공간자료구조 개인프로젝트

- QuadTree 기법을 이용한 관광지 내 경로 분석 -

과목명	공간자료구조
담당교수	박수홍 교수님
학과	공간정보공학과
학번	12150951
이름	도경윤
제출일자	2020.06.29

목 차

1. 개요 ····· 3

1.1 주제 ····· 3

1.2 주제선정 배경 및 목적 ····· 3

1.3 Flow Chart ····· 3

2. 활용 자료구조 및 이론 소개 ····· 4

2.1 QuadTree ····· 4

2.2 A* Algorithm ····· 5

2.3 Finite State Machine (Neighbor Finding Algorithm In QuadTree) ····· 6

3. 설계 ····· 8

3.1 자료 획득 ····· 8

3.2 QuadTree 기법을 활용한 대상 지역 분할 ····· 9

3.3 A* 알고리즘을 활용한 노드 간 최단 거리 계산 ····· 12

4. 결과 ····· 16

4.1 결과 요약 ····· 16

4.2 활용방안 ····· 18

4.3 한계점 ····· 18

4.4 향후 연구방향 ····· 21

5. 고찰 ····· 21

6. 참고문헌 및 참고사이트 ····· 22

1. 개요

1.1 주제

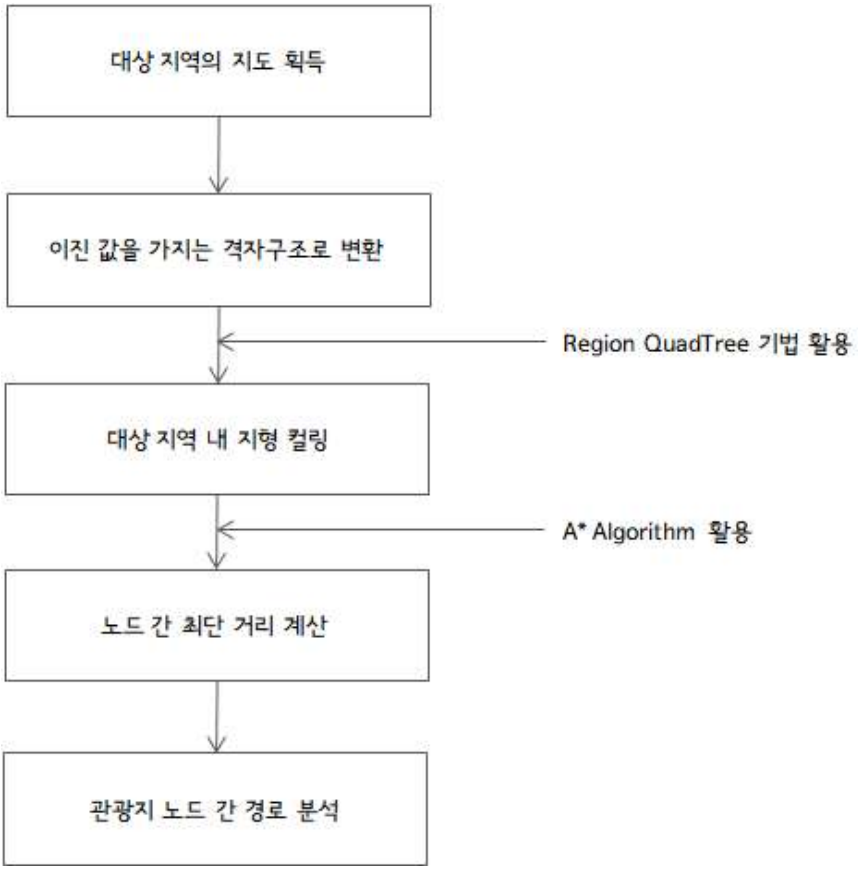
QuadTree기법을 이용한 관광지 내 경로 분석

1.2 주제선정 배경 및 목적

여러 관광명소들이 밀집하여 있는 관광 지구에서 관광객들은 자유롭게 걸어 다니며 관광을 하게 된다. 대부분의 관광객들은 해당 관광지 내의 길을 잘 알지 못하기 때문에 관광명소 간의 이동을 할 때, 어느 길로 가는 것이 최적 경로인지 알 수 없다. 관광지구 내에서 도보로만 이동을 하게 되는 관광객 특성 상 쓸모없는 이동거리의 증가는 관광객에게 있어 치명적인 체력 소모로 이어진다. 따라서 관광객에게 보다 효율적인 동선을 제공하는 것은 관광객의 편의를 향상시키는 것과 직접적인 관련이 있으므로 중요하며 이를 목적으로 한다.

이에 따라 관광 지구의 지형을 관광객들이 걸어 다닐 수 있는 보행지역과 건물, 호수, 벽 등 보행 불가능 지역으로 나누는 과정을 QuadTree를 활용하는 것이 처리 속도와 메모리 측면에서 효과적이라고 생각하여 QuadTree 기법을 활용한 관광지 내 경로 분석을 주제로 선정하게 되었다.

1.3 Flow Chart

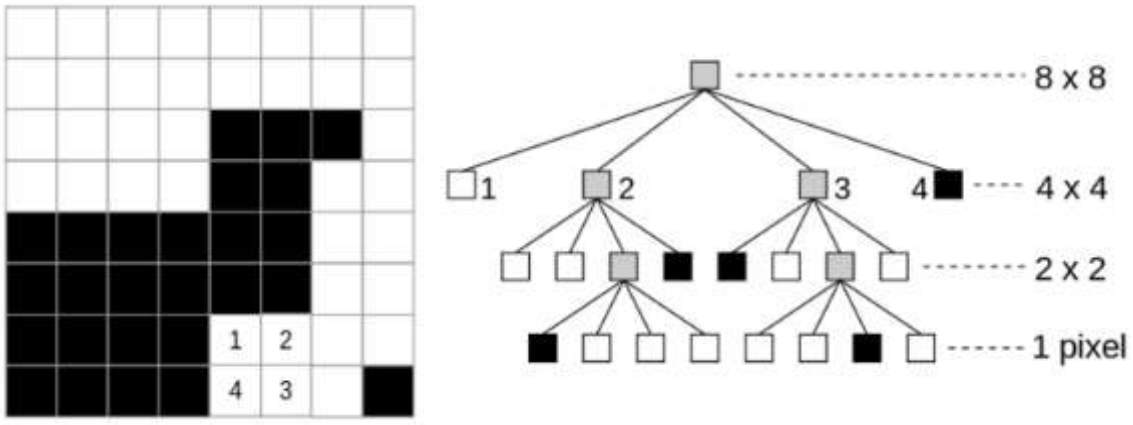


2. 활용 자료구조 및 이론 소개

2.1 Quad Tree

• 정의

- QuadTree란, 트리 자료구조를 기반으로 각 노드가 4개의 자식노드를 가지는 형태의 자료구조이다.
- QuadTree의 종류로는 Point QuadTree, Edge QuadTree, Region QuadTree가 있으며, 해당 설계 프로젝트에서는 Region QuadTree기법을 활용한다.



▲ Region QuadTree

• 특징

- 거대한 지형에 대하여 빠른 처리속도를 이용한 컬링이 가능하며, 메모리 측면에서도 효율적이다.

• 활용

- 대상 관광지역내 지형 중 보행자가 통행가능한 지역과 통행할 수 없는 지역으로 나눌 수 있다. 이 때, QuadTree 기법을 활용하여 통행가능 지역과 통행 불가능 지역을 덩어리 단위로 묶어내어 효과적인 지형의 탐색을 가능하게 한다.

