게임지도에서 에이전트 이동을 위한 경로표 활용

심 동 희† · 강 혁††

요 약

게임지도상에서 에이전트 이동을 위한 경로탐색에 A'를 이용하는 경우는 실시간 게임진행에 지장을 주고있다. 다른 경험적 탐색방법은 경로에 대한 최적성 보장이 없어서 문제가 된다. 이러한 문제를 해결하기 위하여 본 논문에서는 현위치, 목표위치, 최적경유위치로 정의되는 행으로 구성되는 경로표를 제안하였다. 이 경로표를 게임제작시에 미리 작성하여 게임진행시에 이를 이용한다. 이 경로표는 목표위치로의 이동을 위한 최적의 경유위치를 갖고 있어서 최적의 경로도 보장이 되며 탐색으로 인한 시간의 부하도 없다는 장점을 갖고 있다. 그러나 경로표 저장에 메모리가 많이 소요되는 단점도 있다. 이 논문에서는 이 같은 단점을 해소하기 위해 같은 경로표는 생략하는 방법을 이용해도 데이터압축효과를 얻을 수 있음도 보여주었다.

Utilization of the Route Table for the Agent's Move in the Game Map

Dong-Hee Shim† · Hyuk Kang††

ABSTRACT

The use of the A' for the path search of the agent in the game map give the overhead of computing time in real time game processing. The other heuristic search algorithms do not guarantee the optimal path. The route table of which the row is defined by current position, goal position, visiting position is presented in this paper. This route table is made in the game development phase and utilized in game playing. The visiting position which is contained in the optimal path to the goal position from the current position can guarantee the optimal path, and this method has no overhead on computing time. But the memory space is required too much. This problem can also be solved using the data compression by skipping the duplicated route table.

1. 서 론

비디오 게임은 이미지의 처리방식에 따라 2D와 3D로 대별된다. 2D에 속하는 비디오 게임은 다시 에이전트의 이동을 가로 막는 블럭영역이 있는 경우와 블럭영역이 없는 경우로 나누어 볼 수 있다. 에이전트의 이동에서도 움직이는 방향에 따라 4방향, 8방향, 16방향, 32방향으로 분류할 수 있다. 그런데 블럭영역이 없는 경우 에이전트는 주어진 게임지도상에서 위치를 이동할 경우 좌표를 이용하여 직선으로 이동할 수 있으므로 경로탐색이 필요하지 않다. 그러나 블럭영역이 있는 경우에는 에이전트는 이동을 위한 경로를 탐색해야 한다. 이러한 블럭영역은 다시 게임의 실행중에 동적으로 위치가 변하는 경우와 정적으로 고정되어 있는 경우로 나누어 볼 수 있다. 그런데 대부분의 게임에는 이러한 블럭영역이 정적이든 또는 동적이든 존재하기 때문에 경로탐색이 필요하다. 그래서 에이전트의 지능적인 이동을 위해서는 이러한 경로탐색이 효과적으로 이

※ 이 논문은 1999년도 전주대학교 학술연구조성비에 의하여 작성되었음
† 종신회원 전주대학교 전자네트워크공학부 교수
†† 준 회 원 전주대학교 대학원 컴퓨터공학과
논문접수 2000년 5월 12일, 심사완료 · 2000년 10월 4일

