RTS 게임의 협곡에서 일어나는 패싱 알고리즘의 문제와 해결

이한혁° 이만재
아 주 대 학 교 컴 퓨 터 공 학 과
{bbpanda}@madang.ajou.ac.kr

Solving canyon pathing problem on RTS game

Han-Hyouck Lee° Manjai Lee
Dept. of Computer Engineering, Ajou University

요 약

원활한 RTS 게임의 진행을 위해서 경로 찾기의 역할은 절대적이다. 경로 찾기는 게임 유닛의 진행이나 행동에 따라 무한 발생하며 게임의 특성에 따라 변하기 쉽기 때문에 유연해야 한다. 이는 게임의 자유도를 높이는데 중요한 요소가 된다. 그러나 경로 찾기는 맵의 속성이나 환경 또는 사용자의 요구에 따라 행동 패턴들이 무한 발생하므로 체계적 알고리즘을 만들기가 어려워 많은 문제를 일으킨다. 기존의 게임에서는 알고리즘이 맵 상의 협곡에서 속성 처리를 제대로 하지 못하였다. 본 논문에서는 대기모드의 추가를 통한 해결에 방안을 제시 한다.

1. 서론

지금까지의 게임은 많은 개발과 진보를 이룩하였다. 그 결과 많은 게임의 장르가 나오게 됐으며 "Real - Time Strategy Game"(이하 RTS)또한 그 중 하나이다. 이 장르는 최근에 나온 장르답게 가장 진보된 알고리즘과 게임 환경을 지원한다. 그것은 여러 장르의 복합이며 실시간에 네트워크를 기반으로 하게 되어있어 현 시대적 대중적 관심과 맞아 떨어졌고 지금은 가장 각광받는 장르로 급 부상하였으며 여러 파생적 장르와 게임을 만들어 내었다. 현재 많은 개발자들이 선호하고있으며 다른 장르의 게임들 또한 영향을 받고있는 상태에 이르고있다. 그러나 문제가 없지만은 않다. 가장 진보된 알고리즘과 개발 환경을 선호하나 그만큼 문제의 부분도 상당히 난해한 면을 보이며 개발 또한 쉽지는 않다. 여러 문제 중에 대표적인 예로 유닛 증가에 따른 네트워크의 시간 지연은 여러 개발자들과 게임들이 이 장르를 선호했으면서도 참패를 면치 못하게 한 점이다. 또 한가지 알고리즘 적인 문제로는 협곡에서의 문제가 있는 데 이는 게임의 맵에서 유닛들간의 병목현상으로 인해 유닛들 자신이 해야 할 일을 하지 못하고 무한 루프에 빠지는 여러 현상들이다. 이 문제는 특히 유닛의 공격시 뜻하지 않은 행동으로 게임의 발란스를 깨트려서 게임의 질을 떨어뜨린다.

본 논문은 그 문제중의 하나인 협곡에서의 경로 탐색 시에 일어나는 문제를 해결하려 한다. 이 문제는 협곡에서 유닛들의 충돌 시 또는 적과 아군의 공방이나 장애물이 있는 곳에서 그룹후미의 캐릭터들이 탐색 에러를 보임으로써 대기하여 공격하지 않고 엉뚱한 곳에 탐색 이동하게 되는 문제를 말하는 것이다. 여기서는 이 문제를 해결하고 더 효율적인 협곡에서의 탐색 이동이 되는 알고리즘을 구현한다.

본 논문은 제2장에서 문제의 이해를 돕기 위해 게임에서의 경로 탐색 이동의 기본 과정을 간략히 소개하고, 과정 시 사용하는 알고리즘과 협곡에서의 유닛 이동 알고리즘의 문제를 좀더 자세히 살펴본다. 제3장에서는 협곡 에서의 유닛 이동 문제의 해결 방법과 알고리즘을 구현한다. 제4장에서는 해결하지 못한 문제점과 해결방향 새로운 문제 제기로써 결론을 맺을 것이다.