2.2 A* Algorithm

• 정의

- A Star Algorithm은 그래프 탐색 알고리즘의 하나로, 주어진 출발 노드로부터 목표 노드까지의 최단 거리를 계산하는 알고리즘이다.

• 특징

- 출발 노드에서 목표 노드까지 도달할 때 거치는 모든 노드를 찾는 과정에 있어 각 노드를 통과하는 최상의 경로를 추정하는 순위 값인 '휴리스틱 추정 값'을 이용한다.
- 휴리스틱 추정 값을 어떤 방식으로 제공하느냐에 따라 최단 경로 분석의 속도와 정확도가 달라질 수 있다.
- 휴리스틱 추정 값(H Score)과 현재 노드와 이웃 노드간의 거리(G Score)를 더한 값인 F Score가 가장 작게 계산된 노드로 진행하는 과정을 반복하며 최종 목표 노드로 도달하는 알고리즘이다.
- 현재 노드에서 다음 노드로 진행하게 되면 이전 노드는 닫힌 목록(Close List)에 추가하여 다시 접근하지 못하도록 하며, 나머지 이웃 노드는 열린 목록(Open List)에 담겨 F Score를 계산하는 대상이 된다.

• 활용

- 통행가능 지역과 통행 불가능 지역으로 나뉜 대상 관광지역에서 각 관광명소를 노드로 설정한다.
- 각 노드간의 거리, 즉 관광명소간의 최단 거리를 A* Algorithm을 이용한다.
- 해당 설계 프로젝트에서는 휴리스틱 추정 값(H Score)과 이웃 노드와의 거리(G Score)를 계산할 때 유클리드 거리로 측정하였다.

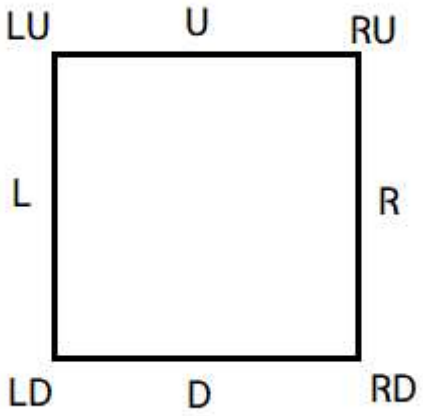
2.3 Finite State Machine (Neighbor Finding Algorithm In QuadTree)

• 정의

- 특정 2차원 지역을 QuadTree 형태로 분할하면 각 노드의 크기가 불규칙하게 분포되는 형태의 지형이 형성된다. 이에 따라 기존 규칙적인 형태의 격자 형태와 달리 각 노드간의 인접성을 찾아내는 방법이 변화하게 된다. 이에 따라 QuadTree로 분할된 지역 내, 특정 노드의 인접성을 파악하는 방법이다.

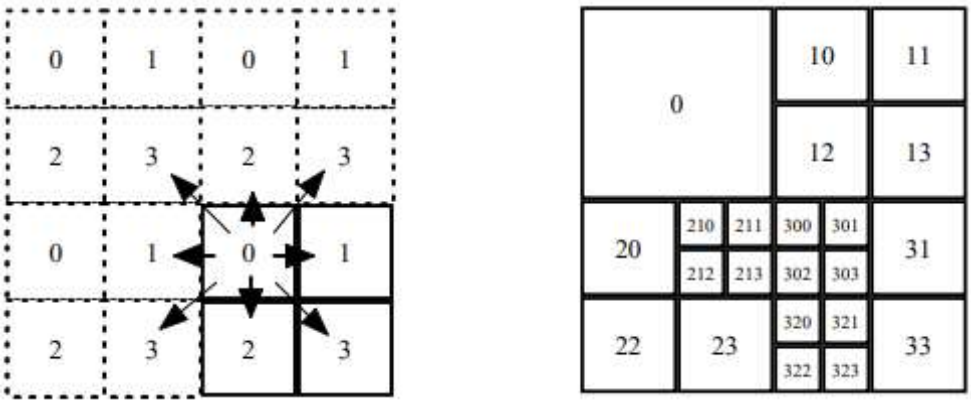
• 특징

- 특정 노드의 인접노드는 상하좌우, 대각선의 8방향에 존재하는 것으로 가정한다.



▲ Node Adjacency Direction

- QuadTree로 분할된 노드가 각각 가지고 있는 Location Code를 이용하여 인접성을 파악한다.



▲ 동일한 Level의 Node(좌)와 다른 Level의 Node에서 Location Code

- 만약, 인접 Quadrant가 동일한 Level이라면, 해당 Location Code 로 변경하며 halt 코드가 도출되게 된다. halt가 도출되면 계산을 중지한다.
- 다른 Level의 Quadrant라면, 특정 Location Code와 특정 방향이 계산되는데, 해당 Location Code로 변경 후에 다음의 계산을 계속해서 진행한다.

Direction	Quadrant 0	Quadrant 1	Quadrant 2	Quadrant 3
R	1,halt	0,R	3,halt	2,R
L	1,L	0,halt	3,L	2,halt
D	2,halt	3,halt	0,D	1,D
U	2,U	3,U	0,halt	1,halt
RU	3,U	2,RU	1,halt	0,R
RD	3,halt	2,R	1,D	0,RD
LD	3,L	2,halt	1,LD	0,D
LU	3,LU	2,U	1,L	0,halt

▲ 현재 Quadrant의 Location Code에 따른 인접 Quadrant Location Code 규칙

- 예) Current Location Code = 320 / LU 방향의 인접 Quadrant 계산
  1. Location Code의 마지막 숫자부터 계산이 시작된다.
  2. Quadrant 0의 LU 방향 결과는 3, LU
  3. Location Code = 320 → 323, LU 방향으로 변경되며 다음 계산은 두 번째 숫자인 2
  4. Quadrant 2의 LU 방향 결과는 1, L
  5. Location Code = 323 → 313, L 방향으로 변경되며 다음 계산은 첫 번째 숫자인 3
  6. Quadrant 3의 L 방향 결과는 2, halt
  7. Location Code = 313 → 213, halt가 계산되었으므로 계산 중지
  8. Location Code 320의 LU 방향 Quadrant의 Location Code는 213 으로 계산 완료.

• 활용

- A* Algorithm 사용 시 인접노드와의 거리를 계산할 때, 노드의 인접성을 파악 시 사용한다.

3. 설계

3.1 자료 획득

① 방법

- 설계 대상지역의 자료는 설계의 결과를 효과적으로 보여주기 위하여 임의로 32X32 크기의 이미지 파일을 작성한다.
- 각 픽셀은 0과 1의 이진 값을 가지는 형태로 한다.

