Graph 상 Cops and Robbers 게임 개발

2019 졸업과제 최종 보고서

정보컴퓨터공학부

팀명 : 오뚜기

201624563 임학천

201624501 송태영

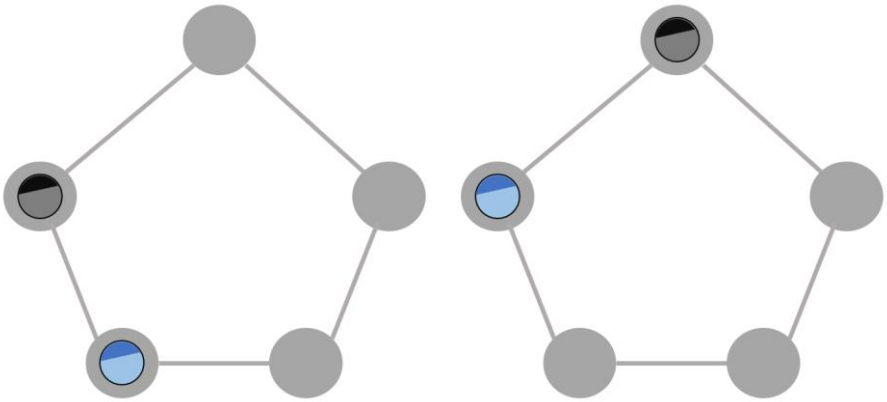
201324456 배선우

지도교수 : 조 환 규

목차

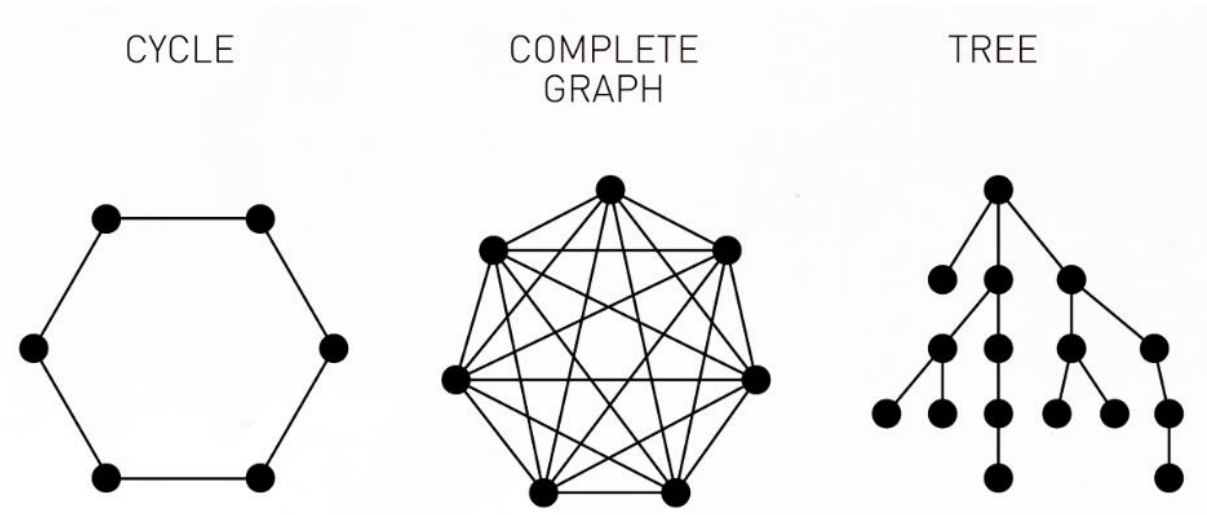
Cops and Robbers Game Theory

해당 과제의 근간이 되는 주제는 Cops and Robbers Game Theory이다. Cops and Robbers Game Theory는 게임 이론중의 하나로, 그래프 상에 Cop과 Robber가 존재하고, Cop이 Robber를 모두 잡을 수 있는지 없는지를 결정하는 문제이다.



<그림> Cops and Robbers Game Theory Graph

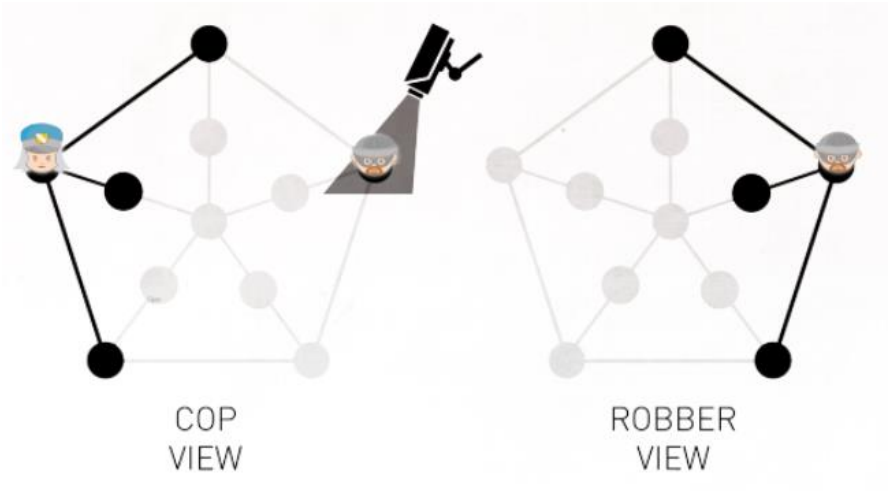
그림과 같이 유한한 그래프가 주어지고, 1명 이상의 Cop과 1명 이상의 Robber가 그래프 위에 배치된다. 각 Cop과 Robber는 각 차례마다 자신과 인접한 노드로 이동할 수 있다. 이 때 Cop이 유한한 차례안에 모든 Robber를 잡을 수 있는지 없는지 판단해야 한다.
만약 Cop이 유한한 차례안에 모든 Robber를 잡아낼 수 있다면 이를 Cop-Win 이라고 하며, 반대로 Cop이 유한한 차례안에 모든 Robber를 잡아낼 수 없다면, 즉 Robber가 무한히 Cop으로부터 도망칠 수 있다면 이를 Robber-Win 이라고한다.