2. 기존 패싱 알고리즘

2.1 패싱(Pathing)

패싱은 RTS 게임에 있어서 맵 상에서 유닛들의 이동시 절대경로를 계산, 효율적인 이동을 하는 행위를 말하며 게임의 전반적인 속도와 안정성에 절대적인 영향을 준다. 그 절차는 아래와 같다.

1)A^* 알고리즘을 이용한 first search(절대경로 탐색)
2)버퍼에 절대경로 및 기타 유닛속성 저장
3)버퍼의 시작점으로부터 다음 포인터의 맵 탐색을 하여 속성 값이 존재하면 그에 상응하는 행동을 취한다.
4)속성 값이 없거나 행동이 끝나면 다음 포인터로 버퍼 포인터가 증가하며 유닛을 이동시킨다.
5) 버퍼의 지난 포인터의 메모리를 지운다. 다시 단계 3으로 돌아가서 버퍼 값이 없을 때까지 진행한다.

단계 1은 A^* 탐색을 이용한 최단거리의 유닛이 이동할 자연스러운 이동경로(절대경로)를 얻는 것이다. 여기서 A^* 알고리즘을 사용하는 것은 인공지능에서 말하는 것으로 그래프 방식을 통한 최단 거리를 구할 때는 가장 효율 적이기 때문이다. 단계 2는 탐색한 절대경로와 움직일 유

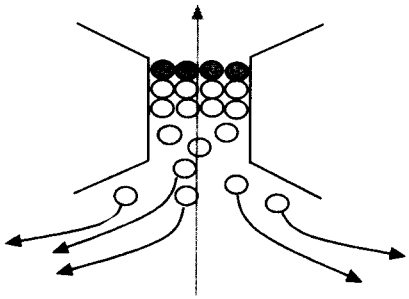
닛과 그 수를 버퍼에 기억시키는 일이다. 단계 3과 4는 탐색한 경로에 따라 한 단계씩 이동하는 대 그 전에 맵 상의 그 위치에 다른 유닛이나, 적 또는 그 외의 상황을 체크하여 그에 맞는 행동을 하거나 아무 이상도 없을 경우 다음 포인터로 이동하는 것이다. 여기서 탐색이란 유닛의 경로에 장애물이나 돌발상황이 발생할 경우 원활한 이동을 위한 행동 알고리즘을 말한다. 단계 5는 이동 후에 필요 없게된 이전의 절대경로 버퍼를 지우는 행동이다. 이때 버퍼 값이 없으면 경로 찾기 절차를 끝내고 그렇지 않으면 단계 3으로 돌아가 버퍼 값이 없을 때까지 반복하여 이동을 완료한다.

2.2 협곡에서의 패싱 문제

협곡에서의 패싱 문제는 협곡에서 패싱시 에 그룹의 후미에 위치하는 유닛들이 장애물이나 돌발상황으로 인해 자신의 원래 행동을 하지 못하고 탐색 이동하여 엉뚱한 곳에 가있는 문제를 말한다 그림1을 보면 알 수 있다. 여기서 협곡이란 게임의 맵 상에서 유닛이 특정 목적지까지 이동하는 중에 존재하는 협소한 외길들을 말한다.

이는 패싱 알고리즘 자체의 오류로써 협곡에서 각 유닛들이 맵 탐색시 전방에서 공격하는 유닛들이나 장애물로 인해 이동해야할 절대 경로가 막힘으로써 계속적인 무한 탐색 루프에 빠져 이동 할 수 있는 빈 공간을 찾아가다 목표가 아닌 예상치 못한 곳으로 이동하게 되는 것이다. 이럴 경우에 무한 루프에 빠지게 되지만 기존의 게임에서는 이런 경우를 대비하여 처음 패싱시 에 일정 루프이상 알고리즘이 돌았을 경우 자동적으로 해제되는 속성 값을 주어 무한 루프에는 빠지지 않게 하는 것이다.

본 문제는 후미 유닛에만 적용되는 문제가 아닌 직접적인 충돌에 들어간 유닛을 제외한 전 유닛에 일어나며 다만 후미유닛을 제외한 유닛들은 이동 할 수 있는 빈 공간이 존재하지 않으므로 대기하는 것처럼 보일 뿐이다. 이러한 문제는 특히 공격 이동시에는 절대적 변수가 되며 같은 수의 유닛으로도 먼저 파괴되어 상대 편 에게 쉽게 무너지는 게임에 밸런스를 흩트려 트리 는 요소가 된다.