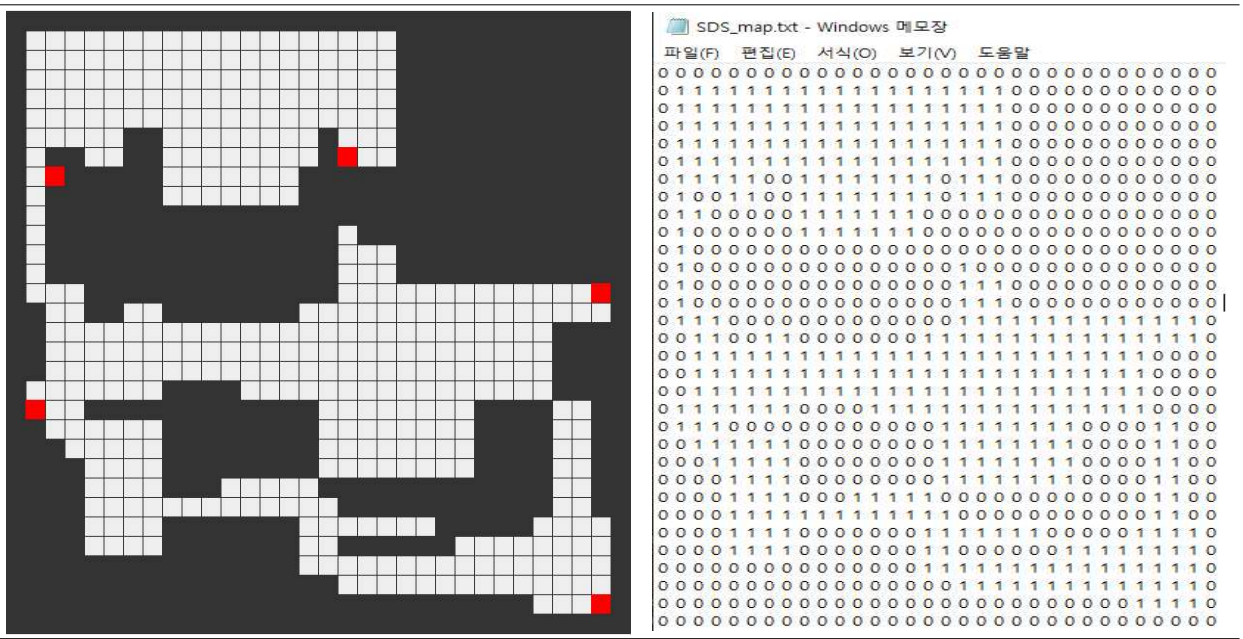
② 과정

```
int[] pixels = new int[width * height];
try {
    FileInputStream fis = new FileInputStream(new File("C:\\Users\\Admin\\Documents\\MATLAB\\SDS_map.txt"));
    BufferedReader bur = new BufferedReader(new InputStreamReader(fis));

    Scanner input = new Scanner(fis);
    for (int i = 0; i < width * height; i++) {
        pixels[i] = Integer.parseInt(input.next());
    }
    bur.close();
} catch (IOException e) {
    e.printStackTrace();
}

for (int i = 0; i < pixels.length; i++) {
    picture[i / width][i % width] = pixels[i];
}
int count = 0;
for (int i = 0; i < height; i++) {
    for (int j = 0; j < width; j++) {
        count++;
        if (picture[i][j] == 1) {
            picture[i][j] = 1;  // 통행 가능 지역
        } else {
            picture[i][j] = 0; // 통행 불가 지역
        }
    }
}
```

▲ 이진 값으로 저장된 파일을 읽어 그 값을 배열에 저장

③ 결과

- **검은색 픽셀 : 통행 불가능 지역 / 흰색 픽셀 : 통행가능 지역 / 빨간색 픽셀 : 관광명소**



▲ 임의로 생성한 32X32 크기의 관광지 파일과 지도의 시각화

3.2 QuadTree 기법을 활용한 대상 지역 분할

① 방법

- QuadTree 기법을 활용하여 32X32, 즉 1024개의 픽셀로 이루어진 대상 지역을 분할한다.
- 0의 값을 가지는 픽셀은 통행 불가지역으로, 1의 값을 가지는 픽셀은 통행 가능지역으로 정의하고 각 Quadrant 내 픽셀의 값이 0 또는 1로 모두 같다면 더 이상 분할하지 않는다.
- 만약 Quadrant 내 픽셀의 값이 동일하지 않다면 4분할하여 4개의 자식노드를 생성한다.
- 픽셀의 크기가 1이 되면 더 이상 분할하지 않는다.

② 과정

메서드 명	설 명
Check()	대상 Quadrant의 범위 내 값이 동일한가를 확인하여 분할 여부를 결정하는 메서드 모두 동일한 값을 가지면 그 값에 따라 통행가능Quadrant, 통행불가Quadrant로 구분함
CheckBoundary()	Check() 메서드 내에서 실제 이미지 파일의 속성 값을 확인하여 분할 여부를 반환하는 메서드
Split()	Check()의 결과에 따라 실제로 Quadrant를 4분할하는 메서드 분할 후 생성된 4개의 Quadrant에 대하여 다시 Check()메서드를 실행

```
void check(QuadTree checkBlank) {
    checkValue cv = checkBoundary(checkBlank);
    if (cv.check) {
        checkBlank.split();
    } else {
        if (cv.value == 0) {
            obstacleQuadTreeList.add(checkBlank);
        } else {
            passableQuadTreeList.add(checkBlank);
        }
    }
}
```

▲ Check() 메서드의 코드

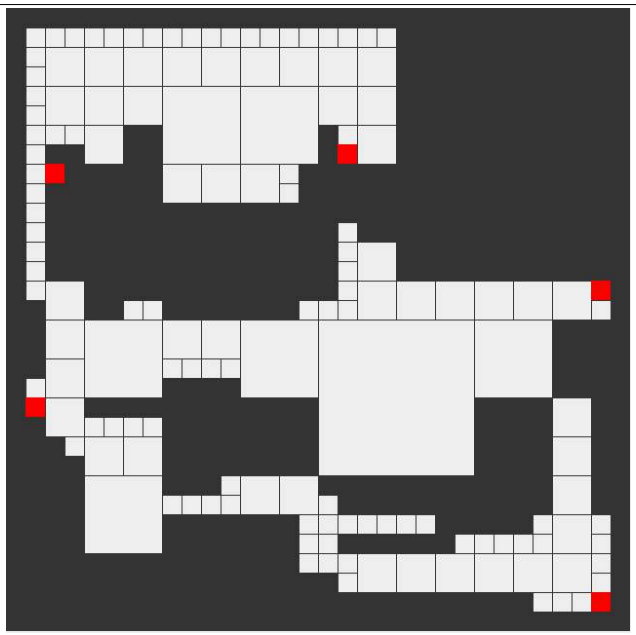
```
checkValue checkBoundary(QuadTree checkBlank) {
    double xStart = checkBlank.boundary.xMin;
    double xEnd = checkBlank.boundary.xMax;
    double yStart = checkBlank.boundary.yMin;
    double yEnd = checkBlank.boundary.yMax;

    int first = picture[(int) yStart][(int) xStart];
    for (double i = yStart; i < yEnd; i++) {
        for (double j = xStart; j < xEnd; j++) {
            if (picture[(int) i][(int) j] != first) {
                return new checkValue(true, 0);
            }
        }
    }
    return new checkValue(false, first);
}
```

▲ CheckBoundary() 메서드의 코드

```
void split() {
    if (this.boundary.getxMax() - this.boundary.getxMin() < 2) {
        if (this.boundary.getyMax() - this.boundary.getyMin() < 2) {
            return;
        }
    }
    c++; // 분할 횟수 측정

    double xOffset = this.boundary.getxMin() + (this.boundary.getxMax() - this.boundary.getxMin()) / 2;
    double yOffset = this.boundary.getyMin() + (this.boundary.getyMax() - this.boundary.getyMin()) / 2;

    // 위 과정을 통해 생성된 4개의 사각형을 정의한다.
    NW = new QuadTree(this.level + 1, new Boundary(this.boundary.getxMin(), this.boundary.getyMin(), xOffset, yOffset), this.name + (this.level + 1) + "'s NW");
    NE = new QuadTree(this.level + 1, new Boundary(xOffset, this.boundary.getyMin(), this.boundary.getxMax(), yOffset), this.name + (this.level + 1) + "'s NE");
    SW = new QuadTree(this.level + 1, new Boundary(this.boundary.getxMin(), yOffset, xOffset, this.boundary.getyMax()), this.name + (this.level + 1) + "'s SW");
    SE = new QuadTree(this.level + 1, new Boundary(xOffset, yOffset, this.boundary.getxMax(), this.boundary.getyMax()), this.name + (this.level + 1) + "'s SE");

    NW.locationCode = this.locationCode + "0";
    NE.locationCode = this.locationCode + "1";
    SW.locationCode = this.locationCode + "2";
    SE.locationCode = this.locationCode + "3";

    check(NW);
    check(NE);
    check(SW);
    check(SE);
}
```