<그림> 여러가지 모양의 그래프

기본적으로, 특정 그래프에서 Cop과 Robber가 각 1명일 때 주어진 해답은 다음과 같다.

- Cycle graph의 경우 Robber가 항상 Cop으로부터 도망칠 수 있으므로 해당 그래프는 Robber-Win 그래프이다.
- Complete graph의 경우 각 노드가 모든 다른 노드와 연결되어 있으므로 항상 한 번의 차례만에 Cop이 Robber를 잡을 수 있으므로 Cop-Win 그래프이다.
- 유한한 Tree의 경우 사이클이 존재하지 않으므로 항상 유한한 시간안에 Robber가 잡히게 되므로 Cop-Win 그래프이다.



<그림> Cop과 Robber가 볼 수 있는 시야에 제한을 두는 변형 방식

또한 Cop 수와 Robber 수를 변수로 두는 것 외에도, 다양한 상황에 대한 변수를 둘 수 있다.

- Cop과 Robber가 볼 수 있는 최대 시야에 제한을 두기
- Cop 및 Robber의 속도를 변경하는 아이템 만들기
- Robber의 차례 다음 모든 Cop이 움직이는 것이 아닌 하나의 Cop만 움직이고 다시 Robber의 차례로 넘어가기 (Lazy Cops & Robbers)

게임 소개

진행 방법

: Cops and Robber 게임 이란 경찰과 도둑이 방향이 없는 유한 그래프(G) 에서 진행되는 게임이다. 다수의 경찰과 한 명의 도둑이 정점에 위치하며 Agent기준으로 턴 방식으로 edge를 따라 한번씩 움직일 수 있다. 경찰이 이동후에 도둑과 같은 정점에 위치하면 게임은 경찰이 승리한다.

그래프(G)에서 강도를 잡는 데 필요한 최소 경찰 수를 해당 그래프의 Cop(G) 라고 한다. Cops and Robber 논문에서는 하나 이상의 그래프를 Subgraph 또는 Induced Subgraph를 금지하여 정의 된 그래프 클래스에서 경찰 수를 연구한다.

그래서 실제 게임에서는 다른 Cop-Win Graph or Robber-Win Graph 의 한계점을 극복하기 위해 Grid Graph를 기본으로 하여 게임을 진행한다.

게임은 두 가지 모드로 진행이 가능하다.

- **Player VS Player**

  Player VS Player, 즉 PVP 모드에서는 온라인을 통해 최대 4명의 사용자들이 함께 Cops and Robbers 게임을 진행할 수 있다.

- **Player VS Computer**

  Player VS Computer, 즉 PVC 모드에서는 플레이어가 Robber가 되어 Cop 알고리즘이 적용된 컴퓨터와 게임을 진행할 수 있다.

Cop 및 Robber의 승리 조건은 다음과 같이 주어진다.

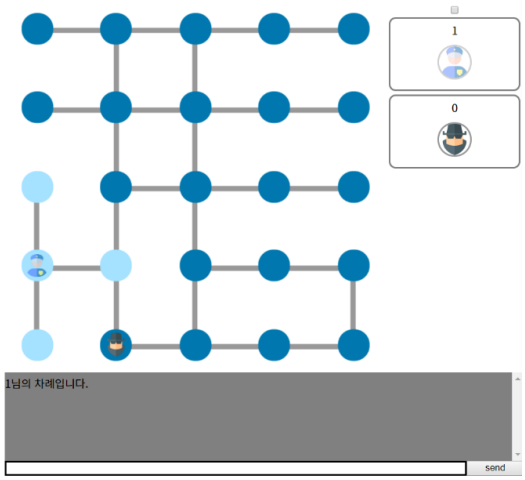
- **Cop의 승리 조건**

  그래프 상에 존재하는 모든 Robber를 제거한다

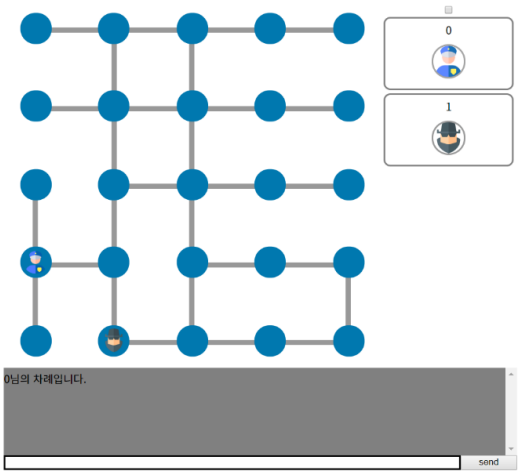
- **Robber의 승리 조건**

  Robber가 $(N + M) \times 2$ 턴 만큼 (이때 N은 그리드의 가로 길이, M은 그리드의 세로 길이) 도망치는데 성공한다.

