Journal of Korea Game Society 2015 Aug; 15(4): 29-38
http://dx.doi.org/10.7583/JKGS.2015.15.4.29

게임기획 및 디자인

스텔스 게임 레벨 디자인 툴의 개선

나현숙, 정상혁, 정주홍
숭실대학교 컴퓨터학부
hsnaa@ssu.ac.kr, jsh0246@naver.com, growingdever@gmail.com

Improving A Stealth Game Level Design Tool

Hyeon-Suk Na, Sanghyeok Jeong, Juhong Jeong
School of Computer Science and Engineering, Soongsil University

요 약

스텔스 게임 레벨 디자이너는 다양한 난이도의 흥미로운 게임환경(레벨)을 제작해야 한다. J. Temblay와 공동 연구자들은 이 과정의 자동화를 돕는 Unity-기반 레벨 디자인 툴을 개발했다. 이 툴은 디자이너가 지도에서 경비병의 경로, 속도, 감시 영역, 플레이어의 출발점과 도착점 등 여러 게임 요소들을 입력하면 플레이어가 취할 수 있는 가능한 경로들을 포함한 다양한 시뮬레이션 결과들을 보여준다. 이를 이용해 디자이너는 현재의 게임 요소들이 자신이 의도한 난이도 및 플레이어 경로를 만드는지 실시간으로 확인할 수 있고, 필요한 경우 이들을 조정할 수 있게 되었다.

여기서는 두 가지 면에서 이 툴의 개선점을 제시한다. 첫째, 디자이너가 몇 개의 지점을 입력하면 이 지점들을 포함하는 흥미로운 경비병의 감시 경로를 난이도별로 추천해주는 기능을 추가해서 레벨 디자인 툴로서의 편의성과 유용성을 높였다. 둘째, 기존의 충돌 체크 함수 및 RRT-기반 경로 탐색 함수를 새로운 충돌 체크 함수와 델로네 로드맵-기반 경로 탐색 함수로 대체하여 시뮬레이션 속도를 크게 향상시켰다.

ABSTRACT

In the stealth game design, level designers are to develop many interesting game environments with a variety of difficulties. J. Tremblay and his co-authors developed a Unity-based level design tool to help and automate this process. Given a map, if the designer inputs several game factors such as guard paths and velocities, their vision, and the player's initial and goal positions, then the tool visualizes simulation results including (clustered) possible paths a player could take to avoid detection. Thus with the help of this tool, the designer can ensure in realtime if the current game factors result in the intended difficulties and players paths, and if necessary adjust the factors.

In this note, we present our improvement on this tool in two aspects. First, we integrate a function that if the designer inputs some vertices in the map, then the tool systematically generates and suggests interesting guard paths containing these vertices of various difficulties, which enhances its convenience and usefulness as a tool. Second, we replace the collision-detection function and the RRT-based (player) path generation function, by our new collision-check function and a Delaunay roadmap-based path generation function, which remarkably improves the simulation process in time-efficiency.

Keywords : Game level design(게임 레벨 디자인), First-Person Shooter game(FPS 게임), Collision detection algorithm(충돌 감지 알고리즘)

Received: July, 10, 2015 Accepted: Aug, 10, 2015
Corresponding Author: hsnaa@ssu.ac.kr(Soongsil University)
E-mail: hsnaa@ssu.ac.kr

ISSN: 1598-4540 / eISSN: 2287-8211

1. 서 론

스텔스 게임이란 FPS(First-Person Shooter) 게임 장르의 한 분야로서 대규모 전투보다는 은신하여 비밀공작, 암살 등의 특수목적을 달성하는 게임을 말한다. 레벨 디자이너는 플레이어가 재미와 성취감을 느낄 수 있도록 다양한 난이도의 게임환경(레벨)을 제작하는데, 경비병의 감시 경로나 속도, 장애물 위치 등이 난이도 조절의 주요 요소이다. 그러나 제작한 레벨이 의도했던 난이도와 플레이어의 움직임을 유도하는지 실제 플레이어를 통해 검증하는 것은 많은 시간과 비용이 필요하다. 따라서 다수의 플레이어가 취할 경로를 탐색하여 보여줌으로써 레벨을 검증 또는 수정하도록 돕는 레벨 디자인 툴의 개발은 최근 게임 분야에서 많은 관심을 받고 있다.