적 유닛 또는 장애물 ○아군 유닛 ◀— 절대경로

<그림 1> 협곡에서의 패싱 문제

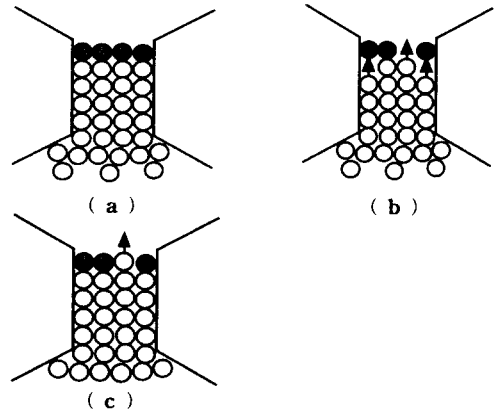
3. 협곡에서의 새로운 패싱 알고리즘

여기서 제시하는 방법은 주로 공격 이동 모드일 때 가장 유효하며 방법은 협곡에서 문제의 발생 시 유닛들이 이동을 할 수 없음을 알았을 때 탐색을 통한 액션을 주어 활로가 생길 때까지 대기를 하다 점차 전진하는 것이다.

3.1 해결방법

기존의 탐색모드 에서는 절대경로가 막힐 경우 이동방향을 중심으로 시계방향이나 또는 그 반대 방향으로 8방향 탐색하여 방향에 상관없이 어떤 빈 공간으로도 이동하게 하였다. 그림 1과 같이 그 문제를 보였다.

본 논문에서는 탐색의 취해야할 행동 중 대기를 추가하여 원치 않는 곳으로 이동하는 것을 막고 이동 할 수 있을 때까지 대기하도록 함으로써 그 문제를 해결하였다. 특히 이 방법은 공격 이동시에 효과적이며 그것은 적이나 파괴가 가능한 장애물을 만났을 경우 그림 2와 같이 대열을 흩트리지 않고 점진적으로 공격하여 전진 이동하기 때문에 아군 유닛의 공격력을 분산키지 않는 이유에서이다. 또 파괴가 불가능한 경우의 장애물을 만나면 협소한 공간이라도 차례차례 진행해 나갈 수 있다. 만약의 경우 절대경로가 막힌 경우에는 다음 명령이 있을 때까지 대기 할 수 있다.



<그림 2> 대기모드를 추가한 협곡에서의 패싱

3.2 새로운 알고리즘

본 논문에서 제시하는 알고리즘은 다음과 같으며 패싱의 단계 3에서 탐색이 발생하였을 경우부터 진행된다.

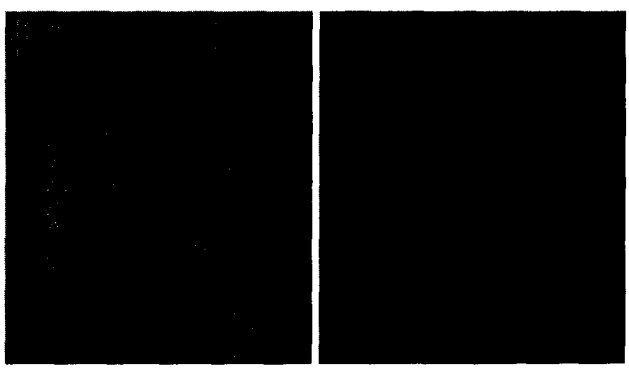
1)탐색 값의 속성 중 맵 속성이 협곡인지를 확인한다.
2)움직여야할 방향을 중심으로 오른쪽에서 왼쪽으로(또는 왼쪽에서 오른쪽으로) 3방향 탐색한다.
3)진행 방향이 막혔을 경우 절대경로를 다시 계산하여 새로운 절대 경로가 있는지를 확인, 발생할 경우 그 방향으로 패싱을 다시 시작한다.
4)새로운 절대경로가 없을 경우 다음 탐색까지 대기한다.
5)탐색 액션을 통한 공간이 발생하면 그 곳으로 이동하여 패싱을 계속 수행한다.

```
Scan_mode(unit_attribute, *scan_points, map_attribute)
  while(scan_points)
    if(next_points_scan(*scan_points, map_attribute))
      scan_mode ← next_points_scan_value
    switch(scan_mode)
      case unit_attack;
```