다음은 실제 구현된 게임 화면을 통한 게임 진행 순서이다.

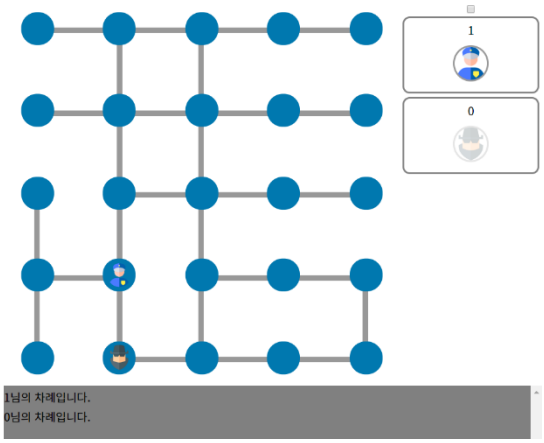


<1턴 Cop 화면>



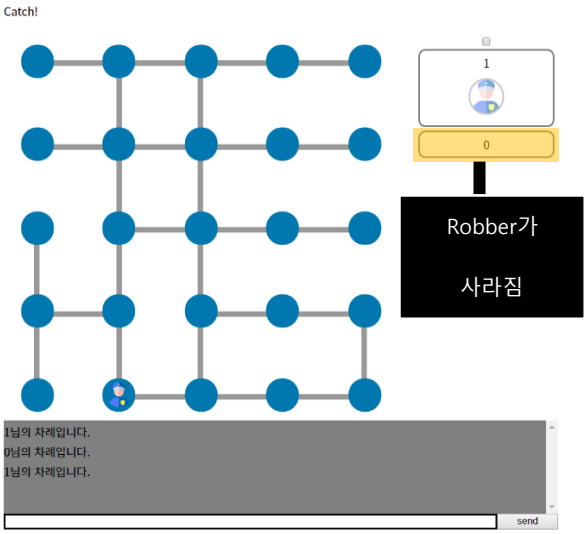
<1턴 Robber 화면>

1. 위의 그림과 같이 1턴에 경찰이 움직일 차례에 Cops화면에서는 움직일 수 있는 Vertex에 Highlight 시켜주고 Robber화면에는 아무런 변화가 일어나지않는다. 채팅창에는 어느 Agent 의 차례인지 표기해준다.



<2턴 Cop화면>

2. Highlight된 Vertex중 하나를 Cop이 클릭하면 해당 Vertex로 이동하게 되고 다음 턴으로 넘어간다.



3. Cop이 Robber가 있는 Vertex로 이동하여 Cop과 Robber가 동일한 위치에 있게되면 Robber는 잡힌 것으로 간주되어 Robber는 제거된다. 만약 모든 Robber가 제거되었다면 해당 게임은 Cop의 승리로 종료된다.

4. 만약 Robber가 $(N+M) \times 2$ 턴 만큼 (이때 N은 그리드의 가로 길이, M은 그리드의 세로 길이) 도망치는데 성공했다면 해당 게임은 Robber의 승리로 종료된다.

기존 게임의 변형

기존 Cops and Robbers Game Theory에 제시된 가장 기본적인 형태 외에도 다음과 같은 변형 방식을 구현하고자 한다.

- 여러 명의 Cops and Robber

Cops and Robber의 가장 기초가 되는 방식은 1명의 Cop과 1명의 Robber에 대해 게임을 진행하는 것이다. 이 경우 주어진 그래프가 cop-win-graph인지 robber-win-graph인지 판단이 가능한 경우가 대부분이지만, Cop과 Robber들이 각각 2명 이상 존재할 경우에 대해서는 아직 알려지지 않은 경우가 대부분이다. 해당 프로젝트에서는 온라인을 통한 사람 대 사람 대결에서 여러 명의 Cop과 Robber을 가지고 게임을 진행해볼 수 있도록 한다

