Journal of The Korea Society of Computer and Information
Vol. 22 No. 6, pp. 49-55, June 2017

www.ksci.re.kr
https://doi.org/10.9708/jksci.2017.22.06.049

Design and Implementation of a friendly maze program for early childhood based on a path searching algorithm

Unil Yun*, Eun Mi Yun**

Abstract

Robots, games and life applications have been developed while computer areas are developed. Moreover, various applications have been utilized for various users including the early childhood. Recently, smart phones have been dramatically used by various users including early childhood. Many applications need to find a path from a starting point to destinations. For example, without using real maps, users can find the direct paths for the destinations in realtime. Specifically, path exploration in game programs is so important to have accurate results. Nowadays, with these techniques, diverse applications for educations of early childhood have been developed. To deal with the functions, necessity of efficient path search programs with high accuracy becomes much higher. In this paper, we design and develop a friendly maze program for early childhood based on a path searching algorithm. Basically, the path of lineal distance from a starting location to destination is considered. Moreover, weight values are calculated by considering heuristic weighted h(x). In our approach, A* algorithm searches the path considering weight values. Moreover, we utilize depth first search approach instead of breadth first search in order to reduce the search space. so it is proper to use A* algorithm in finding efficient paths although it is not optimized paths.

▸Keyword: maze traversal, path searching algorithm, early childhood

I. Introduction

여러 가지 컴퓨터 분야가 발전하면서 일반 사용자 뿐 만 아니라, 유아등을 대상으로 하는 로봇, 게임, 생활 어플리케이션 등 다양한 프로그램[1-3]이 개발되어 지고 있고, 어린이뿐만 아니라, 유아의 교육을 목적으로 하는 다양한 프로그램에 대한 수요가커지고 있다. 이를 위해 사용하기 편하고 출발지점으로부터 도착지점까지 효율적이고 빠르고 정확도 높은 경로 탐색 알고리즘이 필요성이 커지고 있다. 특히 게임에서는 캐릭터가 움직일 때 경로 탐색 [4-6]은 게임 프로그램을 운영하는데 있어 중요시되고 있다. 다양한 프로그램을 어린이나 유아등을 위한 사용도 많아지고 있고 교육용 프로그램도 많이 개발되어 지고 있다. 고전적인 시뮬레이션 방식의 게임에서는 캐릭터가 특정한 크기의 사각형 영역단위로 움직이는데 적 캐릭터가 있는 곳, 아군 캐릭터가 있는 곳으로는 겹쳐있지 못하고, 사용자가 지정한 곳으로 이동을 해야 한다. 유아등의 사용자가 사용하기 편하게 하기 위해서 보다 정밀한 이동이나 경로 탐색, 지정등이 필요로 하고 있다. 이러한 상황에서 사용자가 원하는 지점까지 최단경로로 장애물들을 피해서 가기 위해서는 경로 탐색 알고리즘은 필수적이다 [7-9]. 경로 탐색 알고리즘이 필요한 곳은 일상생활에서 쉽게 찾아볼 수 있다. 스마트폰에서 목적지를 찾는 지도 어플리케이션에서 각각의 교통수단, 도로, 도

• First Author: Unil Yun, Corresponding Author: Eun Mi Yun
*Unil Yun (yunei@sejong.ac.kr), Dept. of Computer Engineering, Sejong University
**Eun Mi Yun (lovemind114@bau.ac.kr), Dept. of Infantile Education, Baekseok Art University
• Received: 2017. 01. 07, Revised: 2017. 03. 28, Accepted: 2017. 06. 07.
• This research was supported by the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (NRF No. 20152062051 and NRF No. 20155054624).
• I would like to express my gratitude to Dongchun Shin for helping to write this manuscript.

보, 인도 등의 정보를 토대로 도로의 길이, 교통비, 통행료 등의 가중치를 토대로 최단거리, 최소 요금, 최소 소요시간 등의 조건으로 경로를 찾을 수 있다. 또한 일반적으로 순서가 있는 일련의 과정에 해당하는 일에도 쓰임이 가능하다[3].