루어져야 한다. 상용화되어 있는 많은 게임에 이러한 경로탐색을 위하여 A* 알고리즘[3]이 이용되고 있는 것으로 알려지고 있다. 그런데 최적의 경로를 탐색하기 위한 A*와 같은 알고리즘은 수행시간을 많이 요구하기 때문에 실시간을 위한 게임에는 다소 부적합하다 A*가 갖는 수행시간에 대한 이러한 단점으로 인하여 많은 비디오 게임에서는 최적경로에 근접한 경로를 구하기 위해 게임 도메인의 지식으로부터 도출한 함수를 이용하여 Greedy 탐색방법을 사용하고 있다[1, 2, 5]. 그러나 이 탐색방법은 최적경로를 도출해주지 않는다는 단점을 갖고 있다[4]. 다른 게임도 어떤 유형의 탐색방법을 이용하고 있지만 게임내부의 프로그램 및 그 사양을 상업적인 이유로 인하여 공개하지 않고 있기 때문에 정확히 어떤 탐색을 이용하는지 모르는 경우가 사실 많다. 블록영역이 동적으로 변하는 경우에는 게임진행중에 A*나 Greedy 탐색을 이용하여 경로를 탐색할 수 밖에 없지만 블록영역이 정적인 경우에는 게임 개발시에 에이전트의 이동을 위하여 최적경로를 미리 작성하여 실시간게임진행에 활용할 수 있을 것이다.

본 논문에서는 블록영역이 정적인 경우 4방향으로 이동할 수 있는 2D게임에서 실시간게임진행에 지장을 주지 않으면서 최적경로를 구하여 이용할 수 있는 방안을 도출하고자 한다. 이를 위하여 먼저 주어진 게임지도상에 존재하는 모든 위치들간의 최적경로를 미리 탐색한 후 이 최적경로를 게임에서 바로 이용함으로써 수행시간을 줄이는 방법을 제안하였다. 또한 모든 최적경로를 저장하는데 소요메모리를 최소화하기 위하여 중복되는 경로는 하나로 저장하는 방법을 제안하였다.

그래서 2장에서는 최적경유필수위치 개념을 제시하고 이에 기반한 경로표를 새로 제안하였으며, 또한 이를 이용한 에이전트의 이동알고리즘을 설계하였다. 3장에서는 경로표를 메모리에 효율적으로 저장하기 위한 경로표 압축방법을 제시하였다. 4장에서는 경로표의 성능평가를 위하여 A*를 이용하는 경우와 경로표를 이용하는 경우에 대한 비교를 하였다

2. 게임지도상의 경로와 이동

2.1 최적경로와 최적경유필수위치

게임이 수행되는 지도상의 모든 위치는 이동을 제어하기 위하여 식별자를 가져야 하는데 이 식별자는 게임지도상의 좌표를 사용하는 것이 바람직하다 그래서 게임에서는 보통 픽셀단위로 부여하는 좌표를 이용하여 위치를 식별하게 된다. 또한 이러한 좌표를 이용할 때 에이전트는 상하좌우 4방향으로 이동할 수 있게 된다. 2차원 게임의 경우 현위치가 (a, b)이면 이동가능 위치는 (a+1, b), (a, b+1), (a-1, b), (a, b-1)이 된다. 그런데 현위치에서 목표위치로 이동하는 최적경로는 이러한 지도좌표상에서 도출이 된다. 그런데 게임지도에서 경로의 이동비용을 보통 똑 같다. 다만 에이전트가 접근할 수 없는 영역(경로비용이 ∞에 해당)이 있는데 이를 블록영역이라 한다. 만일 어떤 게임지도에서 블록영역이 없다면 최적경로는 좌표에 의하여 1회의 계산에 의하여 산출될 수 있다. 예를 들어 현위치 (a, b)에서 목표위치 (c, d)로의 이동을 위한 최적경로는 직선상의 이동을 하면 되는 것이므로 (a, b)에서 (a, d)를 경유하여 (c, d)로 이동하거나 또는 (c, b)를 경유하여 (c, d)로 이동하면 된다. 이러한 두 개의 위치 (a, d), (c, b)를 최적경유필수위치라고 하자. 즉 게임지도상의 경로를 서로 연결하여 작성한 선분에서 같은 선분에 있는 끝 위치를 최적경유필수위치라고 하자. 그러나 게임지도에는 블록영역이 있기 때문에 최적경로를 도출하기 위한 탐색이 필요하게 된다 A*와 같은 알고리즘을 이용하여 최적경로를 도출하면 이 최적경로는 유한한 개수의 선분으로 구성된다 이때 각 선분의 끝점은 최적경유필수위치에 해당하게 된다.