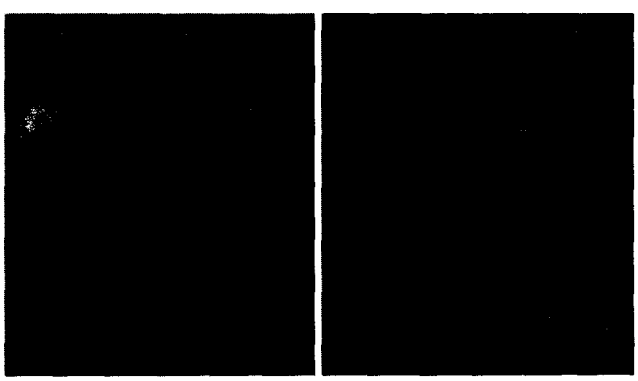
```
        .....
        case unit_wait;
      end switch
    end if
  end while

Next_point_scan(*scan_points, map_attribute)
  switch(map_attribute)
    case level_ground
      .....
    case canyon
        if(three_move_scan(*scan_points)==move_not_point)
      if(first_search())
        scan_out_repathing();
      end if
    end if
    next_points_scan_value ← unit_wait;
  end switch
```

3.3 실험 결과



(a) (b)



(c) (d)

<그림 3> 실험 결과

실험은 그림 3과 같이 텍스트모드의 2차원 맵에서 8방향 이동 패턴의 같은 속성, 속도의 10개 유닛으로 테스트 하였다. 목표지점은 처음 시작 위치의 대각선 반대방향으로 정했으며 맵은 일반적 패싱을 할 수 있는 구조와 협곡에서의 문제 구조를 섞었다.

그림3의 c를 보면 아군 유닛이 장애물을 만났을 경우 대열을 지키며 이동하는 것을 볼 수 있다.

4. 결론

RTS 게임에서는 패싱은 가장 중요한 요소이다. 여기서 해결한 문제는 아주 단편적인 방법으로 해결하였으나 아직 문제가 없지 않다. 본 논문에서 살펴본 패싱은 유닛들이 공격형 패싱 일 때 특히 유효한 것으로 전반적인 패싱 에서 는 큰 효과를 보지 못한다. 또 유닛들간의 속성 차가 있을 경우 문제가 발생 할 수 있으며 유닛의 대기 자체가 역효과를 불러일으킬 수도 있다. 그러나 만약 패싱 자체가 상당히 고도화 될 경우 이런 문제들은 자연스럽게 해결 될 수 있다. 앞으로의 과제는 단편적 문제 해결보다는 더욱 부드럽고 보편화 될 수 있는 패싱 알고리즘의 개발이며 병렬 적 패싱 알고리즘의 개발을 통한 유닛 에이전트 기능의 확대에 있을 것이다.

감사의 글

본 논문은 1999 학년도 아주대학교 연구비 지원에 의하여 연구되었음.

참고 문헌

[1] Dave C. Potinger, "Implementing Coordinated Movement", Game Developer Magazine, February, 1999, PP 48-58.

[2] Dave C. Potinger, "Coordinated Unit Movement", Game Developer Magazine, January 1999, PP 42-51.

[3] W. Bryan Stout, "Smart Moves: Intelligent path-Finding", Game Developer Magazine, October 1996, PP 28-35.

[4] Mickey Kawick, Real-Time Strategy Game Programming Using MS directx 6.0, Wordware Publishing, 1999.

[5] 이동윤, 고욱, "조직대형을 이용한 그룹 경로 찾기", HCI 2000 HCI, CG, VR 학술대회 논문지, 2000, PP 952-956.

[6] Andren LaMothe, Tricks of the Windows Game Programming Gurus, SAMS, October, 1999.