스텔스 게임 레벨 디자인 툴은 Tremblay와 공저자들에 의해 제작, 연구되었는데[1,2,3,4,5], RRT 알고리즘을 이용, 다수의 플레이어의 행동을 시뮬레이션하여 레벨을 분석할 수 있도록 Unity 기반으로 제작되었다. RRT(Rapidly-exploring Random Tree) 알고리즘은 LaValle[6]에 의해 제안된 로보틱스 분야의 대표적인 경로 탐색 방법이다. 한 쌍의 출발점과 도착점에 대해 무작위 샘플링을 통해 경로를 확장하며, 다수 유저들의 움직임을 생성하고 시뮬레이션하는 데 적합하지만, 수많은 무작위 샘플링과 최단 거리 이웃탐색 때문에 속도가 느리다는 단점이 있다[7].

본 논문은 두 가지 면에서 이 툴을 개선하는 방법을 제시한다. 첫째, Xu 등[4,5]의 아이디어를 발전시켜, 제작자에게 유용하고 흥미로울 가능성이 있는 경비병 감시 경로들을 미리 계산해 놓고, 디자이너가 몇 개의 지점을 입력하면 이 지점들을 포함하는 감시 경로들만 이 집합으로부터 난이도별로 추출해 보여주는 기능을 추가한다. 이러한 기능은 디자이너가 레벨을 제작하는데 많은 편의성과 유용성을 제공할 것이다. 둘째, 플레이어의 다양한 경로들을 생성하고 시뮬레이션하는 데 사용되는 두 가지 주요 함수들 - 충돌 체크 함수 및 RRT-기반 경로 탐색 함수 - 을 새로운 충돌 체크 함수과 델로네 로드맵(Delaunay Roadmap)-기반 깊이 우선 탐색 함수로 대체하여 시뮬레이션 속도를 크게 향상시킬 수 있다. 2장에서는 기존 연구를 좀 더 자세히 소개하고, 3장과 4장에서는 두 가지 개선사항을 구체적으로 기술한 뒤, 실험을 통해 검증한다.

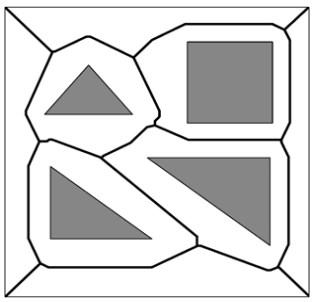
2. 관 련 연 구

Tremblay 등은 논문[1]에서 스텔스 게임에 RRT 알고리즘을 이용, 다수 플레이어의 행동을 시뮬레이션하여 레벨을 분석할 수 있도록 하는 Unity 기반 툴을 제시했다. 논문[2]에서는 전투나 체력회복제와 같은 게임 요소를 추가하여 기존의 연구를 확장하였고, 논문[3]에서는 플레이어 경로의 위험도를 경비병의 경로와의 거리, 경비병의 시야(Field Of View)과의 근접성, FOV를 얼마나 아슬아슬하게 피해가는가의 3가지 수치로 측정, 분석하는 방법을 연구했다.

이 툴을 이용한 플레이어 시뮬레이션 과정은 다음과 같다. 경비병의 감시 경로와 속도를 포함한 모든 게임요소가 정해지면 3차원 자유공간 $\mathbb{F}$(모든 시간에 대해 시간별 자유영역(감시 영역과 장애물 영역을 제외시킨 영역)의 합집합)을 배열 형태로 계산해 놓는다. RRT 알고리즘으로 플레이어 경로를 탐색하는데, 3차원 무작위 점을 뽑고 이미 건설한 경로 중 가장 가까운 점까지 연결한 선분이 자유공간 $\mathbb{F}$ 내부에 있는지, 즉 경비병의 시야 및 장애물과 충돌이 발생하는지 확인 후, 충돌이 없으면 연결하고, 충돌이 발생하면 다시 샘플링한다. 이러한 시뮬레이션 방식은 단위 연산인 충돌 체크 함수에서 사용할 자유공간 $\mathbb{F}$의 배열 계산에 너무 많은 시간과 공간을 소모하게 되어, 디자이너가 여러 요소를 수정한 후 그에 따른 플레이어 변화를 실시간으로 확인하고 조정하기에는 너무 느리다는 문제점이 있다.