유아나 어린이들도 많이 사용하는 간단한 예로 슬라이딩 퍼즐에서의 해답도 경로 찾기 알고리즘으로 얻을 수 있다. 퍼즐에서 일정한 순서가 있기 때문에 각 선택이 최종 정답과 얼마나 가까운지를 추정하는 방법으로 가장 나은 선택을 이어가면 정답에 도달할 수 있다는 방식이다. 이처럼 유아를 포함한 모든 사용자들을 대상으로 생활, 게임 전반에 걸쳐서 경로를 찾는 알고리즘은 그 활용도가 높다. 그 알고리즘 중에서 본 논문에서는 A*알고리즘[1-2]을 이용하여 유아들이 사용하기 편한 그래픽 기반의 장애물이 있는 공간에서 최단 경로를 찾는 프로그램을 설계하고 구현한다. 이 알고리즘은 매 순간의 휴리스틱 추정값을 이용하여 가장 효율이 좋은 경로를 범위를 넓혀가며 찾아내는 방식이다.

적용한 다양한 연구[10-11][13-14]가 진행되었고, 이에 대한 응용[12]도 적용되고 있다.

그의 예로 A* 경로계획 적용을 위한 플랫폼 게임 인공지능의 후처리 알고리즘[4]을 제안하였고, 일맵에서 A* 알고리즘을 이용한 유닛들의 길찾기 방법 제안[5]하였으며, 에이스타 알고리즘을 이용한 무인자율주행자동차의 경로 계획[6]을 하였다. 또한, VTA* 알고리즘 [7]을 제안한 연구도 있었고, 최적경로 탐색을 위한 개선된 양방향 A*알고리즘 [8]도 개발되었으며, 컨테이너 터미널에서 베이 내 컨테이너의 최적 재정돈을 위해 A* 알고리즘 [9]이 적용된 연구도 진행되었다. 본 논문에서는 경로 검색 알고리즘의 기본이 되는 다익스트라 알고리즘의 변형인 A*알고리즘을 미로 찾기 알고리즘에 적용한다. A* 알고리즘은 경로 상에 장애물이 복잡하게 있을수록 추정값에 따라 검색을 하므로 약간 필요 없는 부분까지 탐색을 할 수 있지만 비교적 최단 경로를 잘 찾는 장점이 있으며 유아들의 수준에 적당한 미로는 상대적으로 복잡도가 덜하기 때문에 이 알고리즘의 적용으로도 유아들이 사용하기 적당하다고 판단된다.

II. Preliminaries

1. Path Searching Algorithm

다익스트라 알고리즘[2]은 1959년 네덜란드의 컴퓨터과학자 에츠허르 다익스트라의 이름에서 유래한다. 다익스트라 알고리즘은 출발점과 노드와 가중치 간선이 있는 그래프가 주어지면, 각 노드까지의 최단 경로를 찾아준다. 만약 노드 a에서 b까지의 최단 경로를 d[b] 라고 하고 이 길이를 알고 있으며, 노드 b에서 노드 c까지 길이가 w(b,c)인 변이 있다고 가정하면, a에서 c까지 최단경로는 d[b]+w(b,c)가 된다. 이것을 기본 아이디어로 모든 노드에 대해 경로의 비용이 적어지는 노드를 선택하다보면 각 노드에 접근하는 최단 경로를 구할 수 있다. A* 알고리즘은 다익스트라 알고리즘의 변형이다.

A* 알고리즘[2]은 1968년 저자 피터 하트, 닐스 닐슨, 버트램 라팰이 처음 고안했다. 저자는 A알고리즘이라고 언급하였지만, 적절한 휴리스틱 추정값을 사용해서 이 알고리즘을 사용하면 경로는 최적의 결과를 보여준다고 해서 A* 알고리즘이라고 불린다. A* 알고리즘은 자신이 지나온 경로의 가중치와, 목표지점까지의 가중치값을 더한 휴리스틱 추정값을 이용해서 최적의 경로를 찾는다. 따라서 반드시 출발하는 지점과 목표 지점이 있는 상황에서만 길을 찾을 수 있다는 단점이 있다.

게임에서 유닛의 이동과 같은 경우는 사용자가 직접 마우스 클릭이나 다른 명령을 통해서 목표지점을 설정할 수 있기 때문에 A* 알고리즘으로 길을 찾기에 적절하지만, 출구가 어느 곳인지 정해지지 않은 실제 미로와 같은 상황에서는 적절하지 않다. 도착점을 알아야 하므로 다익스트라 알고리즘과 유사하지만, 한 변씩 경로를 추가하며 최단 경로를 찾는 다익스트라 알고리즘과 비교해서 A*는 도착점까지의 경로 추정값을 사용한다는 점이 다르다. 기존의 관련연구[4-8]에서 A* 알고리즘을

III. Design and Implementation of Maze finding program

1. A* Algorithm

A* 알고리즘[1-3]은 앞에서 언급했다시피 출발점과 도착점 장애물에 대한 정보가 있을 때 도착점까지의 경로를 찾는 알고리즘이다. 이러한 정보들을 이용해서 경로를 찾는 과정은 아래와 같다.