▲ Split() 메서드의 코드

③ 결과

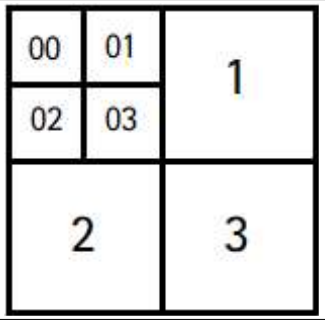


▲ QuadTree를 이용하여 분할한 대상 지역 지도

- 대상 지역을 QuadTree를 활용하여 분할한 결과는 다음과 같다.

분할 횟수 : 105회
통행 가능한 Quadrant : 147개
통행 불가능한 Quadrant: 169개
총 Quadrant : 315개

- 분할 이전에는 의미 없는 픽셀들이 최소 크기 단위로 모두 존재하여 총 1,024 개의 픽셀이 포함되었지만, QuadTree를 활용하여 분할한 결과 315개의 픽셀만 포함하게 된다.
- 분할 시 각 Quadrant에 LocationCode를 부여한다.




```
통행가능 Quadrant [ LocationCode: 33312 ]
통행가능 Quadrant [ LocationCode: 33320 ]
통행가능 Quadrant [ LocationCode: 33321 ]
통행가능 Quadrant [ LocationCode: 33330 ]
통행불가 Quadrant [ LocationCode: 00000 ]
통행불가 Quadrant [ LocationCode: 00001 ]
통행불가 Quadrant [ LocationCode: 00002 ]
```

LocationCode는 좌상, 우상, 좌하, 우하 순서대로 0, 1, 2, 3 의 값을 가지며, 분할 횟수와 그 자릿수가 동일하며 이를 통해 해당 Quadrant의 위치를 찾을 수 있다.

3.3 A* Algorithm을 이용한 노드 간 최단 거리 계산

① 방법

- 출발지 Quadrant와 도착지 Quadrant를 정의한다.
- 출발지 Quadrant에서부터 도착지 Quadrant까지의 경로를 탐색할 때, 각 Quadrant의 인접한 Quadrant를 찾는다. 이 때, 앞서 소개한 Finite State Machine(FSM) 방법을 이용한다.
- 특정 Quadrant에서 이웃 Quadrant의 중심점 간 거리를 G Score, 각 이웃 Quadrant에서 도착지 Quadrant까지의 거리를 H Score, G + H를 F Score로 하여 계산한다.
- 모든 이웃노드 중 F Score가 가장 낮은 방향으로 경로를 진행한다.
- 이 때, 한번 방문한 Quadrant 또는 통행 불가능한 Quadrant는 닫힌 목록(Close List)에 추가하여 해당 방향으로 진행할 수 없게 한다.
- 반대로 이동 가능한 Quadrant는 열린 목록(Open List)에 존재하게 된다.

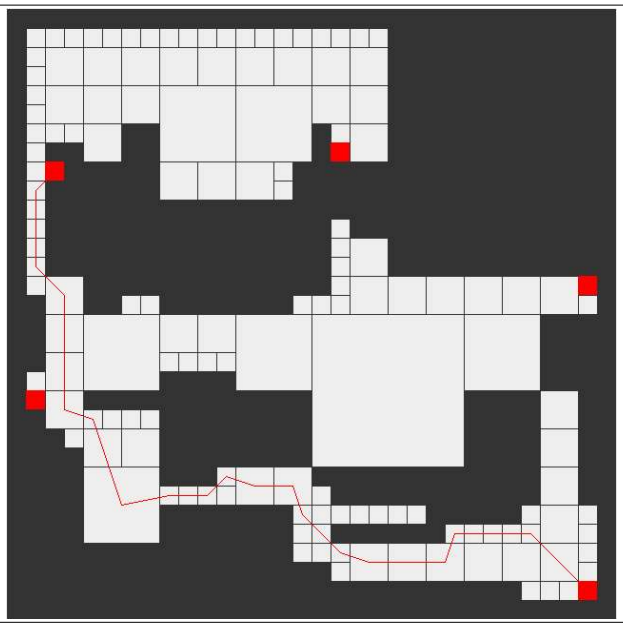
② 과정

메서드 명	설 명
AStarAlgorithm()	출발지 Quadrant를 열린 목록에 추가하는 것으로 시작하여 H, G, F Score의 계산 등 전체적인 A* Algorithm을 수행
findCheapestPath()	열린 목록에 있는 Quadrant 중 F Score가 가장 낮은 방향의 Quadrant를 반환
IsExist()	FSM 방법을 적용하여 인접한 Quadrant의 LocationCode를 반환
getNeighbors()	IsExist()의 결과, 즉 인접한 Quadrant의 LocationCode 중 통행 가능한 Quadrant만을 추출하여 반환

※ 해당 코드는 공간자료구조와 큰 관계가 없다고 생각하여 제외하였습니다.

③ 결과

- 출발 Quadrant는 LocationCode 33330, 아래 그림에서 왼쪽 하단의 Quadrant이다.
- 도착 Quadrant는 LocationCode 02010, 아래 그림에서 왼쪽 상단의 Quadrant이다.



▲ A* Algorithm을 이용한 최단 경로 분석

- 코드를 실행하면서 Log를 다음과 같이 기록하여 어떠한 과정을 통해 결과가 도출되었는지 확인할 수 있도록 하였다.

```
   ==============test================
1  StartQuad's LocationCode : 33330
2  EndQuad's LocationCode : 02010
3      현재노드 (30.5, 30.5) locationCode: 33330
4  [3330, 33312, 33313, 33321, 33331, 33323, 33332, 33333]
5  [3330, 33312, -1, 33321, -1, -1, -1, -1]
6  이웃노드 List[3330, 33312, -1, 33321, -1, -1, -1, -1]
7      이웃노드 (29.0, 29.0) locationCode: 3330
8       g: 2.1213203435596424 h: 33.50373113550191 f: 35.625051479061554
9      이웃노드 (30.5, 29.5) locationCode: 33312
10      g: 1.0 h: 35.0 f: 36.0
11     이웃노드 (29.5, 30.5) locationCode: 33321
12      g: 1.0 h: 34.828149534535996 f: 35.828149534535996
```

Line 1,2 : 출발지 및 도착지 LocationCode 출력
Line 3 : 최소 F값을 가지는 Quadrant를 선택한 후 LocationCode를 출력
Line 4 : IsExist()의 결과, 인접한 Quadrant의 LocationCode 출력
Line 5, 6 : getNeighbors()의 결과, 통행가능한 Quadrant만 추출
Line 7~12 : 이웃 Quadrant의 LocationCode와 G, H, F Score를 계산

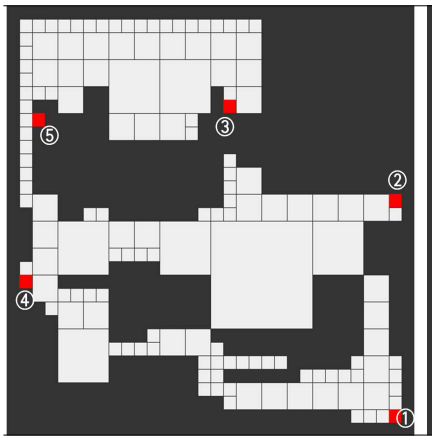
```
Path Finding Complete : Success
최단 거리: 49.21943567764122
33330 3330 33033 33032 33023 33022 32133 3231 3230 3221 32201 32022 23131 2311 2310 23011
  23012 23003 23002 221 2032 20302 2021 2003 2001 0223 02203 02201 02023 02021 02003
 02010
```

경로 찾기가 완료되면 출발지와 목적지 사이의 거리를 반환하고 해당 경로의 LocationCode를 출력한다.