〈표 1〉 경로표

목표위치	최적경유필수위치
(x1, y1)	(x2, y2)
.	
.	

2.2 에이전트의 최적경로 도출방법

게임진행시 에이전트가 이동하기 위해서는 최적경로를 구해야 한다. 앞서 서론에서도 설명한 바와 같이 게임의 진행시에 최적경로를 구하면 실시간 진행에 부담을 주기 때문에 바람직하지 않다. 이 단점을 해결하는 방법으로 게임 설계시에 각 위치마다 모든 목표위치에 대해 최적경로를 미리 작성하여 이를 저장한 후 게임시에 이를 이용할 수 있다. 이 방법은 게임의 실시간 진행에는 거의 부담을 주지 않는 방법이다 그러나 모든 위치에서 모든 위치로의 최적경로를 저장하는

것은 많은 메모리를 요구하기 때문에 이 또한 적절한 방법이 될 수 없다. 이러한 메모리 부담을 경감할 수 있는 방법으로 최적경로를 모두 기억시키지 않고 선분의 단위로만 기억시키는 방법을 활용할 수 있다. 그래서 각 위치마다 목표위치에 대한 최적경유위치를 기억시켜 이를 반복적으로 이용하면 이러한 메모리 부담을 경감할 수 있다.

2.3 경로표

지도상의 모든 위치는 <표 1>과 같은 경로표를 갖도록 한다. <표 1>에서 목표위치는 현위치에서 이동하고자 하는 위치이며 최적경유필수위치는 목표위치로 이동하기 위한 최초의 최적경유필수위치를 나타낸다.

2.4 최적경유필수위치의 성질

<표 1>에 표기된 최초의 최적경유필수위치는 현위치에서 최적경유필수위치로 가는 길로가 바로 계산될 수 있도록 일직선상에 위치해야 한다. 즉 최적경로가 도출되면 이 최적경로를 선분단위로 분할하여 각 선분의 끝위치가 최적경유필수위치가 되게 해야 한다. 이렇게 생성된 최적경유필수위치는 다음과 같은 성질을 갖게 된다.

(1) 목표위치가 현위치와 같은 선분에 있는 경우

목표위치가 현위치와 같은 선분에 있으면 최적경유필수위치는 목표위치와 동일하게 된다.

(2) 최적경로가 선분 n개로 구성되는 경우

최적경로가 선분 n개로 구성되면 최적경유필수위치는 n개(각 선분의 끝)가 되며 경로표에는 이중 최초의 것을 나타내는 것이다.

(3) 목표위치도 최적경유필수위치에 해당

목표위치도 마지막 최적경유필수위치에 해당한다고 볼 수 있다.

2.5 에이전트의 이동 알고리즘

에이전트는 자신의 위치를 알 수 있으며, 목표위치도 기억하고 있으며, 상, 하, 좌, 우 네 방향으로만 이동할 수 있는 환경을 고려하기로 하자. 에이전트가 현위치에서 목표위치로 이동하기 위하여 다음과 같은 알고리즘을 이용한다.

이동알고리즘

① 현위치가 목표위치인지 확인한다. 목표위치면 단계⑤로 간다.
② 현위치의 경로표에서 목표위치에 대한 최적경유필수위치를 가져온다.
③ 현위치에서 최적경유필수위치로 이동하고 현위치를 최적경유필수위치로 갱신한다.
④ 단계 ①로 간다.
⑤ 종료한다.

즉 에이전트는 현위치에서 목표위치로 이동하기 위하여 현위치에 대한 경로표에서 목표위치의 최적경유필수위치를 찾아서 최적경유필수위치로 이동한다. 현위치에서 최적경유필수위치로의 이동을 위한 경로는 실행시간에 대한 부하없이 좌표에 의한 1회의 계산에 의해 바로 결정될 수 있다. 에이전트는 이 최적경유필수위치를 새로운 현위치로 간주하며 목표위치에 도달했는지 확인하여 목표위치가 아니면 다시 목표위치로의 이동을 경로표를 이용하여 반복적으로 수행하게 된다.