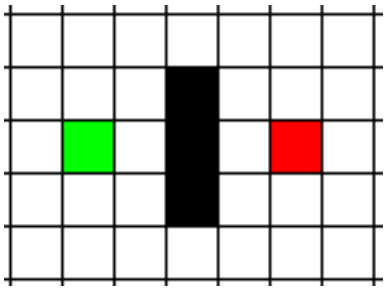

Fig. 1. Basic Map

A* 알고리즘 [1-3][10-11]에서는 출발점과 도착점의 경로를 나누어진 구역을 따라 인접한 구역을 넓혀가며 찾는 방식이다. 그 중에서 가장 간단하게 가로 세로 길이가 같은 정사각형의 형태로 나누어진 구역에서의 길을 찾는 방식을 설명한다. 그림 1과 같이 사각형의 격자로 나누어진 구역에서 각 사각형은 출발점, 도착점, 장애물, 이동가능 구역 이렇게 크게 4종류의 정보를 가질 수 있다. 그림 1에서 초록색 사각형은 출발점, 빨간색 사각형은 도착점, 검은색 사각형은 장애물, 나머지 흰색 사각형은 이동 가능한 비어있는 공간으로 생각할 수 있다. 길을

찾는 과정의 시작은 출발점의 사각형으로부터 인접한 8개의 사각형에 대해서 휴리스틱 추정값 h(x)를 계산해 나간다. 출발 사각형에서 처음 수행하는 과정은 표 1과 같다.

Table 1. Calculation of starting rectangle

Add starting rectangle to the opened list. Find an adjacent rectangle of the starting rectangle. Add the passable rectangle among adjacent rectangles to the opened list and store the parent rectangle of the added rectangle as a starting rectangle. Calculate and store F cost of the added rectangle respectively. Remove the starting rectangle to the opened list and add the starting rectangle to the closed list.

여기서 열린 목록은 최종 경로가 될 수 있는 후보 사각형이라고 생각을 하면 쉽다. 앞으로 비슷한 과정을 반복할 때 특별한 방법으로 계산된 추정값에 따라 열린 목록 중에서 사각형 하나를 선택해서 인접 사각형에 대해 경로를 찾는 과정을 반복하게 된다. 반대로 닫힌 목록은 그 사각형에 대해서 경로에 대한 비용을 계산하고, 인접한 사각형에 대해서 모든 계산을 마친 사각형 즉, 더 이상 이 사각형이 더 좋은 경로인지 아닌지 확인할 필요가 없는 사각형을 의미한다. 현재 사각형을 선택하는 기준은 열린 목록 중에서 비용값이 가장 작은 사각형을 하나씩 선택하게 되므로 선택된 사각형보다 더 좋은 비용을 가진 사각형은 있을 수 없으므로 닫힌 목록이 된다.

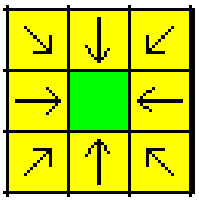

Fig. 2. Basic Map

출발점에서 위의 과정을 거치게 되면 위 그림 2에서 처럼 인접한 8개의 사각형의 부모 사각형은 출발점을 가리킨다. 부모 사각형은 사각형에서 부모사각형을 거쳤을 때 비용이 가장 적게 드는 것을 의미한다. 다시 말하면 자신의 위치까지 오는데 부모 사각형을 통해서 오는 것이 가장 효율적이라는 의미다. A* 알고리즘에서 가장 중요한 휴리스틱 추정값을 구하는 방법은 아래의 표 2의 식과 같다.

Table 2. Calculation of starting rectangle

$F = G + H$ F : Cost G : Distance from the starting point to the current point H : Distance from the current point to the destination point

이 식에서 F는 총 비용으로, 하나의 사각형이 현재까지 자신이 출발점에서 현재 지점까지 경로를 찾은 거리인 G 값과, 현재 지점에서 도착지까지의 어림짐작한 거리인 H값의 합이다. 여기서 확실히 구분해야할 것은 H값은 현재 지점부터 도착지점까지의 직선적인 거리를 의미하고, G값은 출발 지점부터 현재 지점까지의 실제 경로의 거리를 의미한다. A* 알고리즘은 인접한 구역의 비용값만 계산해나가므로 도착지점에 인접해 있지 않은 이상 실제 도착점까지의 경로 거리는 바로 알 수 없다. 따라서 직선적인 거의의 추정값인 H값을 사용하는 것이다. H값을 계산할 때에는 중간에 있는 장애물은 전혀 고려하지 않는다.