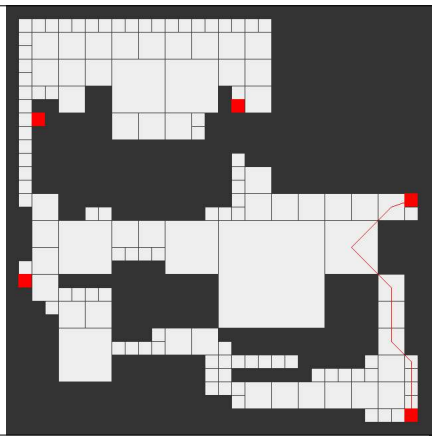
▲ A* Algorithm Log

- 미리 지정해 놓은 5개의 노드 간 거리를 모두 계산한 결과는 다음과 같다.
- 편의상 각 노드에 다음과 같이 번호를 부여한다.

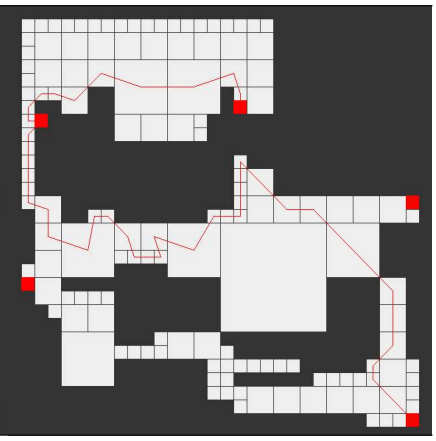
노드 번호	1번 노드	2번 노드	3번 노드	4번 노드	5번 노드
Location Code	33330	13330	10223	20201	02010



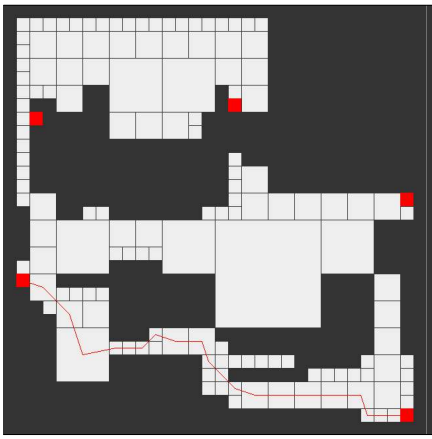
▲ 노드 번호 부여



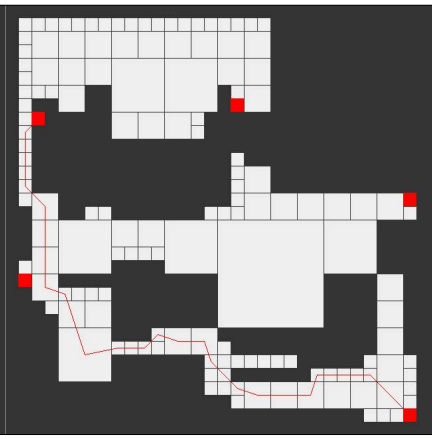
▲ 1번 노드 - 2번 노드



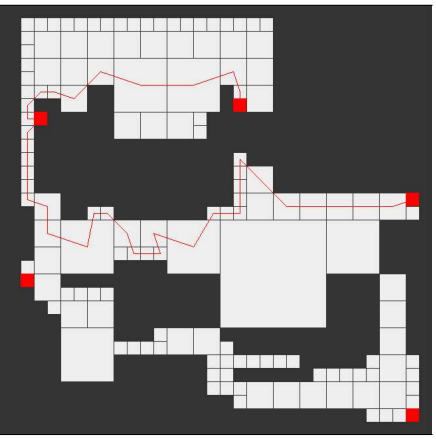
▲ 1번 노드 - 3번 노드



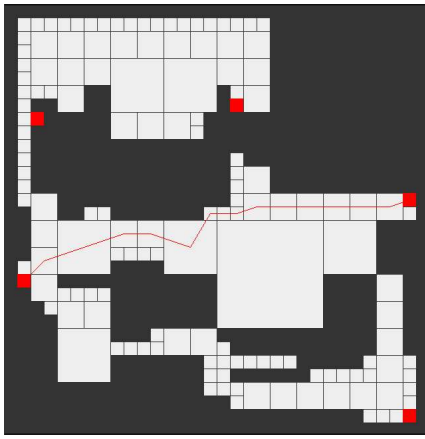
▲ 1번 노드 - 4번 노드



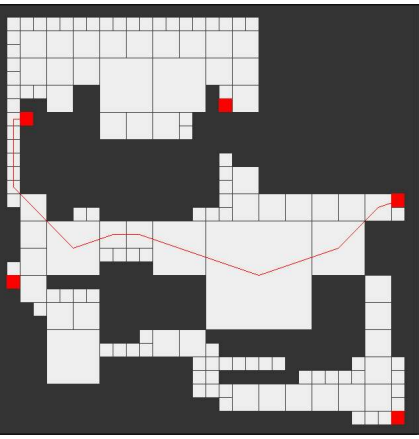
▲ 1번 노드 - 5번 노드



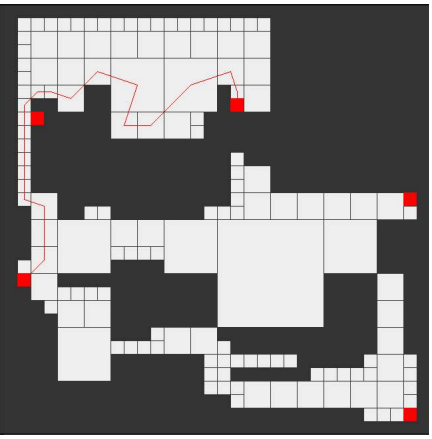
▲ 2번 노드 - 3번 노드



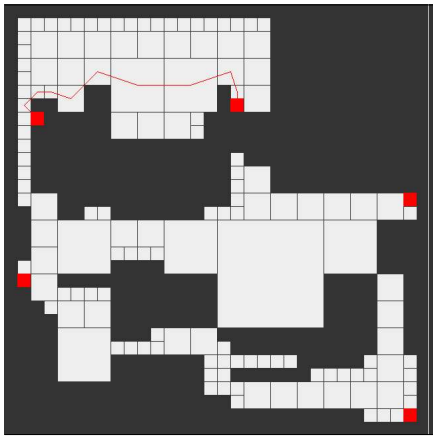
▲ 2번 노드 - 4번 노드



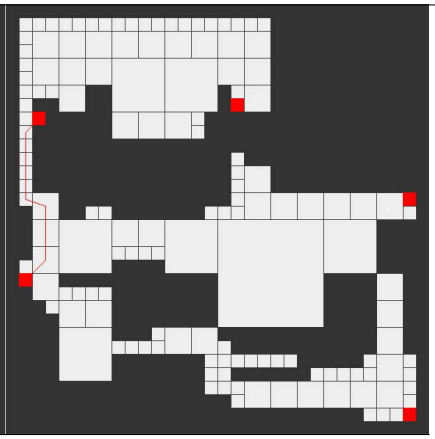
▲ 2번 노드 - 5번 노드



▲ 3번 노드 - 4번 노드



▲ 3번 노드 - 5번 노드



▲ 4번 노드 - 5번 노드

	1번 노드	2번 노드	3번 노드	4번 노드	5번 노드
1번 노드	0	20.19	84.60	35.69	49.22
2번 노드	20.19	0	72.33	31.69	39.16
3번 노드	84.60	72.33	0	39.84	21.14
4번 노드	35.69	31.69	39.84	0	14.12
5번 노드	49.22	39.16	21.14	14.12	0