2.6 경로표의 생성

경로표를 작성하기 위해서는 A* 알고리즘[3]을 함수로 이용하여 다음과 같은 과정을 수행하였다.

경로표작성 알고리즘

```
node = 지도상의 처음 지점
do
{
  node1 = 지도상의 처음 지점
  do
  {
     path[] = AStarSearch(node,node1)
     length = LengthOfPath(path[])
     pathlength = 1
     while(IsParallel(node,path[pathlength]))
     { pathlength++ }
     pathlength --
     Route-Table[node][node1] = path[length]
     node1 = 지도상의 다음 지점
  }until (node1 != 지도상의 마지막 지점)
  node = 지도상의 다음 지점
} until(node != 지도상의 마지막 지점)
```

AStarSearch는 주어진 첫 번째 노드와 두 번째 노드 사이의 최적경로를 배열로 반환하는 함수이다. LengthOfPath는 주어진 경로의 길이를 반환하는 함수이다. ShortestPath는 주어진 첫 번째 노드와 두 번째 노드 사이에 장애물이 없다고 가정했을 때 최단거리(대각선은 제외)를 반환하는 함수이다. IsParallel은 노드의 x좌표나 y좌표가 같으면 *true*을 반환하고 아니면 *false*를 반환하는 함수다.

2.7 경로표의 예

다음과 같이 위치 9개를 갖는 간단한 지도를 가정하자. 여기서 5, 8번 위치는 접근할 수 없는 곳이라고 가정하자. 이에 대한 경로표는 5, 8번 위치를 제외한 모든 위치에 대하여 작성해야 한다. 현위치가 1인 경우 경로표는 <표 3>에 나타낸 바와 같다. 예를 들어 목표위치가 9일 경우 최적경로는 (1, 2, 3, 6, 9)가 된다. 이런 최적경로에서 위 알고리즘에서는 이를 선분단위로 구분하여 나타낸다. 즉 1→3→9로 연결되는 두 개의 선분이 최적경로이므로 최적경유필수위치는 3과 9이다. 경로표에서는 이 중 첫 번째 최적경유필수위치만 보관한다. 또 다른 예로 <표 4>의 경우를 보자.

〈표 2〉 지도 예 1

1	2	3
4	5	6
7	8	9

〈표 3〉 현위치가 1인 경우 경로표 예

목표위치	최적경유필수위치
2	2
3	3
4	4
6	3, 6
7	7
9	3, 9

〈표 4〉 지도 예 2

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99

<표 4>에 나타낸 지도에서 색이 칠해진 부분은 블록영역이다. A* 알고리즘에서 <표 5>에 나타낸 최적경로를 구하면 이를 선분단위로 분해하여 최적경유필수위치 53, 59, 9, 5를 구할 수 있다. 이 중 첫 번째 위치를 경로표에 보관한다.

〈표 5〉 경로표 예 2

현위치3, 목표위치5인 경우	
최 적 경 로	3,13,23,33,43,53,54,55,56,57,58,59,49,39,29,19,9,8,7,6,5
최적경유필수위치	53, 59, 9, 5

2.8 경로표의 크기

가로 크기가 N이고 세로 크기가 M인 지도에서 각 지점을 표시하는 인덱스는 $\log_2^{N*M}$의 bit를 사용하여 표현할 수 있게 된다. 그러므로 각 경로표는 $\log_2^{N*M}$*N*M의 bit에 해당하는 크기를 가지게 된다 또 지도의 모든 지점은 경로표를 갖게 되므로 경로표는 $(N*M)^2$* $\log_2^{N*M}$에 근접한 크기를 갖게 된다.