주어진 지도에서 다양한 난이도의 감시 경로와 장애물을 생성하는 방법은 Shi 등[8]과 Xu 등[4,5]에 의해 연구되었다. Shi 등[8]은 정적인 경비병과 장애물을 배치하는데, 성공한 플레이어의 경로가 최소 피해량을 갖는 경우, 최대 길이가 되는 경우, 급격한 피해량 변화를 야기하는 경우 등을 다루었다. Xu 등[4,5]은 경비병의 감시 경로와 감시 카메라의 위치를 체계적으로 생성하고 분석하기 위한 방법론들을 연구했다. 논문[4]에서는 2차원 지도를 보로노이 영역(Voronoi Region)으로 분할한 뒤, 경비병의 경우 직선의 감시 경로를 갖고, 카메라의 경우 감시 범위를 최대화하도록 배치했다.

논문[5]에서는 감시 카메라의 위치를 삼각화(Triangulation)의 3-Coloring 기법[9]을 통해 생성하고, 경비병들은 자유공간의 미디얼 액시스(Medial Axis) 위에서 무작위로 선택한 경로를 왕복운동 하도록 했다. 여기서 자유공간의 미디얼 액시스란 2개 이상의 장애물로부터 같은 거리에 있는 자유영역의 점 집합인데([Fig. 1]), 3개 이상의 장애물로부터 같은 거리에 있는 점이 정점(vertex), 나머지 부분, 즉 정점과 정점 사이를 연결하는 선들이 간선(edge)이며, 장애물의 코너로 연결된 간선을 제외하고 모든 간선이 감시 경로의 후보가 된다. 특히, 문법적인 요소를 도입하여 감시 경로의 각 선분을 더 나누어 잠깐 멈춤, 회전 등과 같은 요소를 재귀적으로 삽입할 수 있도록 하였다. 하지만 이 논문에서 이미 지적된바, 무작위 경로 생성 방법은 레벨 디자이너가 원하는 난이도의 감시 경로들을 생성하기에는 적합하지 않다.



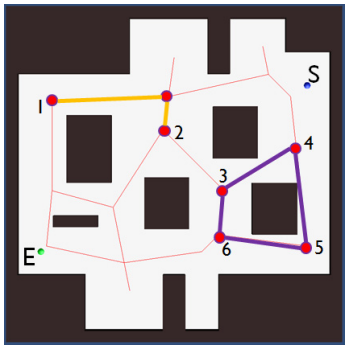
[Fig. 1] Medial Axis

본 논문에서는 Xu 등[4,5]의 아이디어를 발전시켜, 주어진 지도에서 개발자에게 유용한 경비병의 감시 경로 후보들을 모두 계산해 놓은 후, 개발자가 원하는 정점들을 입력하면 이 정점들을 포함하는 후보 경로들을 난이도별로 출력해주는 기능을 제안한다. 또한, 다수 플레이어 경로들을 생성하고 시뮬레이션하는 부분에서, 기존의 충돌 체크 함수와 RRT-기반 경로 탐색 함수 대신, 새로운 충돌 체크 함수와 로드맵-기반 깊이우선탐색(DFS)을 사용하는 방안을 제시한다.

3. 감시 경로 생성 및 플레이어 시뮬레이션

3.1 감시 경로 후보집합 Z 및 경로추천

주어진 지도에 경비병들의 감시 경로를 배치하는 방법은 다양하다. Xu 등[5]은 경비병의 감시 경로를 생성하는 방법으로 미디얼 액시스 상에서의 무작위 경로를 이용했는데, 이러한 무작위적인 경로 생성 방법은 레벨 디자이너가 원하는 형태 및 난이도의 감시 경로를 생성하기에는 적합하지 않다.



[Fig. 2] Map 1, and two types of guarding paths: Type 1(yellow), Type 2(purple)

여기서는 개발자에게 유용하고 흥미로운 감시 경로의 후보 집합 Z를 구하고(Step 1과 2), Z의 경로 중 원하는 형태의 경로들만을 난이도별로 추천하는 알고리즘(Step 3)을 제안한다. 디자이너는

주어진 지도에서 감시 경로 후보 집합 Z를 한번 계산해 놓고, 필요할 때마다 Step 3를 시행하여 원하는 형태 및 난이도의 감시 경로들을 선택, 시뮬레이션할 수 있다.

Step 1: 주어진 지도의 미디얼 액시스를 만들고, 이 그래프 상에서 다음 두 가지 타입의 감시 경로들을 모두 계산한다. 첫 번째 타입은 임의의 두 정점 사이를 (다익스트라 알고리즘으로 구한) 최단경로로 왕복운동 하는 경로이고, 두 번째 타입은 일정 길이 이하의 사이클(Cycle) 경로이다. 예를 들어, [Fig. 2]는 정점 1과 2 사이를 최단경로로 왕복운동 하는 타입1 감시 경로와, 정점 3, 4, 5, 6으로 이루어진 타입2 사이클 경로를 보여준다.