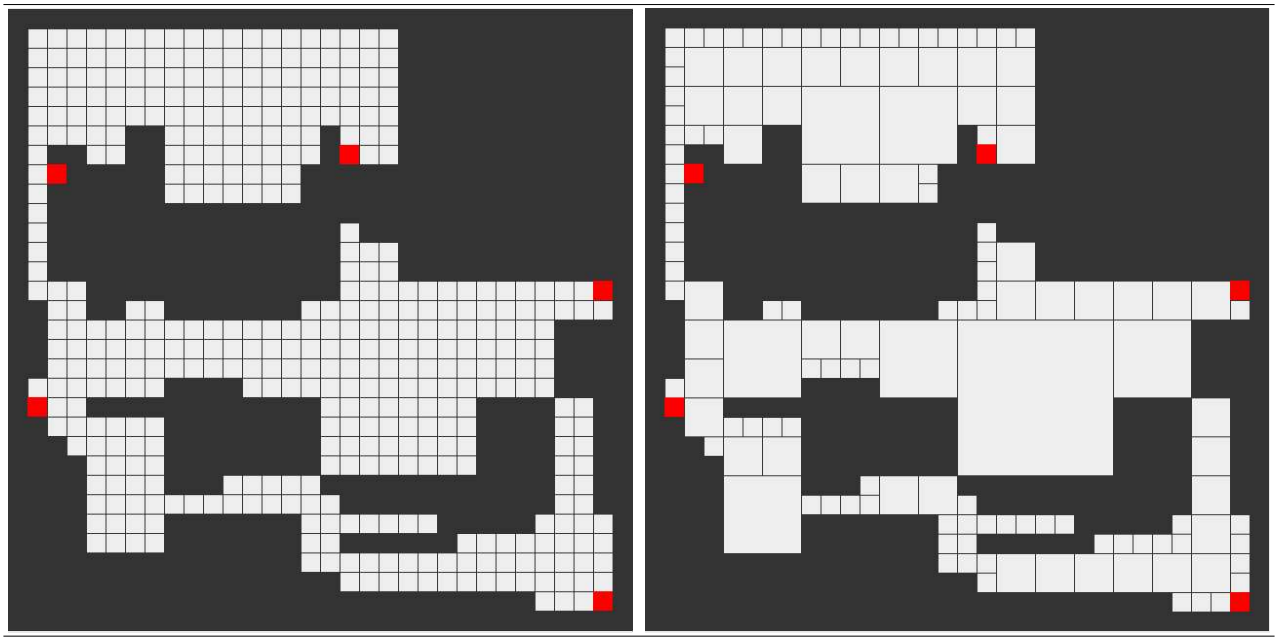
4. 결과

4.1 결과 요약

① 대상 지역을 QuadTree 기법을 이용해 분할한 결과

• 임의로 지정한 32 x 32 크기의 대상 지역을 QuadTree 기법을 활용하여 분할하면 다음과 같은 결과를 얻을 수 있다.

- 총 1,024개의 픽셀로 표현되던 대상 지역을 단, 315개의 사분면만으로 표현할 수 있게 된다.
- 따라서 315 / 1,024 * 100 = 30.76% 정도의 메모리만으로 대상 지역을 나타낼 수 있게 되는 것이다.



총 픽셀 개수 : 1024 개

분할 횟수 : 105회
통행 가능한 Quadrant : 147개
통행 불가능한 Quadrant: 169개
총 Quadrant : 315개

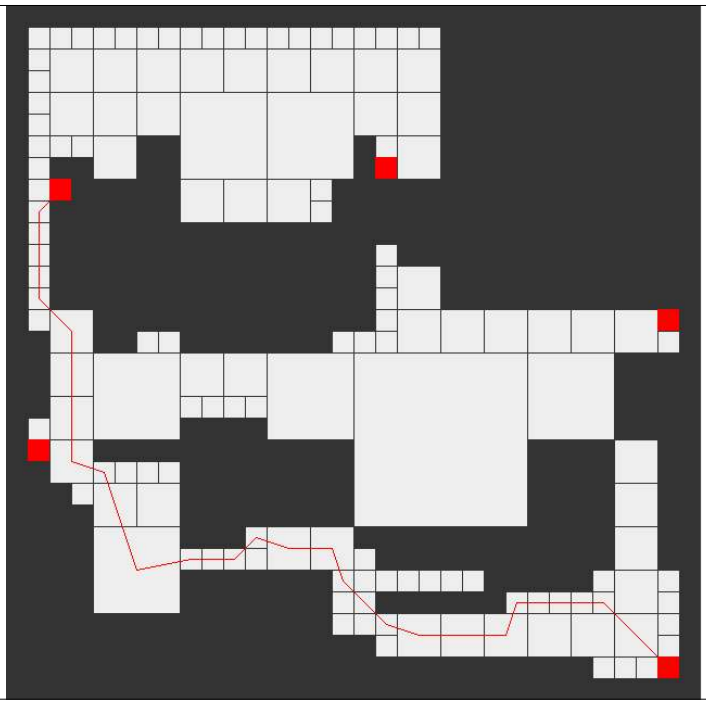
② 노드 간 최단 거리

- 설계 프로젝트에서 정의한 5개의 관광명소 노드 사이의 거리를 다음과 같이 도출하였다.
- 다음 표와 같이 계산된 각 노드간의 거리와 지도의 축척을 고려하면 실제 거리를 계산할 수 있다.

	1번 노드	2번 노드	3번 노드	4번 노드	5번 노드
1번 노드	0	20.19	84.60	35.69	49.22
2번 노드	20.19	0	72.33	31.69	39.16
3번 노드	84.60	72.33	0	39.84	21.14
4번 노드	35.69	31.69	39.84	0	14.12
5번 노드	49.22	39.16	21.14	14.12	0

③ 노드 간 최단 경로 시각화

- 각 노드간의 최단 거리를 계산하는 것뿐만 아니라 실제 사용자들의 편의를 돕기 위하여 해당 경로를 시각화하여 보여주었다.