3. 경로표의 압축처리

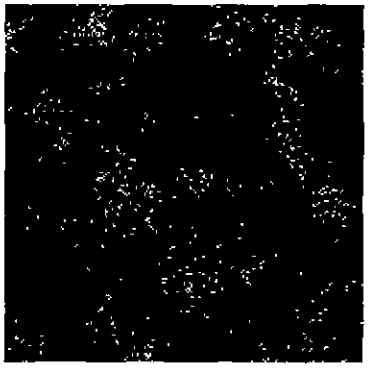
지도상의 경로표들은 상당 부분 중복될 수 있다. 따라서 각각의 경로표에 현위치 기준으로 인덱스를 붙이고 중복된 경로표를 생략한 후 지도상의 각 지점들은 경로표의 인덱스를 보유하게 함으로서 경로표 개수를 줄여 데이터압축효과를 기할 수 있다.

3.1 압축효과의 실험

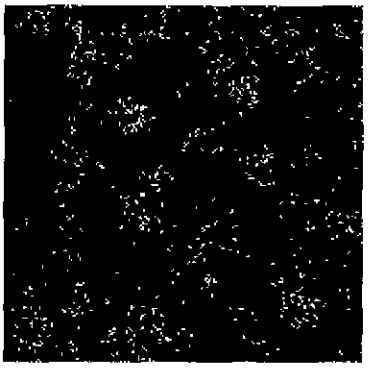
얼마만큼의 압축 효과가 발생하는 지를 알아보기 위해 여러 가지 데이터를 사용해서 실험해 보았다. <표 2>, <표 4>에서 회색으로 나타낸 영역이 통과할 수 없는 블록(Block)된 영역이다. 압축효과를 실험하기 위하여 주어진 비율만큼의 블록영역을 갖는 임의의 지도를 생성하는 프로그램을 만들었다. 이 프로그램을 이용하여 지도크기와 지도상에서 블록된 영역의 비율을 변경해 가며 데이터 압축후 몇 개의 경로표가 생성되는 지를 검사하였다.

(그림 1)과 (그림 2)는 이렇게 생성된 지도인데 모두 100*100 픽셀이며, 블록영역이 각각 5%, 10%인 경우

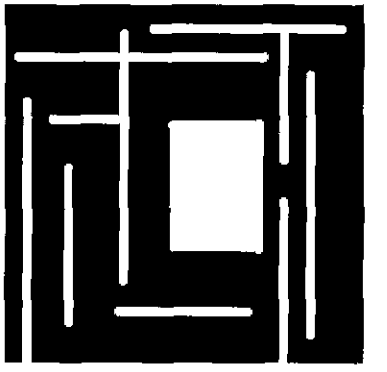
에 해당한다. <표 6>에 나타낸 16가지 각 경우에 대하여 30회를 실험하였다. 이 실험의 평균결과는 <표 6>에 나타낸 바와 같다. 이 표의 세로축은 지도의 가로와 세로 크기를 나타내며 가로축은 지도상에 포함되어 있는 블록영역의 비율을 나타낸다. 셀 내부는 해당 지도상에서 생성되는 경로표의 개수를 나타낸다. 이 표에서 ()의 수치는 압축율을 의미한다 위 실험 결과 데이터 압축효과는 블록영역 비율이 적을수록 크게 나타남을 알 수 있다 그런데 실제 게임에서는 블록된 영역이 (그림 1), (그림 2)에서와 같이 랜덤하지 않고 일반적으로 연속적으로 나타나게 된다. (그림 3), (그림 4)에는 YS-Eternal 게임[6]에서 나타나는 지도를 보여주고 있는데 실제 게임에서는 블록된 영역이 이와 같이 연속적으로 나타나게 된다. 이를 랜덤 지도 발생기에 반영시켜서 다시 한번 실험한 결과는 <표 7>에 나타낸 바와 같다.