F값을 계산하는 하나의 예를 들면, 그림 1의 초록색 사각형 오른쪽 상단의 사각형을 열린 목록에 추가할 때를 생각해보자. 우선 G값은 출발점으로부터 현재 사각형까지의 경로 거리다. 이 값을 계산하려면 현재 사각형의 부모 사각형의 G값과, 부모 사각형에서 현재 사각형까지의 거리 값을 더하면 된다.

부모사각형에도 출발점으로부터 부모 사각형까지의 경로 거리의 합이 G값으로 저장되어 있기 때문에 이러한 계산을 할 수 있다. 위 예에서는 부모사각형이 출발점이기 때문에 부모 사각형의 G값은 0, 출발점부터 오른쪽 위의 사각형까지의 거리는 14로 할 수 있다. 직각 거리를 10 대각선 거리를 14로 했을 때 거리가 간단한 정수형으로 나와서 계산이 편리하다. 즉 결과적으로 G값은 0+ 14=14 가 나온다. 그리고 H값은 현재 사각형으로부터 도착점까지의 직선적 거리인데, 대각선 방향을 고려하지 않고, 가로축 세로축 방향을 고려하여 거리를 계산한다. 예제에서는 가로축으로 3칸, 세로축으로 1칸을 이동하면 되므로 총 4칸에 각 사각형 간 거리 10을 곱한 40이 H값이 된다. 이 때 중간의 장애물은 고려하지 않는다. 이렇게 계산된 G=14 와 H=40을 더한 F값은 54가 된다. 이 과정을 인접한 8개의 사각형에 대해 모두 진행하면 8개의 사각형은 모두 출발 사각형을 부모 사각형으로 가리키고 각각의 F비용을 계산해서 가지고 있게 된다. 또한 열린 목록에는 8개의 사각형이 들어간다.

다음 과정은 이전에 출발점에서 했던 과정을 똑같이 반복하는 것이다. 그렇다면 기준이 되는 사각형을 정하는 것이 가장 큰 문제인데 그것은 F값을 이용하면 간단하게 정할 수 있다. 바로 열린 목록에 있는 사각형 중에서 F값이 가장 작은 사각형을 선택하는 것이다. F값이 작다는 것은 출발점부터 그 사각형까지의 거리와 그 사각형부터 도착점까지의 거리가 짧다는 것을 의미하므로 최단 경로에 근접한 사각형이다. 따라서 F값이 작은 사각형을 잠재적인 해답 경로라고 생각하고 계산을 이어나간다.

열린 목록에 있는 사각형 중에서 F값이 가장 작은 사각형을 선택했다면, 출발점에서의 과정과 같이 인접한 8개의 사각형에 대해 계산 작업과 수정 작업을 수행한다. 장애물이라면 아무것도 하지 않고, 비어있는 사각형이라면 그 사각형의 F비용을 계산해 넣고, 그 사각형의 부모 사각형을 현재 사각형으로 표시한다. 마지막으로 만약 인접한 사각형이 열린 목록에 들어있다면 그 사각형의 부모 사각형이 계속 그대로 있어도 좋은지 검사한다. 다시 말해서 그 사각형까지 가는 경로가 현재의 사각형을 거치는 것이 좋다면 그 사각형의 부모를 현재 사각형으로 업데이트한다. 이것을 확인하는 방법은 열린 목록에 저장되어 있는 G값과 현재사각형을 거쳤을 때의 G값을 새로 계산해서 비교해서 만약 현재 사각형을 통한 G값이 더 작다면 인접 사각형의 부모를 현재 사각형으로 바꾼다.

2. Screen design

본격적인 프로그램은 MFC를 이용한 그래픽 기반 프로그램이다. 우선 프로그램을 실행했을 때 출발점과 도착점이 표시되어 있고, 마우스 입력을 통해서 장애물을 그릴 수 있고, 엔터키를 누르면 출발점부터 도착점까지 최단경로를 찾고 도형이 이동하는 애니메이션을 볼 수 있다. 최단경로를 찾는 방법은 A* 알고리즘을 반복적으로 동작하게 해서 열린 목록과 닫힌 목록을 기록해나가면서 도착점을 찾는다.