▲ 1번 노드와 5번 노드 사이의 최단 경로

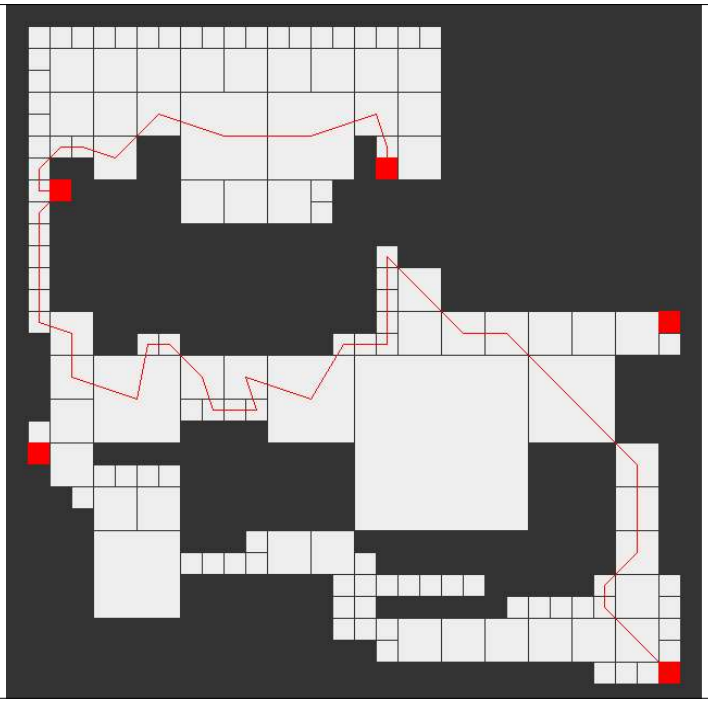
4.2 활용방안

- 설계 프로젝트의 결과로 도출된 각 관광명소간의 최단 거리와 그 경로는 관광객들에게 실시간으로 제공될 수 있어야 한다.
- 이러한 기능을 포함하는 스마트폰 어플리케이션을 개발하여 관광객들이 실시간으로 출발지와 목적지, 또는 경유지 등 사이의 거리를 확인할 수 있게 된다면 해당 지역을 잘 모르고 여행하는 관광객에게 큰 도움이 될 것이라 생각한다.
- 이러한 길 찾기 기능을 실제 관광지역에서 사용하기 위해서는 해당 지역에 대한 토지피복도 등을 획득하여야 한다. 그 후 토지피복에 따라 통행 가능지역과 통행 불가능 지역으로 구분 후 쿼드트리를 이용하여 분할하면 효과적으로 지형 컬링을 할 수 있을 것이다.
- 해당 설계 프로젝트의 결과는 특정 관광지를 대상으로 하지 않고, 자동차의 도로 네트워크상에서도 적용될 수 있다. 그러나 이는 도로의 실시간 복잡도나 특수한 경우를 고려할 수 없기 때문에 상대적으로 비효율적일 것이라 생각된다.

4.3 한계점

① 정확도의 한계

- 설계 프로젝트의 결과는 다음과 같은 특정 상황에서 정확도를 보장할 수 없다는 한계점이 있다.

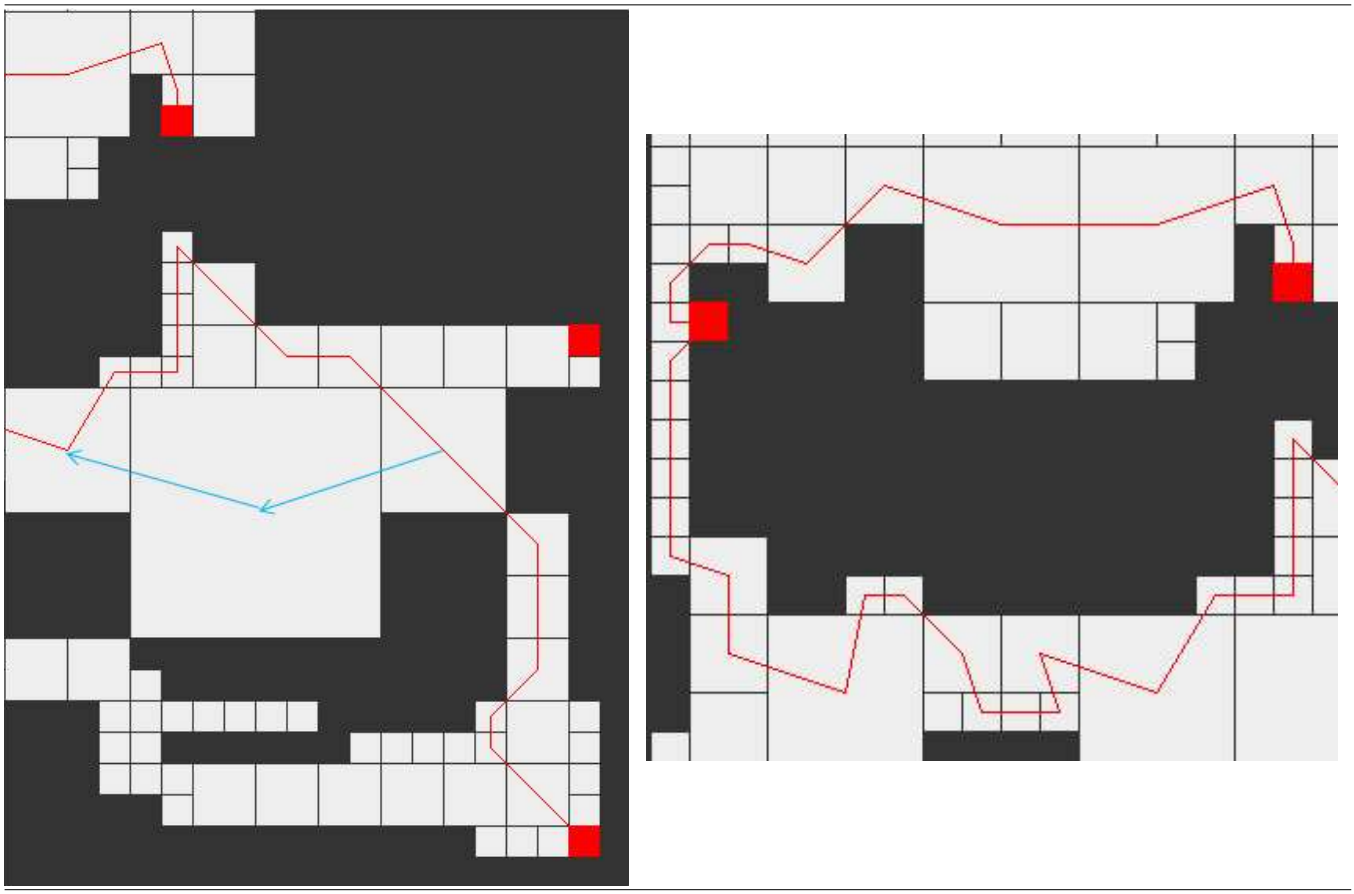


▲ 1번 노드와 3번 노드 사이의 최단 경로

• 1번 노드와 3번 노드 사이의 최단 경로 시각화한 결과이다. 결과를 보면 한 눈에 봐도 최단 경로가 도출되지 않았다는 것을 판단할 수 있다. 이는 길 찾기 알고리즘으로 활용된 A* 알고리즘의 처리 과정에 있어 부족한 점이 있었기 때문이다.

• A* 알고리즘의 과정 중 '휴리스틱 추정 값(H)'는 이웃 노드와 목적지 노드 사이의 직선경로로 결정되도록 처리하였다. 하지만 이 H 값의 결정 과정 중 장애물의 위치에 대한 고려를 하지 않아 아래와 같이 최단 경로가 아닌 목적지와 직선거리가 짧은 방향으로 진행하게 되는 것을 확인할 수 있다.

• 이러한 결과는 '휴리스틱 추정 값'을 결정하는 새로운 방법을 고안하여 적용하면 해결할 수 있을 것이다.