Step 2: Step1에서 얻은 감시 경로들을 조합하여 임의의 숫자의 경비병들의 감시 경로를 생성할 수 있다. 여기서는 세 명 이하의 경비병들에 대해 각 경로의 간선들이 겹치지 않는 경우만 다루고, 감시 경로의 출발점이 플레이어의 출발점을 감시하는 경우와 같은 극단적인 경우는 제외시켰다. 이렇게 구한 감시 경로들의 집합이 Z이고, 이들은 모두 유용하고 흥미로운 감시 경로의 후보가 됨을 알 수 있다.

Step 1과 2를 통해 지도 1에서 추출된 감시 경로 후보들의 개수(=$|Z|$)는 [Table 1]과 같다. 후보들의 개수는 경비병의 숫자가 늘어날수록 크게 증가하므로, 이들 모두에 대해 플레이어 움직임을 시뮬레이션하는 것은 많은 시간이 걸릴 뿐만 아니라 실질적으로 불필요하다. 따라서 Z안의 경로들 중 원하는 형태의 경로들만 뽑아 난이도별로 추출해내는 과정이 필요하다.

[Table 1] Number of guarding paths extracted from Map 1

No. of guards	No. of guarding paths
1	175
2	7869
3	159575

Step 3: 디자이너가 입력한 정점들을 포함하는 감시 경로를 Z에서 무작위로 일정 개수(여기서는 80개)만큼 추출한다. 추출된 감시 경로 각각에 대해 다음 절들에서 소개하는 플레이어 시뮬레이션을 시행, 난이도를 계산하고 난이도별로 구분하여 출력한다.

3.2 새로운 충돌 감지 알고리즘

레벨 디자인 툴의 주요한 용도는 입력된 게임 환경에 대해 플레이어들의 움직임을 시뮬레이션하여 해당 레벨이 원하는 수준으로 디자인 되었는지 그 결과를 분석할 수 있게 하는 것이다. 툴을 이용하면 개발자는 사람을 통해 직접 테스트할 때보다 빠르게 결과를 확인할 수 있기 때문에 좀 더 효율적으로 게임 요소들을 조정할 수 있게 된다.

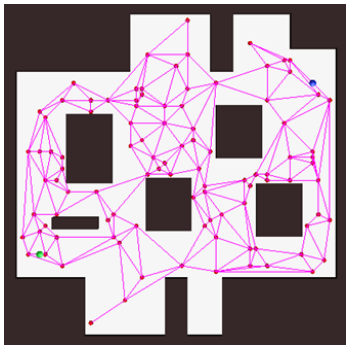
Tremblay 등[1,2,3,4,5]이 개발한 디자인 툴은 시뮬레이션의 단위 연산인 충돌 체크 함수에 필요한 자유공간 $\mathbb{F}$의 배열 계산에 너무 많은 시간과 공간을 소모하게 되어, 개발자가 게임 요소들을 변경한 뒤 빠르게 다시 결과를 확인하기엔 적합하지 않다.

이러한 문제점을 극복하기 위해 우리는 더 시간 효율적인 충돌 체크 함수를 제안한다. 감시 경로가 정해지고 나면 경비병의 시간에 따른 위치와 바라보는 방향을 계산할 수 있다. 이 데이터를 통해 시간에 따른 감시 영역을 구할 수 있으므로 t1에서의 감시 영역과 t2에서의 감시 영역을 이어 시간을 z축으로 하는 다면체를 만들 수 있게 된다. 이렇게 만들어진 다면체를 이용하여 플레이어의 시간별 위치로 만들어지는 선분과의 충돌 체크를 함으로써 경비병의 감시 영역에 플레이어가 들어갔는지 확인할 수 있다. 물론, 플레이어의 시간별 동선을 의미하는 이 선분은 장애물 영역에 들어가서도 안 되므로 장애물을 표현하는 다면체와의 충돌 여부도 검사한다. 이러한 방법을 통해 기존 방식보다 빠르게 충돌 검사를 할 수 있게 된다. 4장의 [Table 4]는 이 충돌체크 알고리즘을 사용함으로써 플레이어 시뮬레이션 시간이 크게 향상되었음을 보여준다.