그림 3에서 볼 수 있듯이 화면을 20*20 사이즈들의 사각형으로 구성하고, 왼쪽의 초록색 사각형은 출발점, 오른쪽 빨간색 사각형은 도착점으로 표시한다. 그리고 화면에서 보이는 검은색 사각형들은 마우스로 입력할 수 있는 장애물들이다. 이 상태에서 엔터키를 누르면 경로를 찾고, 길을 찾으면 파란색 공이 경로를 따라서 움직인다.

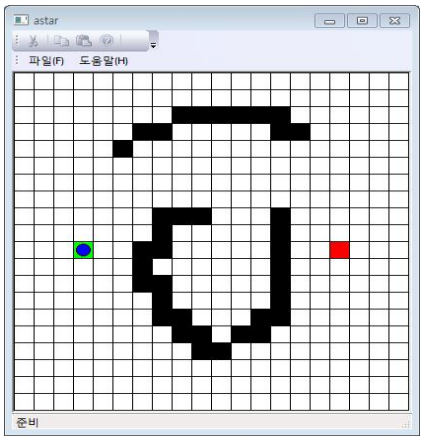

Fig. 3. Program Interface

3. Program structure

MFC 기반으로 프로그램을 작성하였고, 길을 찾는 함수인 find_route(), 초기화 함수 init(), 애니메이션을 위한 OnTimer() 이벤트 핸들러를 사용했다. 인접한 사각형을 탐색하는 과정을 반복문으로 구현했고, 반복 종료조건을 줌으로써 최종 경로를 찾을 수 있다. 길을 찾는 find_route() 함수를 의사코드로 표현하면 표 3과 같다.

기본적으로 사각형 각각의 정보를 저장하기 위한 구조체를 만들고 그 속성 값으로 열린 목록, 닫힌 목록, 출발점, 도착점, 장애물 여부, g·h·f값, 사각형의 좌표, 부모 사각형의 좌표, 정답길 여부 등을 만든다. 길을 찾을 때 이러한 속성 값들을 업데이트 하는 방식으로 길을 찾아나간다.

처음 시작은 출발점에서부터 시작해야 하므로 1번 줄에서 앞으로 정답 경로가 될 수 있는 목록인 열린 목록에 출발 사각형을 추가한다. 2번 줄에서 반복문을 시작하는데 길을 찾거나, 모든 가능한 경우를 다 찾아도 길을 못 찾는 경우에만 반복문을 빠져나온다. 그 조건은 3, 4번 줄에서 볼 수 있다. 열린 목록이 비었다는 것은 모든 열린 목록을 현재 사각형으로 선택해서 인접한 사각형을 다 봤는데도 도착 사각형을 못 찾았다는 의미다. 즉 장애물에 가로막혀 도착점까지 진행할 수가 없으므로 길을 찾지 못하고 false를 반환한다. 반대로 도착점이 열린 목록에 추가된 것은 인접한 사각형을 계속 찾아나갔을 때 도착점까지 도달한 것이므로 경로를 찾은 것이고, true를 반환해서 길을 찾았음을 알린다. 위 두 가지 경우가 아니라면 다음의 작업을 계속한다.

처음에는 인접한 사각형을 살펴볼 현재 사각형을 선택한다. 이 사각형을 선택하는 기준은 열린 목록에 있는 사각형 중 F비용이 가장 작은 사각형이다. 이유는 A* 알고리즘에서는 F비용이 가장 작은 사각형이 정답 경로일 확률이 높다고 판단하기 때문이다. 처음에는 열린 목록에 출발 사각형만 들어있으므로 출발 사각형이 선택되고 이렇게 선택된 사각형은 이제 열린 목록에서 제외하고 더 이상 살펴볼 필요가 없는 닫힌 목록에 추가한다. 이 과정은 5~7 줄에서 보여준다.

이러한 과정으로 현재사각형이 선택되면 현재 사각형에 인접한 8개의 사각형을 순차적으로 보면서 경우에 따른 연산을 한다. 구현한 프로그램에서 나타날 수 있는 경우는 크게 3가지다. 첫째, 진행할 수 없는 사각형. 이 경우는 지정된 맵의 범위를 벗어나거나, 대각선으로 움직이려고 할 때 장애물이 있는 경우다. 이는 그림 4 에서처럼 가운데 선택된 현재 사각형이 대각선으로 오른쪽 위쪽 사각형으로 진행하려고 비용 값을 계산하려고 하는데 만약 그 왼쪽이나 아래쪽에 장애물이 있다면 그 방향은 갈 수 없는 경로로 판단한다. 즉 열린 목록에 추가하지 않고, 다른 인접한 사각형을 확인한다. 경로에서 사각 형이 이동한다고 생각하기 때문에 이러한 조건을 붙인다. 그림 6에서 대각선으로 이동하기 위해서는 장애물을 절반가량 뚫고 지나가야 한다. 물론 방식을 다르게 할 순 있지만 구현은 이런 상황에서 오른쪽으로 돌아가도록 했다. 이는 의사코드 17~20줄에서 보여준다.