- 시야 제한

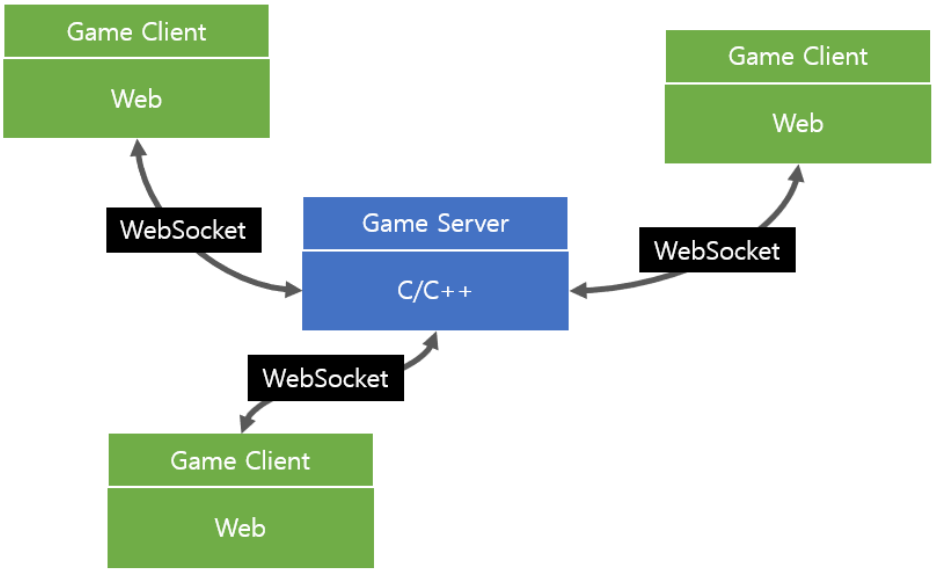
Cops and Robber의 변형 버전에서 가장 자주 언급되는 변형법인 시야 제한을 구현하고자 한다. 각 Cop과 Robber는 자신과 인접한 k개의 노드에 대해서만 시야를 가지고 있으며, 시야가 없는 노드의 경우 해당 노드에 누가 위치하고 있는지에 대한 정보를 알 수 없다. 이는 현실 세계에서 실제 인접한 장소에 대한 정보를 획득할 수 있는 상황과 부합한다.

- **이동 제어**

  또 다른 변형 중의 하나는 Cop과 Robber가 한 턴에 움직일 수 있는 노드의 거리에 변형을 두는 것이다. 그래프 상의 특정 노드에 아이템이 생성되고, 해당 아이템을 획득하면 해당 플레이어는 한 번에 2개의 노드를 건너 뛸 수 있게 된다. 이는 현실 세계에서 교통 수단 등을 통해 빠르게 이동하는 상황과 부합한다.

전체 설계

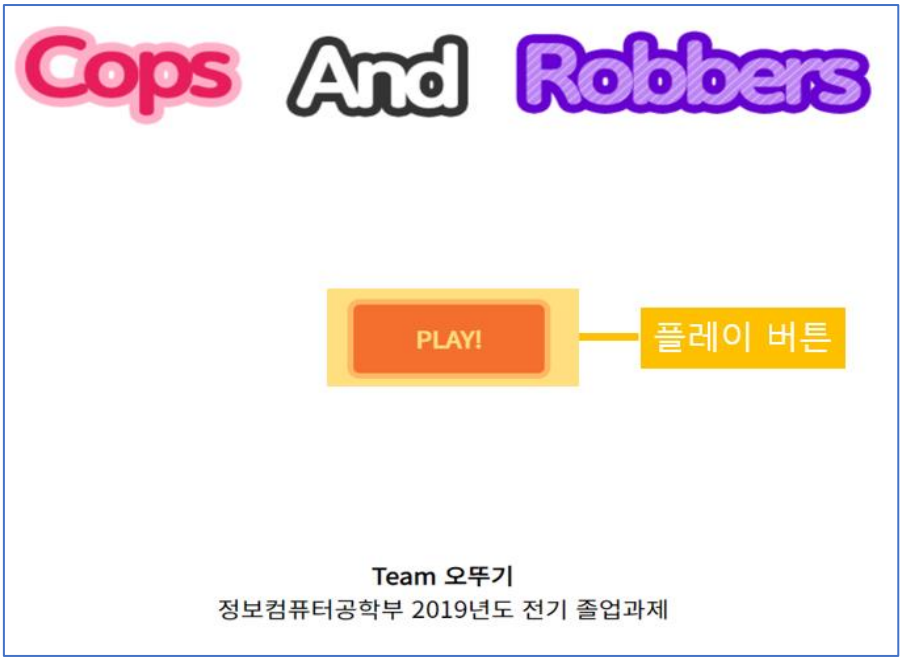
Cops and Robbers 게임은 멀티 플레이가 가능한 구조로 게임 서버와 게임 클라이언트로 이루어져 있다. 게임 서버는 C/C++로 작성되었으며 게임 클라이언트는 웹으로 작성되었다.
서버와 클라이언트는 WebSocket 프로토콜을 통해 통신한다.