3.3 델로네 로드맵 기반 플레이어 시뮬레이션

다양한 플레이어의 경로를 시뮬레이션하기 위해 RRT 알고리즘으로 경로 탐색을 할 경우, 수많은 무작위 샘플링 수행과 최단거리 이웃탐색에 많은 시간이 소요된다. 이 시간을 단축하기 위해서 미리 로드맵을 만들어 놓고 깊이우선검색(DFS) 방식의 탐색을 수행하면 RRT 알고리즘과 유사한 성공률을 보이되, 시간은 더 단축시킬 수 있다.

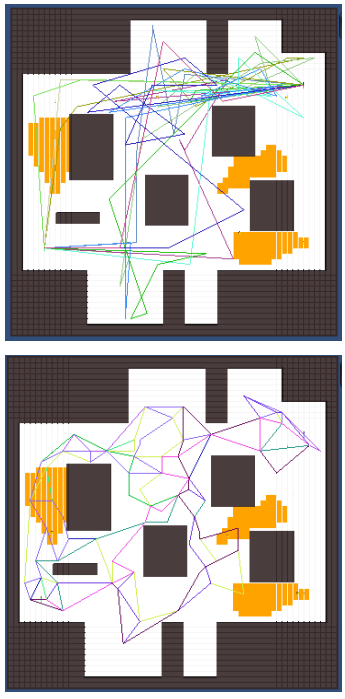
델로네 삼각화(Delaunay Triangulation)[9]는 주어진 정점들을 연결하는 삼각화의 한 종류로서 유클리드 최소신장트리(EMST)를 포함한다거나 삼각형들의 내각의 최소값이 최대가 되는 등의 최적의 성질을 많이 갖고 있다. 따라서 본 논문에서는 델로네 삼각화를 이용하여 경로 탐색에 사용되는 로드맵을 생성한다. 주어진 지도에서 장애물 위에 있지 않은 100개의 정점을 무작위로 샘플링하여 델로네 삼각화를 만든다. 장애물과 겹치는 간선은 제거하고 겹치지 않는 간선들로만 로드맵을 구성한다. 이 로드맵은 무방향그래프이고 [Fig. 3]은 지도 1에서의 델로네 로드맵의 예시이다.



[Fig. 3] Delaunay roadmap of Map 1

[Fig. 4]는 동일한 경비병의 감시 경로에 대해서 RRT 알고리즘을 이용해 경로 탐색을 한 경우와 델로네 로드맵을 이용해 경로 탐색을 한 경우의 성공한 경로들이다. 성공한 경로들은 위상적으로 유사하지만, RRT는 임의의 정점들을 지나는 경로들을, 델로네 로드맵은 로드맵 위의 경로들을 만들어낸다.

새로운 충돌 체크 함수와 델로네 로드맵을 이용한 시뮬레이션 방법을 정리하면 다음과 같다. 주어진 지도에서 델로네 로드맵(플레이어 경로)과 미디얼 액시스(경비병의 감시 경로)를 계산해 놓는다. 경비병의 감시 경로와 속도 등을 포함한 모든 게임요소들이 정해지면 충돌영역을 동적으로 만들기 위해 미리 시간에 따른 경비병의 위치와 바라보는 방향을 계산해둔다.



[Fig. 4] Successful player paths found by RRT algorithm(top) and by Delaunay roadmap(bottom)

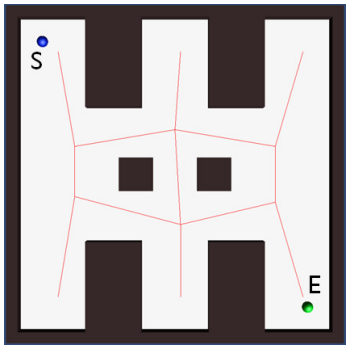
이 때, 모든 시간에 따른 감시 영역을 계산해두는 것은 오히려 비효율적이므로 그때그때 필요한 감시 영역을 동적으로 계산하고, 캐싱(Caching)해 둠으로써 차후 불필요한 계산을 줄인다. 델로네 로드맵에서 플레이어의 출발점 및 도착점과 가장 가까운 정점들을 탐색 경로의 시작점과 종점으로 한다. 로드맵에서 깊이우선탐색 방식으로 경로를 탐색하면서 구간마다 감시 영역 및 장애물 영역으로부터 계산되는 충돌영역과 플레이어와의 충돌이 발

생하는지 검사하는데, 다음의 조건을 따른다.