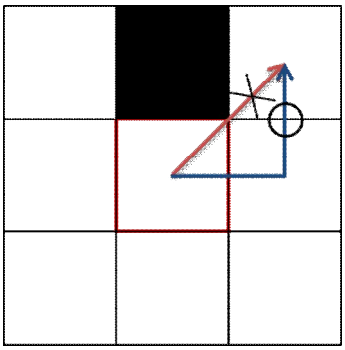

Fig. 4. Path of Rectangle proceeding

둘 째, 장애물인 경우다. 이 경우도 앞의 경우와 마찬가지로 열린 목록에 추가하는 등의 추가적인 작업을 하지 않고, 다른 인접한 사각형을 확인한다. 마지막으로 세 번째는 진행할 수 있는 비어있는 사각형인 경우다. 의사코드 21~27줄에서 확인할 수 있는데, 우선 인접한 사각형의 부모 사각형을 현재 사각형으로 가리키게 한다. 그리고 출발 사각형으로부터 인접한 사각형까지의 경로 거리인

G값을 계산해서 넣어야하는데 이 G값은 현재사각형의 G값에 현재 사각형부터 인접 사각형과의 거리를 더하면 된다. 인접사각형이 대각선 거리에 있다면 14를, 수평 수직 칸에 있다면 10을 더한다. 이는 피타고라스 정리에 의해 대각선 길이를 14로 정수형으로 바꾸어서 계산을 빠르게 하는 방법이다. 다음으로 H값은 도착점까지의 예상되는 거리의 추정값인데 이 값은 도착점까지의 직선거리가 아닌, 수평·수직 이동 값들의 합이다.

예컨대 그림 4 에서 좌측 상단의 사각형으로부터 우측 하단의 사각형까지의 거리는 가로로 2, 세로로 2칸이다. 한 칸의 거리는 10이므로 추정 거리는 40이 된다. 이렇게 계산된 G값과 H값을 더해서 총 비용 F를 저장한다. 이 과정은 의사코드 23~25 줄에서 보여준다. 이 과정이 끝나면 해당 인접 사각형을 열린 목록에 추가한다. 이렇게 하나의 사각형을 선택하고 닫힌 목록에 추가하고, 인접 사각형들을 열린 목록에 추가하고, 또 다시 F값이 적은 사각형을 선택하는 과정을 반복하면 범위를 넓혀가며 길을 찾다가 열린 목록에 도착점이 들어가거나, 열린 목록이 비면 반복을 마치고 경로 탐색을 마친다.

Table 3. Pseudo codes of find_route function

```
OpenedList.Add(startingRectangle);
while(){
  if (openList is empty) return false;
  if (destinationPoint is added to openedList) return true;
  currentRectangle=min(openedList.F-cost);
  openedList.Remove(currentRectangle);
  closedList.Add(currentRectangle);
  for (each adjacent rectangle of current rectangle){
    if (opened list exists){
      if (G through current rectangle < previous G){
        adjacentRectangle->parentRectangle=currentRectangle;
        adjacentRectangle.G=currentRectangle.G
                          +distance from currentRectangle;
        adjacentRectangle.H=distance from adjacent Rectangle
                                    to the destination point;
adjacentRectangle.F=adjacentRectangle.G+adjacentRectangle.H;
      }
    }
    else if(it is the unprocessable rectangle)
      continue;
    else if(it is obstacle)
      continue;
    else if(it is processable but emply rectable){
      adjacentRectangle->parentRectangle=currentRectangle;
      adjacentRectangle.G=currentRectangle.G+distance from
                                              currentRectangle;
      adjacentRectangle.H=distance from adjacentRectangle
                                    to destination point;

adjacentRectangle.F=adjacentRectangle.G+adjacentRectangle.H;
      openedList.Add(adjacentRectangle);
    }
  }   // End of adjacentRectangle process
}     // Completeness of path finding
```

find_route 함수에서의 과정을 거치고 나면 각 사각형마다 열린 목록, 닫힌 목록 정보 등이 저장된다. 만약 도착 사각형이 열린 목록에 추가되었다면 최종 경로를 찾을 수 있다. 사각형이 가리키는 부모 사각형은 그 사각형을 통해 자신의 위치까지 왔을 때 휴리스틱 추정값 h(x)의 값이 최소가 된다는 것을 의미한다. 따라서 도착 사각형에서 부모 사각형을 계속 따라가다 보면 결국 출발 사각형이 나온다. 이렇게 출발점에서 도착점까지의 최단 경로가 구해진다. 프로그램에서는 이러한 과정을 눈으로 볼 수 있도록 보여준다.