<그림> 서버 및 클라이언트 설계도

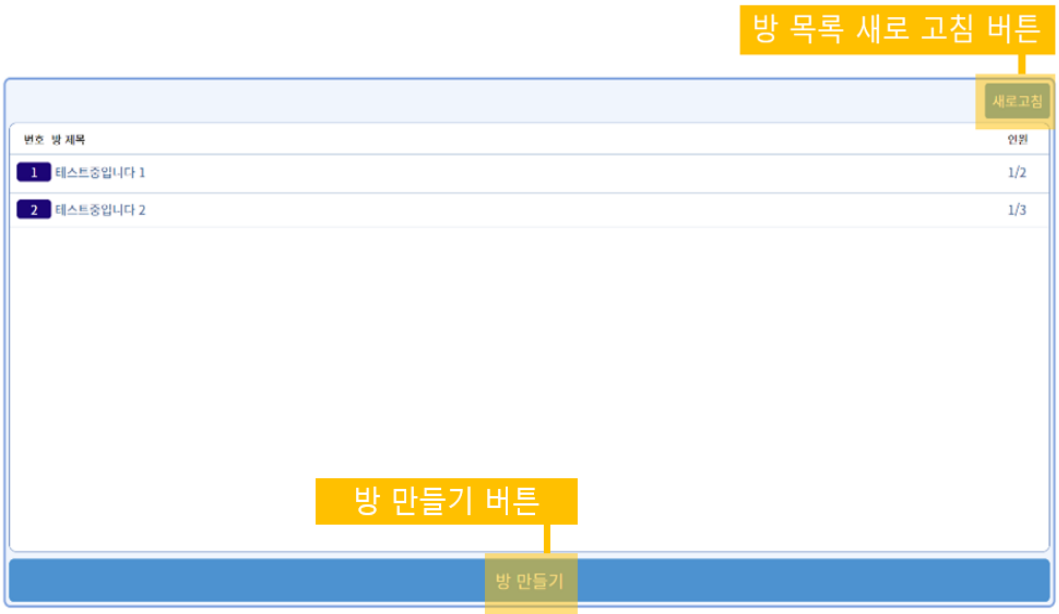
클라이언트 구조

화면 구성



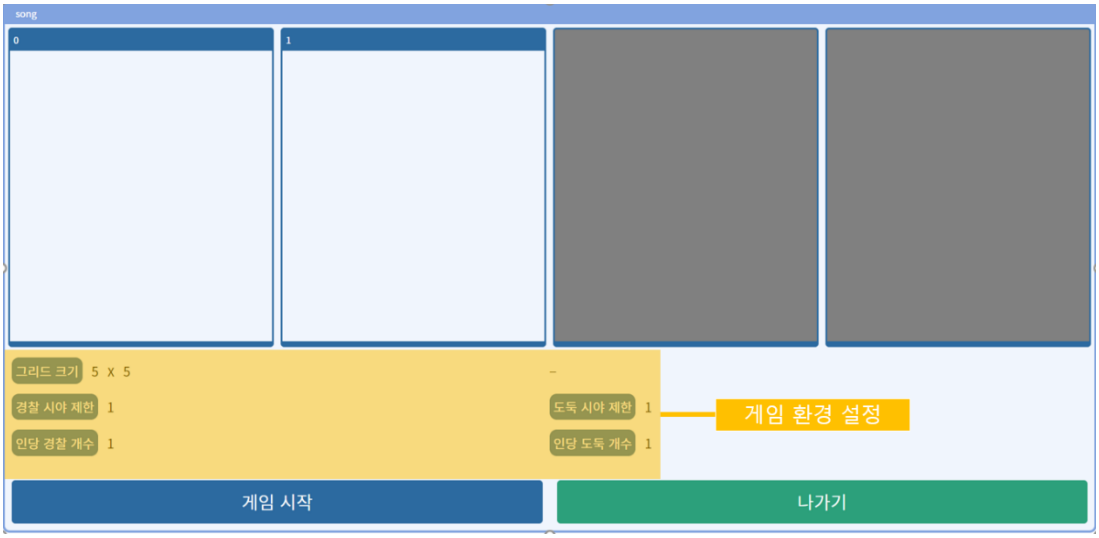
<그림> 게임 시작 화면

- 게임 시작 화면으로 Play 버튼을 누르면 닉네임을 설정하고 로비화면으로 넘어간다.



<그림> 방 목록

- 로비는 <그림 2>와 같이 방 목록 들이 있는데 방을 클릭하면 입장할 수 있다.
- 새로 고침 버튼을 누르면 방 목록 들이 새로 고침 되어 나타난다.
- 방 만들기 버튼을 누르면 방 제목과 제한인원 수를 설정하여 방을 만들 수 있다.