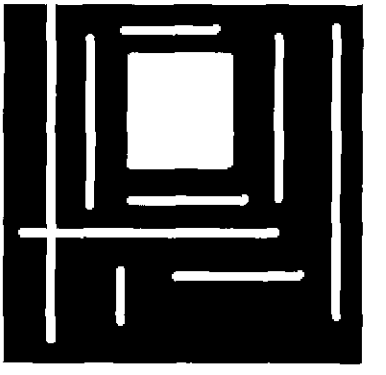
(그림 1) 랜덤 지도 1



(그림 2) 랜덤 지도 2



(그림 3) 비디오 게임 지도 1 (흰색이 블록영역)



(그림 4) 비디오 게임 지도 2 (흰색이 블록영역)

〈표 6〉 랜덤지도에서 경로표의 압축비율

단위 : 픽셀	5%	10%	20%	30%
10*10	27(0.73)	42(0.58)	64(0.36)	72(0.28)
100*100	2453(0.75)	4319(0.57)	7826(0.22)	6453(0.36)
1000*1000	239875(0.76)	421821(0.58)	812928(0.19)	738392(0.26)
10000*10000	21364553(0.79)	10782117(0.60)	75378938(0.25)	72846383(0.28)

〈표 7〉 블록된 영역이 연속적인 경우 경로표 갯수

	5%	10%	20%	30%
10*10	4(0.94)	8(0.92)	13(0.87)	14(0.86)
100*100	12	20	32	38
1000*1000	133	232	332	379
10000*10000	342	529	820	2127

<표 7>의 경우 압축율은 거의 100%에 가깝게 나타났다 <표 6>과 <표 7>을 비교해보면 일반적으로 게임분야에 사용되는 지도의 경우에는 훨씬 적은 메모리 공간을 사용하여 경로표를 저장할 수 있음을 알 수 있다 만약 경로표를 사용하지 않고 최적경로를 그대로

저장하는 경우에는 현위치와 목표위치가 모두 다르기 때문에 모든 경로가 다르게 된다. 즉 이와 같은 압축 방법을 사용할 수 없게 된다.

4. 경로표의 성능 측정

4.1 성능비교 방법

경로표를 사용했을 경우의 속도 향상을 A* 알고리즘[3]을 대상으로 하여 실험하였다. 실험에 사용된 시스템은 Intel 계열 PC상에서 C언어를 사용하여 구현하였으며 블록된 영역이 연속적으로 나타나도록 수정된 랜덤 지도 작성기를 사용하여 추출한 서로 다른 크기를 갖는 4개의 맵에서 각각 10, 20, 30, 40개의 에이전트가 동시에 임의의 출발점에서 임의의 목적지에 도달하는 시간을 측정하였는데 지도맵의 크기만큼 실험을 반복하였다. 단 에이전트 여러 개를 이동하는 경우 서로 충돌할 수 있는데 A*의 경우는 충돌여부를 무시하고 이동시켰으며, 경로표 방법에서는 이동위치에서 충돌이 발생하는 지 미리 구간별로 검토하여 충돌이 발생하는 경우는 에이전트의 인덱스 순으로 이동을 시켰다.

4.2 성능비교결과

〈표 8〉 10*10 지도의 경우

(단위 : ms)

	A*	경로표
10 에이전트	0.8	0.3
20 에이전트	1.2	0.3
30 에이전트	1.7	0.4
40 에이전트	2.0	0.4

〈표 9〉 100*100 지도의 경우

(단위 : ms)

	A*	경로표
10 에이전트	12.3	0.3
20 에이전트	23.6	0.3
30 에이전트	37.2	0.3
40 에이전트	49.2	0.4

〈표 10〉 1000*1000 크기 지도의 경우

(단위 : ms)

	A*	경로표
10 에이전트	324.2	1.2
20 에이전트	578.8	1.3
30 에이전트	812.6	1.3
40 에이전트	1311.1	1.5

〈표 11〉 10000*10000 지도의 경우

(단위 : ms)

	A*	경로표
10 에이전트	3021.3	2.1
20 에이전트	6103.2	3.2
30 에이전트	8934.8	4.0
40 에이전트	10301.2	4.9

