 시간의 평가가 가능한 미로 탐색 알고리즘을 평가하기 위한 플랫폼을 개발하였다. 플랫폼은 메인 레이어, 인터페이스 레이어, 사용자 레이어의 3개 레이어로 구성되어 알고리즘을 쉽게 교체적용 할 수 있는 특징이 있다.

본 플랫폼을 이용하여 좌수법, 확장좌수법, 언덕오르기 알고리즘, A* 알고리즘 등의 미로 탐색 알고리즘을 테스트하여 다음과 같은 결과를 확인하였다. 첫째, 여러 가지 미로 탐색 알고리즘을 동일한 조건에서 적용하여 테스트 할 수 있었다. 둘째, 하드웨어 상에서 구현하기 어려운 장애물을 쉽게 설치하여 다양한 형태의 미로를 생성하고 테스트 할 수 있었다. 향후 다양한 알고리즘의 적용과 미로의 3차원 확장에 대한 연구를 진행할 예정이다.

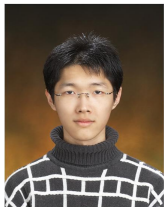
참 고 문 헌

[1] 윤지녕, "*마이크로로봇바이블*", 성안당, 2000.
[2] 이상용, "*인공지능의 세계*", 21세기사, pp. 77-79, 2008.
[3] S. Rabin 외, "*AI Game Programing Wisdom*", 정보문화사, pp. 189-199, 2003.
[4] M. Deloura 외, "*Game Programing Gems*", 정보문화사, 2000.
[5] E. W. Dijkstra, "*A note on two problems in connexion with graphs*", Numerische Mathematik, pp. 269-271, 1959.
[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and Clifford Stein, "*Introduction to Algorithms*", MIT Press and McGraw-Hill, pp. 588 - 592, 2001.

저 자 소 개

서효석(Hyo-Seok Seo)
2010년 : 공주대학교 컴퓨터공학부(공학사)
2010년 ~ 현재 : 공주대학교 일반대학원 컴퓨터공학과(공학석사)

관심분야 : 인공지능, 컨텍스트 인식 등
E-mail : angs@kongju.ac.kr



박재민(Jae-Min Park)
2010년 : 공주대학교 컴퓨터공학부(공학사)
2010년 ~ 현재 : 공주대학교 일반대학원 컴퓨터공학과(공학석사)

관심분야 : 인공지능, 컴퓨터 게임 등
E-mail : soulsieg@kongju.ac.kr

이상용(Sang-Yong Lee)
1984년 : 중앙대학교 전자계산학과(공학사)
1988년 : 일본동경대학대학원 총합이공학연구과(공학석사)
1988년 ~ 1989년 : 일본 NEC 중앙연구소 연구원
1993년 : 중앙대학교 일반대학원 전자계산학과(공학박사)
1996년 ~ 1997년 : University of Central Florida 방문교수
1993년 ~ 현재 : 공주대학교 컴퓨터공학부 교수

관심분야 : 인공지능, 컨텍스트 예측, 컴퓨터게임 등
E-mail : sylee@kongju.ac.kr