① 각 정점의 Degree만큼 탐색을 반복한다.
② 방문할 정점의 선택은 무작위로 선택한다.
③ 방문했던 정점은 다시 방문하지 않는다.
④ 플레이어가 간선을 지나가는데 걸리는 시간은 [5, (간선의 길이×15)] 사이의 무작위 정수값으로 놓는다.

4. 실험 결과

이 장에서는 Tremblay 등[1,2,3,5,6]이 개발한 플레이어 시뮬레이션을 새로운 충돌 체크 함수와 델로네 로드맵을 적용한 시뮬레이션 방식과 비교한다. 그리고 마지막 절(4.4)에서는 감시 경로의 후보 집합 중 디자이너가 원하는 형태의 감시 경로를 난이도별로 분류한 결과를 기술한다.



[Fig. 5] Map 2

모든 실험은 지도 1, 2에서 진행하였으며 S는 플레이어의 출발점, E는 도착점을 나타낸다. 각 지도에는 미디얼 액시스가 함께 나타나 있다. 실험에 사용된 PC의 사양은 Intel i5-2500 3.30GHz, 8GB RAM이다.

4.1 실험 방법

RRT 경로 탐색에 기반을 둔 기존 시뮬레이션(이하 RRT로 기술)과 새로운 충돌 체크 함수 및 델로네 로드맵에 기반을 둔 시뮬레이션(이하 DEL로 기술)을 비교하는 실험 방법은 다음과 같다.

① 비교실험은 총 3회 실시한다. 각 실험에 사용된 감시 경로는 3.1절의 [Table 1]에 나타난 감시 경로들 중, 경비병 2명과 3명에 대해 각각 무작위로 80개씩 추출한 것이다.

② 주어진 감시 경로에 대해 RRT와 DEL의 "성공률"은 총 50회 경로 탐색을 실시하여 1000초의 시간 이내에 출발점에서 도착점까지 도달하는데 성공한 횟수를 의미한다. RRT는 지도를 60×60의 Grid로 나누어서 샘플링했으며, 1000번의 샘플링 횟수 제한을 두었다. DEL의 경로 탐색방식은 3.3절에 기술된 바를 따른다.

③ 두 알고리즘 모두 플레이어의 속력은 최대 속력 범위 안에서 무작위로 변하며 경비병의 속력은 항상 일정하다.

4.2 RRT와 DEL의 성공률 비교

이 절에서는 DEL이 감시 경로의 난이도를 평가한 결과가 RRT의 그것과 유사함을 입증한다. RRT는 자유영역 어디에나 수많은 샘플링을 통해 경로 탐색할 수 있으므로, 적은 개수의 정점을 가진 델로네 로드맵 위에서만 경로 탐색을 하는 DEL에 비해 언제나 높은 성공률을 보일 수밖에 없다. 그러나, 80개의 감시 경로 중 상대적으로 난이도가 높은 감시 경로들에 대해서는 둘 다 낮은 성공률을, 난이도가 낮은 감시 경로들에 대해서는 둘 다 높은 성공률을 보인다는 것을 보여야 한다. 여기서는 통계학에서 흔히 쓰이는 두 가지 상관계수를 조사한다.

스피어만 상관계수(Spearman correlation coefficient)는 순위 상관계수(Rank correlation)의 일종으로, 두 변수 간의 순위관계가 얼마나 일치하는가를 나타내고, 피어슨 상관계수(Pearson correlation coefficient)는 두 변수 간에 선형적 관련성을 나타낸다. 둘 다 -1부터 1까지의 값을 가지며 1에 가까울수록 강한 양의 상관관계를 가지고 있음을 의미한다.

[Table 2] Spearman correlation coefficient between RRT and DEL

Map 1	Two guards	Three guards
Test 1	0.438	0.621
Test 2	0.451	0.683
Test 3	0.434	0.582

Map 2	Two guards	Three guards
Test 1	0.689	0.638
Test 2	0.650	0.642
Test 3	0.654	0.624

[Table 3] Pearson correlation coefficient between RRT and DEL

Map 1	Two guards	Three guards
Test 1	0.585	0.615
Test 2	0.550	0.634
Test 3	0.559	0.681

Map 2	Two guards	Three guards
Test 1	0.665	0.632
Test 2	0.665	0.631
Test 3	0.664	0.651

[Table 2, 3]은 지도 1, 2에 대한 RRT 성공률과 DEL 성공률간의 스피어만 상관계수와 피어슨 상관계수를 나타낸다. 스피어만 상관계수를 통해 두 성공률 간에 순위관계가 비교적 일치함을 알 수 있고, 피어슨 상관계수는 이들 간에 선형관계 또한 강함을 보여준다.