성능비교결과를 각 지도 크기별로 <표 8>, <표 9> <표 10>, <표 11>에 각각 나타냈다. 이 표에 나타낸 수치는 A*를 이용한 경우와 경로표를 이용한 경우에 이동에 소요된 시간의 산술평균을 나타낸다. 맵이 작은 경우는 큰 차이가 없지만 맵이 커질수록 많은 차이를 보이고 있다. 이 소요시간은 실시간 시스템의 구현에서 이는 매우 중요한 요소이다. 이를 통해 경로표를 이용한 시스템이 최적 경로를 이동하는 데 매우 우수한 수행 시간을 갖고 있고, 특히 실시간 시스템을 구현하는 데 있어 장점을 갖고 있다는 것을 알 수 있다.

5. 결론 및 향후 연구 방향

정적인 블록영역을 갖는 비디오 게임에서 경로표를 이용한 경로 탐색은 미리 정해진 지도 상에서 많은 에이전트들이 실시간으로 경로탐색을 수행해야 하는 경우 우수한 수행성능을 보이는 시스템이다. 이는 특히 비디오 게임 분야에서 많은 응용이 가능하리라고 예측된다. 그러나 시스템이 경로표를 보유하고 있어야 한다는 문제점을 갖고 있기에 게임지도가 커지면 경로표로 인하여 수행 속도가 저하되리라는 것을 예측할 수 있다. 그래서 현위치를 기준으로 경로표가 일치하는 경우 이를 압축하여 저장하여 이 문제점을 완화할 수 있음을 보였다. 그리고 또 많은 경우에 목표위치와 경유위치가 일정 구간 계속해서 일치하는 부분이 있다. 이러한 부분을 생략하는 자료 구조를 사용한다면 경로표의 크기는 크게 압축될 수 있을 것으로 생각된다.

본 연구에서는 4방향에 대해서만 적용가능한 경로표를 제안하였지만 최근의 게임에서는 8방향, 16방향, 32방향도 이용하는 추세이므로 이런 경우에 이용가능하도록 좀더 일반화된 경로표가 요구된다고 하겠다. 또한 실제 게임을 구현하면 정적 블록영역의 경우에도 에이전트가 존재하는 위치에 대해서는 동적인 블록영

역이 되는데 이의 처리에 대해서도 정밀한 처리방법이 보완되어야 하겠다. 그리고 경로표의 데이터들을 좀 더 압축시킬 수 있는 방향에 대한 연구가 필요하며 많은 실험을 통해서 지도의 크기와 경로표의 크기 사이의 함수 관계를 밝혀 내는 것도 병행되어야 하겠다

참고 문헌

[1] Command and Conquer - Red Alert , WestWood사, 미국, 1996

[2] Dune2, WestWood사, 미국, 1993

[3] Nilsson, N J., Problem-Solving Methods in Artificial Intelligence, McGraw-Hill, New York, 1971

[4] Stuart, Russel and Peter Norvig, Artificial Intelligence, Prentice-Hall, 1995

[5] War Craft 2-Tides of Darkness, Blizzard사, 미국, 1995

[6] YS Eternal, Falcom사, 일본, 1987

심 동 희

e-mail : dhshim@jeonju.ac.kr

1980년 서울대학교 산업공학과 졸업(학사)

1982년 서울대학교 대학원 산업공학과 졸업(공학석사)

1994년 고려대학교 대학원 전산과학과 졸업(이학박사)

1982년~1985년 국토개발연구원

1985년~1990년 해양수산개발원

1990년~현재 전주대학교 전자매체공학부 컴퓨터공학 전공 부교수

관심분야 : 기계학습, 인터넷 보안, 인터넷 게임

강 혁

e-mail : DEV666@chollian.net

1998년 전주대학교 컴퓨터공학과 졸업(학사)

1999년~현재 전주대학교 대학원 컴퓨터공학과 석사과정

관심분야 : 컴퓨터 게임, 인터넷보안