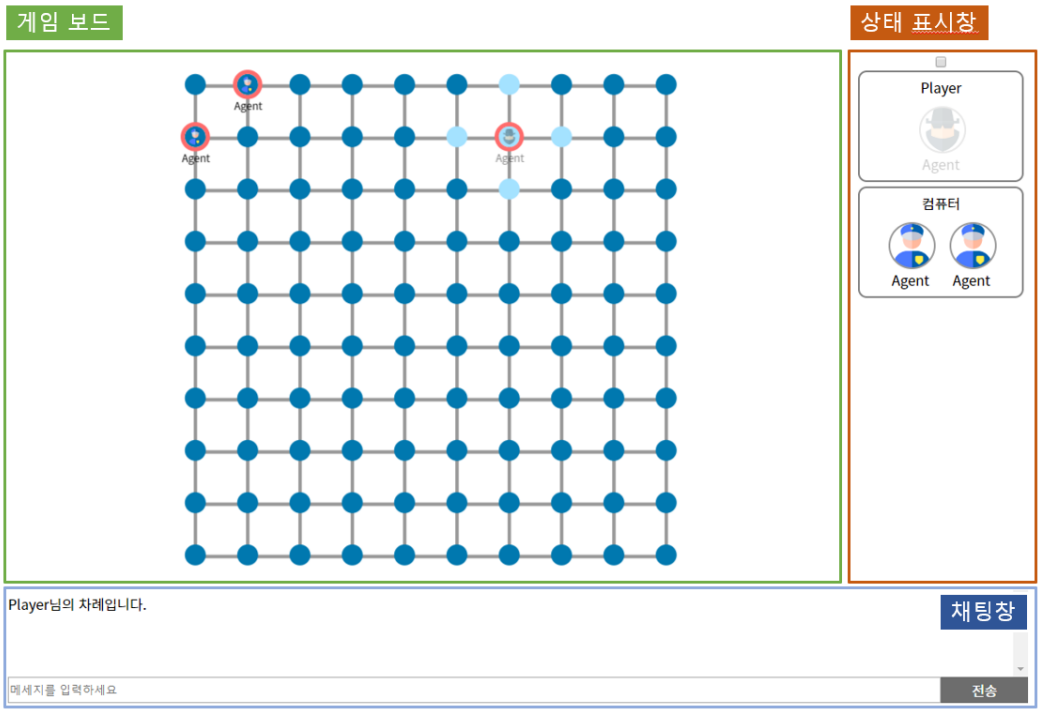


<그림> 게임 대기 방 화면

- 방장이 시작 버튼을 누르면 게임이 시작된다
- 나가기 버튼을 누르면 해당 방에서 나가게 된다.
- 방장은 그리드 크기 및 경찰 시야 제한 범위, 인당 경찰 개수, 도둑 시야 제한 범위, 인당 도둑 개수를 설정할 수 있다.

<그림> 역할 선택 화면

- 게임을 시작하기전 각 Player 마다 직업을 선택하기위해 직업선택 버튼을 눌러 선택한다.

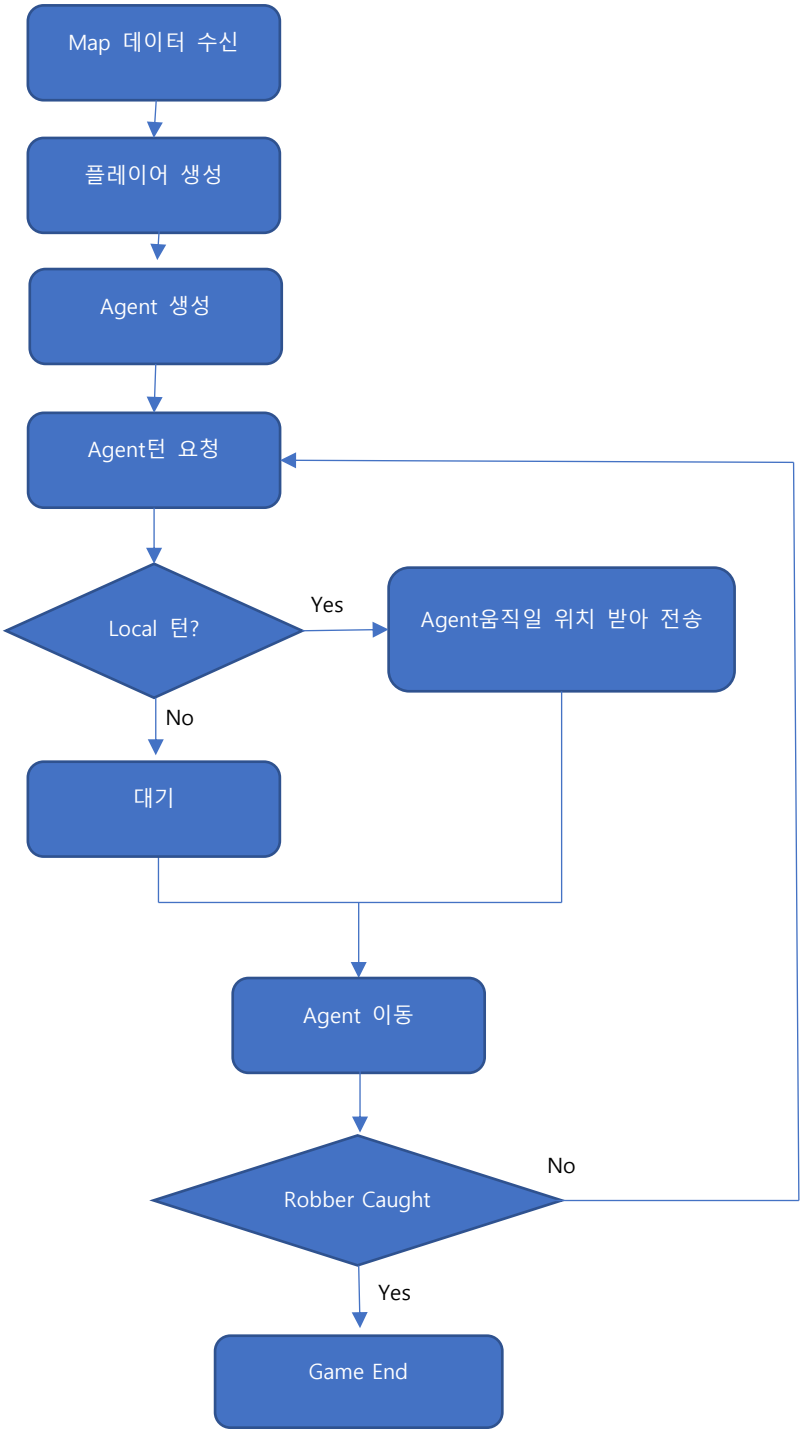


<그림> 인게임 화면

- 게임 보드 : 그래프 및 현재 Cop & Robber의 위치를 표시해준다.