Ⅳ. Implementation screen of Maze finding program

구현한 프로그램은 윈도우 프로그램으로 창을 통한 마우스, 키보드 입력을 받을 수 있다. 또한 메뉴 버튼을 통한 명령의 수행도 가능하다. 기본적으로 처음 프로그램을 실행하면 출발점은 초록색, 도착점은 빨간색 사각형으로 나타나고 장애물은 아무것도 없다. 이 상태에서 마우스 왼쪽 버튼을 누르고 드래그하면 드래그 하는 곳의 사각형이 검은색 사각형으로 표시된다.

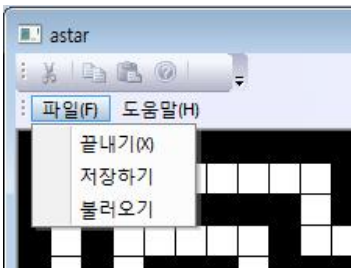

Fig. 5. Buttons of save and load

그림 5에서처럼, 입력한 장애물 정보와 맵 크기 등의 정보를 저장하기 위해서 저장하기 불러오기 기능을 구현했다. 어느 상태에서든 저장하기 버튼을 누르면 사각형의 크기, 맵 사이즈, 장애물 위치, 출발점, 도착점의 위치가 텍스트 파일 형식으로 저장되고, 불러오기를 선택하면 그 상태 그대로 불러올 수 있다. 스크롤바를 위 아래로 움직이면 사각형 하나마다의 사이즈를 늘리고 줄일 수 있다. 또한 키보드 방향키를 이용해서 맵의 크기를 줄이거나 늘릴 수 있다.

그림 6은 미리 저장해둔 미로를 불러온 화면이다. 출발점과 도착점 역시 마우스 클릭을 통해서 얼마든지 위치를 변경할 수 있고, 장애물을 그린 상태에서 엔터키를 누르면 사각형들을 열린 목록에 추가하고 부모 사각형이 어떤 사각형을 가리키는지 그림 5와 같이 볼 수 있다. 청록색 사각형은 닫힌 목록, 노란색 사각형은 열린 목록이다. 이미 경로를 찾았기 때문에 파란 공이 진행을 하고 있고, 지나온 길을 표시하기 위해 빨간색 화살표를 방향을 바꿔서 그린다. 원래는 도착점에서 출발점으로 거꾸로 찾아오기 때문에 화살표가 반대로 되어 있지만, 파란 공을 이동시키며 방향을 바꿔준다.

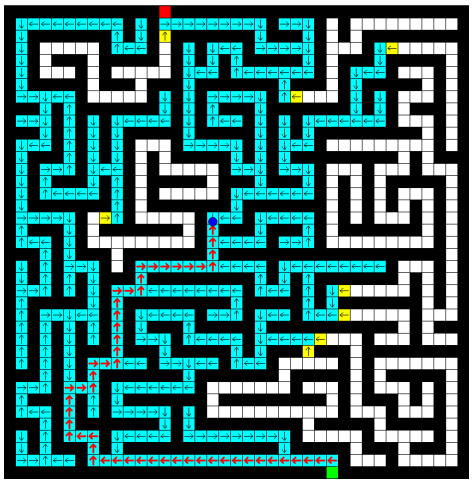

Fig. 6. Implementation example of final program

Ⅴ. Conclusions

본 논문에서는 경로 검색 알고리즘을 적용하여 유아들이 친근하게 사용할 수 있는 퍼즐 프로그램을 설계 하고 구현하였다.