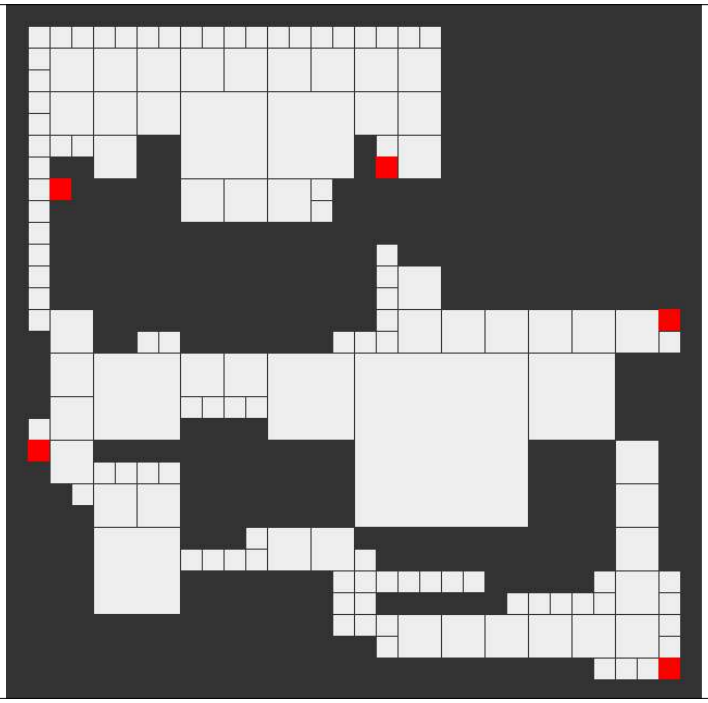


- 빨간색 방향이 아닌, 파란색 방향으로 진행하는 것이 최단 경로이다.
- 이러한 결과는 출발지와 목적지 사이에 장애물이 긴 형태로 위치해 있을 경우 극대화되며 오차는 계속해서 전파된다.
- 이미 방문했던 노드는 더 이상 방문할 수 없기 때문에 극단적인 경우 경로 찾기에 실패할 수도 있다.

▲ 1번 노드와 3번 노드 사이의 최단 경로로 도출된 결과

② 불필요한 Quadrant의 생성

- QuadTree 내에서 어떤 Quadrant와 이웃하는 Quadrant를 찾아내는 방법으로써 Location Code를 바탕으로 하는 Finite State Machine, FSM 기법을 활용하였다.
- 이 기법을 사용할 시 이웃하는 Quadrant의 존재 유무를 Location Code를 통해 확인할 수 있게 되는데, 지도의 가장자리 부분에서는 올바르지 못한 결과를 반환하게 된다.
- 예를 들면 다음과 같다.



- 지도의 가장 왼쪽 상단의 픽셀의 Location Code는 00000 이다.
- 이 픽셀의 LU 방향 이웃노드를 FSM 기법을 사용하여 계산하면, 그 결과는 33333이 될 것이다. (7P 참고)
- 하지만 이 픽셀의 LU 방향에는 어떠한 노드도 존재하지 않는다.

- 이와 같은 이유로 설계 프로젝트를 진행할 때에는 가장자리의 한 픽셀을 모두 통행 불가능한 지역으로 구분하여 진행하였다. 이에 따라 불필요한 분할이 이루어지게 되고 필요하지 않은 Quadrant가 여러 개 생성되는 결과를 보였다.
- 이러한 결과를 막기 위해서 지도의 가장자리에서 예외적인 처리를 하는 코드를 추가적으로 작성해 주어야 할 것이다.

4.4 향후 연구방향

- 한계점에서도 이미 언급했듯이, 몇 가지 한계점이 존재한다. 이번 설계 프로젝트의 목적에 맞게 사용자의 편의를 증대시키기 위해서는 길 찾기 알고리즘의 수정을 통하여 보다 정확한 결과를 도출하도록 할 것이다.
- 이번 설계 프로젝트에서는 임의로 작성한 지도에 대한 분석을 수행하였지만, 실질적인 의미를 가지기 위해서는 실제 관광지나 놀이공원 등의 지역의 자료를 획득한 후 전처리 과정을 거쳐 분석을 수행해보아야 할 것이다.
- QuadTree 기법을 활용하여 특정 지형을 탐색하는 것이 시간적 측면에서 효율적이라고 하였는데, 이를 효과적으로 보여줄 수 있을만한 상황과, 방법에 대하여 생각해 보아야 할 것이다.

5. 고찰

이번 설계 프로젝트에서 QuadTree 자료구조를 이용한 지형의 분할과 이를 이용한 경로 분석을 수행해보았다. QuadTree 자료구조가 거대한 지형을 컬링 할 때, 시간적 측면에서 효율적이라고 하여 이 주제를 선정하게 되었다. 하지만 프로젝트를 진행하면서 이러한 장점을 체감하지 못했다. 그 이유로, 특정 지역에 대한 이미지 파일 속성 값을 읽어 0과 1의 이진 값으로 변환하는 것은 공간을 분할하지 않고 모든 격자를 탐색하는 것이 오히려 더 빠른 처리결과를 보였다. 하지만 QuadTree를 이용하여 공간을 분할한 결과만 놓고 봤을 때는 분할하지 않았을 때보다 더욱 깔끔하고 효과적인 작업이 가능할 것 같다는 생각이 들었다.

이번 설계 프로젝트에서 임의로 작성한 대상 지역의 크기가 32 x 32 밖에 되지 않아 이러한 효과를 체감하지 못하였을 수도 있었을 것 같다. 실제로 어떤 관광지 내 실제 크기가 5km * 5km 정도 된다고 가정하였을 때, 1m의 공간 해상도를 가지게 하려면 5000 x 5000 크기의 영상이 얻어지게 될 것이다. 이 영상을 QuadTree를 활용하지 않고 분석하게 된다면 총 2천 5백만 개의 픽셀을 이용하여 작업을 처리하여야 한다. 하지만 설계 프로젝트의 결과에서도 볼 수 있었듯이 QuadTree를 이용하여 지형을 분할하게 되면 훨씬 적은 수의 Quadrant로 그 지역에 대한 작업을 수행할 수 있을 것이다. 이 점을 고려한다면 이번 설계 프로젝트의 목적을 위해 QuadTree를 이용하는 것이 적합하였다 생각할 수 있다.

설계 프로젝트를 JAVA를 이용한 코딩으로 진행하면서, QuadTree 기법에 대한 전반적인 지식을 확실하게 갖추게 되었고 A* 알고리즘과 FSM 기법 등 여러 논문을 찾아보면서 QuadTree 기법의 활용을 보조할 수 있는 방법 또한 알게 되었다.

프로젝트의 결과에 대하여 크게 만족하지는 못하였지만, 난이도가 쉽지 않았고 접해보지 못한 여러 방법에 대한 공부도 할 수 있었던 것 같아 좋았고 이를 더욱 개선한다면 실제로 실용적인 서비스를 진행해볼 수도 있을 것 같아 좋은 경험이 되었다.

6. 참고문헌 및 참고사이트

• Robert Yoder, Peter Bloniarz,「A Practical Algorithm for Computing Neighbors in Quadtrees, Octrees, and Hyperoctrees 」,
http://web.archive.org/web/20120907211934/http://ww1.ucmss.com/books/LFS/CSREA2006/MSV4517.pdf

• 최단 경로 탐색 - A* Algorithm
http://www.gisdeveloper.co.kr/?p=3897