- 상태 표시창 : 현재 누구의 차례인지, 어떤 말이 살아있는지 표시해준다.
- 채팅창 : 게임 진행 메시지 및 사용자들 간 대화를 위한 채팅창이다.

Flow Chart



<표> Client In-game Flow Chart

서버 설계

실질적인 게임을 실행하는 부분이다. WebSocket을 이용해서 클라이언트에서 받은 정보로 대기실을 운영하거나 게임을 진행하고 처리한 결과를 다시 클라이언트로 보낸다.

파일 구성

main.cpp : main문이 있고 사용자가 웹페이지 접속 또는 나갔을 때와 클라이언트에서 메시지를 받는 부분

message.cpp : 웹소켓과 통신하기위해 쓰이는 메시지를 처리하는 부분. 받은 메시지를 분류하고 프로그램(서버)에서 처리한 결과를 메시지로 보낸다.

message.hpp : message.cpp의 헤더파일

room.cpp : 대기실과 관련된 작업을 처리하는 부분. 방 만들기, 방 입장, 방 나가기 등을 처리한다.

room.hpp : room.cpp의 헤더파일

map.cpp : 맵과 관련된 일을 하는 부분. 주어진 조건대로 Vertex들과 Edge들을 생성하여 맵을 구성한다.

map.hpp : map.cpp를 위한 헤더파일

game.cpp : 게임과 관련된 일을 하는 부분. 게임을 진행시키고 게임 정보를 저장한다.

game.hpp : game.cpp를 위한 헤더파일

통신 프로토콜

서버와 클라이언트 간 통신을 위한 메시지 프로토콜은 다음과 같이 정의된다.

방향	이름	설명	대응 함수
클->서	request_username	사용자가 정한 닉네임을 보낸다.	setName()
서->클	player_init	정해진 사용자의 ID와 닉네임을 보낸다.	
클->서	request_room_list	대기실들의 정보를 요청한다.	roomList()
서->클	room_list	대기실들 정보를 보낸다.	

클->서	request_room_create	방 생성을 요청한다.	createRoom(), joinRoom()
서->클	room_join_accept	방에 입장됨을 알리고 방 정보를 보낸다.	
서->클	room_player_join	방에 있는 유저들에게 새로 들어온 유저 정보를 보낸다.	
클->서	request_room_join	방 입장을 요청한다.	checkroom(), joinRoom()
서->클	room_join_reject	방 입장이 거절됨을 알리고 거절 사유 코드를 보낸다.	
클->서	request_room_leave	방 나가기를 요청한다.	checkroom(), leaveRoom()
서->클	room_player_leave	방에 있는 유저들에게 나간 유저 정보를 보낸다.	
클->서	request_game_start	게임 조건을 보내고 게임 시작을 요청한다.	startGame(), makeMap(), createVertex(), excludeVertex(), createVertex(), excludeVertex()
서->클	game_start	게임이 시작됨을 알리고 정해진 게임 정보를 보낸다.	
클->서	request_role_select	선택한 역할을 요청한다.	selectRole(), allSelected()
서->클	role_select	게임 유저들에게 유저가 선택한 역할을 보낸다.	
서->클	game_role_data	게임 유저들에게 모든 유저의 역할 정보를 보낸다.	
클->서	request_map_data	맵 정보를 요청한다.	
서->클	game_map_data_start	맵 정보 보내기가 시작됨을 알린다.	
서->클	vertex_create	맵의 Vertex 정보를 보낸다.	
서->클	edge_create	맵의 Edge 정보를 보낸다.	
서->클	game_map_data_end	맵 정보 보내기가 끝남을 알린다.	
클->서	request_agent_create	말 생성을 요청한다.	createPlayer()
서->클	agent_create	게임 유저들에게 생성된 말 정보를 보낸다.	

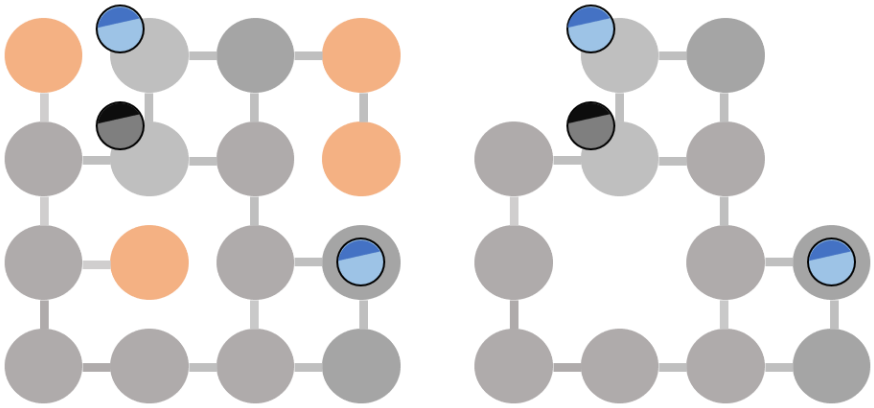
클->서	Request_turn_start	턴 시작을 요청한다.	findTurn()
서->클	Agent_move_turn	게임 유저들에게 이동할 차례의 말 정보를 보낸다.	
클->서	Request_agent_move	말 이동을 요청한다.	movePlayer(), arrested(), gameWon(), findTurn()
서->클	Agent_move	게임 유저들에게 말 이동 정보를 보낸다.	
서->클	Agent_caught	게임 유저들에게 말이 잡혔음을 알리고 잡힌 말 정보를 보낸다.	
서->클	Game_end	게임 유저들에게 게임이 끝남을 알리고 승패 정보를 보낸다.	