4.3 속도 비교

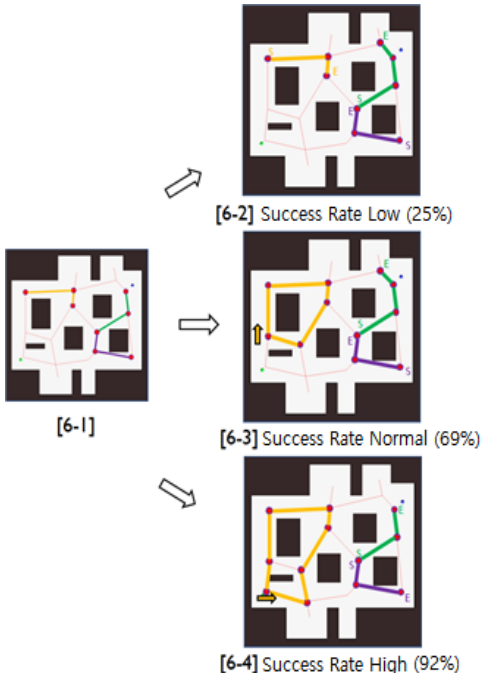
RRT는 수많은 샘플링을 통해 경로 탐색의 성공률을 높이지만, 로드맵을 사용하는 DEL에 비해 많은 시간을 소모한다. 또한, 경로 탐색에서 단위 연산으로 기능하는 충돌 체크 함수가 효율적으로 구현되어 플레이어 시뮬레이션의 속도가 크게 향상되었음을 확인할 수 있다. [Table 4]는 [Table 2, 3]의 실험에서 80개 경로에 대한 시뮬레이션을 수행하는데 걸린 시간(50회 시행한 평균값)이다. 경비병 2명/3명의 감시 경로 80개를 시뮬레이션하여 난이도별로 구분할 때 필요한 시간이 RRT는 12.4초/15.6초, DEL은 3.7초/5.1초이며, 하나의 감시 경로만을 테스트할 때 걸리는 시간은 RRT는 0.2초, DEL은 0.05초이다. 따라서 새로운 충돌 체크 함수와 델로네 로드맵을 이용한 DEL은 개발자가 게임요소들을 변경한 후 그에 따른 플레이어 변화를 실시간으로 분석하고 조정하기에 적합함을 알 수 있다.

[Table 4] Time comparison of RRT and DEL(sec)

Map 1	Two guards		Three guards	
	RRT	DEL	RRT	DEL
Test 1	12.6	3.7	15.6	5.1
Test 2	12.3	3.7	15.6	5.2
Test 3	12.2	3.8	15.4	5.1

Map 2	Two guards		Three guards	
	RRT	DEL	RRT	DEL
Test 1	13.9	3.6	17.6	4.8
Test 2	14.0	3.6	17.8	4.8
Test 3	13.9	3.6	17.8	4.9

4.4 감시 경로의 추출



[Fig. 6] [6-2,3,4] illustrate guard paths with success rate of 25%, 69%, 92%, respectively, containing the path of [6-1]

3.1절에서 지적된 바와 같이, [Table 1]에 있는 감시 경로 후보 집합 Z의 모든 감시 경로에 대해서 시뮬레이션하는 것은 많은 시간이 걸릴 뿐 아니라 불필요한 작업이다. 따라서 후보 집합 Z로부터 디자이너가 원하는 형태, 즉 특정한 정점이나 간선을 포함하는 감시 경로들만을 추출하여, 난이도별로 출력해주는 것이 실질적으로 유용한 접근이다.

[Fig. 6-1]은 3명의 경비병이 반드시 경유해야 할 감시 경로의 예로서 3가지 색깔로 표시하였다. 감시 경로 후보 집합 Z로부터 이들을 포함하는 경로로서 총 40개의 경로가 추출되었으며, 이들에 대해 DEL 알고리즘으로 시뮬레이션한 뒤 성공률을 계산하였다. [Fig. 6-2,3,4]는 난이도별로 감시 경로의 예이다. 각 그림에서 S와 E는 한 명의 경비병이 왕복운동 하는 경로의 시작점과 끝점이며, 화살표는 사이클 경로의 출발점과 방향을 의미한다. 이 세 개의 경로를 비교함으로써 경비병이 얼마나 유기적으로 플레이어의 움직임을 제한하는가에 따라 난이도가 달라짐을 확인할 수 있다. [Fig. 6-2]에서 플레이어는 지도의 우측 상단에 있는 시작점으로부터 왼쪽과 아래쪽으로 움직일 수 있는데 노란색 경로의 경비병과 녹색 경로의 경비병이 같은 시간에 두 곳 모두를 동시에 감시하고 있으므로 난이도가 어렵게 측정된다. 보통과 쉬운 난이도에서는 경비병들이 유기적으로 플레이어의 가능한 경로를 모두 막아서는 경우가 적기 때문에 성공률이 비교적 높게 측정된다.