경로 검색 알고리즘 중 출발점과 도착점이 있는 환경에서 최적의 경로를 찾는데 효율적인 A* 알고리즘을 적용하였으며, 모든 가능한 경우의 수를 전부 찾는 방법이 아니고 추정 값을 근간으로 경로를 찾기 때문에 제한점이 있지만 유아들이 사용하기에 한정된 미로에서 빠르게 경로를 찾을 수 있기에 유아용 미로 프로그램에 적용하는데 적당하다고 판단된다. 추후연구로는 경로 탐색알고리즘중 보다 유아에게 적당한 탐색알고리즘들을 적용할 수 있다. 또한, 유아용 그래픽 사용자 인터페이스(GUI)에 좀 더 신경을 써서 유아들이 사용하기 더욱 편한 친근한 사용자 인터페이스로 개선이 필요하며 기능적으로도 복잡한 기능보다는 간단하면서 다양한 기능의 개선이 필요하다. 이러한 추후연구는 복잡하지 않은 환경의 유아용 게임이나 교육용 프로그램에서도 더 많이 적용될 수 있으며 특화된 영역에도 적용이 가능할 것이다.

REFERENCES

[1] Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. Systems Science and Cybernetics, IEEE Transactions on, 4(2), 100-107.

[2] Dijkstra, E. W. (1959). A note on two problems in connection with graphs. Numerische mathematik, 1(1), 269-271.

[3] AlShawi, I. S., Yan, L., Pan, W., & Luo, B. (2012). Lifetime enhancement in wireless sensor networks using fuzzy approach and A-star algorithm. Sensors Journal, IEEE, 12(10), 3010-3018.

[4] H. Kim, K. Kim (2013). Post Processing Algorithm Proposal of Platform Game Artificial Intelligence for A* Path Planning Application. KIISE conference, 643-645.

[5] S. Lee (2004). Units' Path-finding Method Proposal for A* Algorithm in the Tilemap. KSCI, 9(3), 71-77.

[6] G. Song, J. Lee (2012). Path Generation for Autonomous Vehicle using A* Algorithm. ICROS, 125-126.

[7] J. Kim, D. Jo (2010). VTA* Algorithm. KIICE, 14(3), 663-668.

[8] S. Lee, G. Lee, B. Hwang, W. Jung (2002). Modified Bidirectional A* Algorithm for Optimal Car Route Guidance Search. Korea Society For Internet Information, conference, 3(1), 42-45.

[9] B. Ha , S. Kim (2012). A* Algorithm for Optimal Intra-bay Container Pre-marshalling Plan. Journal of the Korean Institute of Industrial Engineers, 38(2), 157-172.

[10] S. Han (2013). Implementation of Active Location Detecting System using A-Star Algorithm. KIITORKR, 11(11), 133-140.

[11]J. Go, S. Park (2013). AGV Integrated System Using A-Star Algorithm in Random Space. KIITORKR, 11(12), 227-235.

[12] H. Jo, D. Kim, J. Lee, J. Park (2010). Implementation of Active Location Detecting System In 3D. KIISE conference, 37(1), 157-172.

[13] Cheng LP, Liu CX, Yan B, Improved hierarchical A-star algorithm for optimal parking path planning of the large parking lot. In: 2014 IEEE international conference on information and automation, 2016, pp. 695- 698.

[14] Akshay Kumar Guruji, Himansh Agarwal, D.K. Parsediya, Time-efficient A* Algorithm for Robot Path Planning, 3rd International Conference on Innovations in Automation and Mechatronics Engineering 2016, ICIAME 2016, pp. 144-14

Authors

Unil Yun received the M.S. degree in computer science and engineering from Korea University, Seoul, Korea, in 1997 and the Ph.D. degree in computer science from Texas A&M University, College Station, TX, USA, in 2005.

From 1997 to 2002, he worked with the Multimedia Laboratory, Korea Telecom. After receiving the Ph.D. degree, he worked as a Postdoctoral Associate for almost one year at the Department of Computer Science, Texas A&M University. Then, he worked as a Senior Researcher with the Electronics and Telecommunications Research Institute. In March 2007, he joined the School of Electrical and Computer Engineering, Chungbuk National University, Korea. Since August 2013, he has been with the Department of Computer Engineering, Sejong University, Seoul. His research interests include data mining, information retrieval, database systems, artificial intelligence, and digital libraries.

Eun Mi Yun received the B.S. M.S. and Ph.D. degrees in Infantile Education from Chung-Ang University, Korea, in 1990, 2000 and 2007, respectively Dr. Yun joined an adjunct professor of the Department of Infantile Education at Ajou University Graduate School of Education, Seoul, Korea, in 2009-2010. She is currently a Professor in the Department of Infantile Education Baekseok Art University. She is interested in early childhood computer education and early childhood mathematic and science education and parent education.