5. 결 론

본 논문에서는 Tremblay 등이 Unity를 이용해서 개발한 스텔스 게임의 레벨 분석 툴에 대해 두 가지 개선사항을 연구했다. 첫째는 디자이너가 몇 개의 지점을 입력하면 이 지점들을 포함하는 흥미로운 경비병의 감시 경로를 난이도별로 추천해주는 기능을 추가해 레벨 제작을 돕는 툴로서의 유용성과 편의성을 높였고, 둘째는 기존의 충돌 체크 함수 및 RRT-기반 경로 탐색 함수를 새로운 충돌 체크 함수와 델로네 로드맵 기반 경로 탐색 함수로 대체하여 시뮬레이션 속도를 크게 향상시켰다.

향후 연구방향으로서, 디자이너가 입력한 지점들을 포함할 뿐만 아니라 각 지점들에서 주어진 난이도를 만족하는 감시 경로들만을 추천하는 방법을 개발하는 연구와, 시뮬레이션 결과로부터 유의미한 통계 및 자료를 출력해 주는 방법에 대한 연구 등이 있다.

REFERENCES

[1] J. Tremblay, P. A. Torres, N. Rikovitch, and C. Verbrugge, “An exploration tool for predicting stealthy behaviour”, Proc. of the 2013 AIIDE Workshop on Articial Intelligence in the Game Design Process, pp.34-40, 2013.

[2] J. Tremblay, P. A Torres, and C.Verbrugge, “An algorithmic approach to analyzing combat and stealth games”, Proc. of the 2014 IEEE Conference on Computational Intelligence and Games (CIG), pp.1-8, 2014.

[3] J. Tremblay, P. A. Torres, and C. Verbrugge, “Measuring Risk in Stealth Games”, Proc. of the 9th International Conference on Foundations of Digital Games (FDG), 2014.

[4] Q. Xu, J. Tremblay, and C. Verbrugge, “Procedural Guard Placement for Stealth Games”, Proc. of the 5th workshop on Procedural Content Generation (PCG), 2014.

[5] Q. Xu, J. Tremblay, and C. Verbrugge, “Generative Methods for Guard and Camera Placement in Stealth Games”, Proc. of the Tenth Annual AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE), 2014.

[6] S. M. LaValle, “Rapidly-exploring random trees: A new tool for path planning.”, Technical Report No. 98-11, October 1998.

[7] H. Choset et al, “Principles of Robot Motion: Theory, Algorithms, and Implementations”, MIT Press, Boston, MA, 2005.

[8] Y. Shi and R. Crawfis, “Optimal Cover Placement against Static Enemy Positions”, Proc. of the 8th International Conference on Foundations of Digital Games (FDG), pp. 109 - 116, 2013.
[9] J. O’Rourke, “Computational geometry in C”, Cambridge university press, 1998.

나 현 숙(NA, HYEON-SUK)

약력 : 1993 서울대학교 수학과 이학사
1995 포항공과대학교 수학과 이학석사
2002 포항공과대학교 수학과 이학박사
2001-2002 프랑스 INRIA Post Doc.
2002-2003 홍콩 HKUST 전산학과 Post Doc.
2003-현재 숭실대학교 IT대학 컴퓨터학부 교수
관심분야: 알고리즘, 계산기하학, 컴퓨터그래픽스

정 상 혁(Jeong, Sanghyeok)

약력 : 2007-2012 숭실대학교 IT대학 컴퓨터학부 공학사
2013-2015 숭실대학교 IT대학 컴퓨터학부 석사 수료
관심분야 : 알고리즘, 로보틱스, 게임프로그래밍

정 주 홍(Jung, Juhong)

약력 : 2013-2015 숭실대학교 IT대학 컴퓨터학부 재학
관심분야 : 알고리즘, 게임프로그래밍