Cop 알고리즘 개발

해당 과제에서는 Cop이 Robber를 잡는 효율적인 알고리즘을 연구하였다.

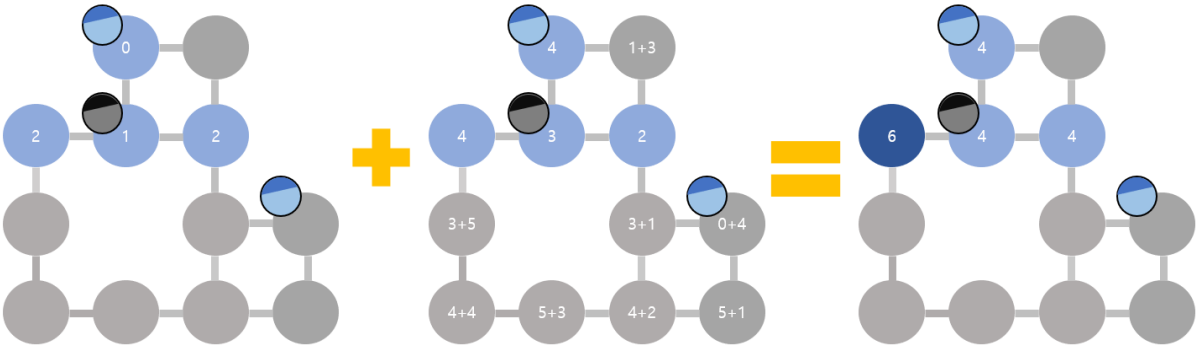
Robber 알고리즘

우선 시뮬레이션을 위한 Robber 알고리즘을 작성할 필요가 있다. Robber 알고리즘은 특별한 전략 없이 간단히 모든 경찰로부터 최대한 멀어지는 경로를 택하도록 한다.



<그림> 그래프에 존재하는 Pitfall(주황색)을 모두 제거

우선 그래프 상에서 막다른 길을 모두 제거한다. 이는 노드의 Indegree가 1인 노드가 존재하지 않을 때 까지 순차적으로 제거함으로써 이뤄낼 수 있다.



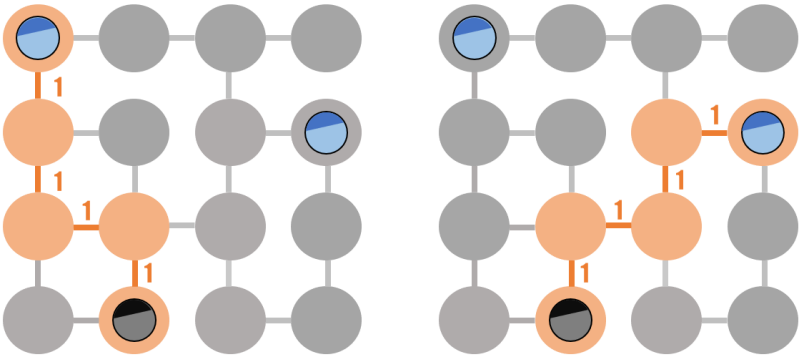
<그림> Robber 기준 각 노드의 비용 계산

이후 각 노드에 대해 모든 Cop으로부터의 거리의 합을 계산한다. Robber로부터 인접한 노드 중 합의 비용이 가장 높은 노드를 다음 경로로 선택한다.

Cop 알고리즘

Algorithm A

Cop의 가장 기본적인 알고리즘인 Algorithm A는 각 Cop들이 Robber를 향한 최단 경로를 탐색하고 선택된 해당 탐색 경로를 통해 이동하는 것이다. 각 Cop 마다 독립적으로 Dijkstra 알고리즘을 통해 최단 경로를 탐색한다. 이 때 각 Edge의 비용은 1이다.



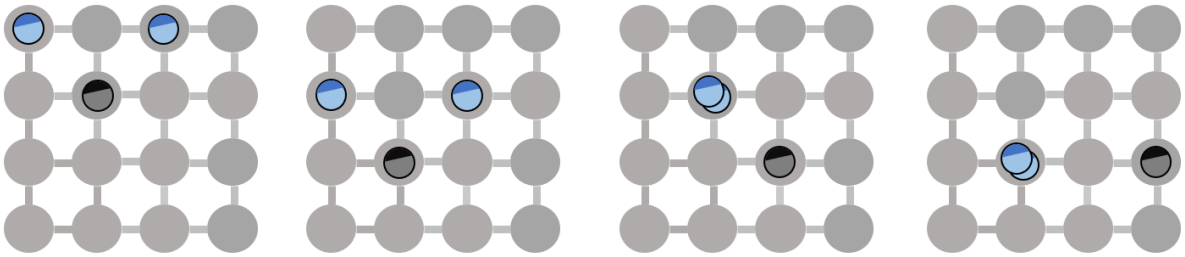
<그림> Algorithm A의 경로 선택 전략

Algorithm A의 Pseudo code는 다음과 같다. 일반적인 Dijkstra algorithm과 동일하다.

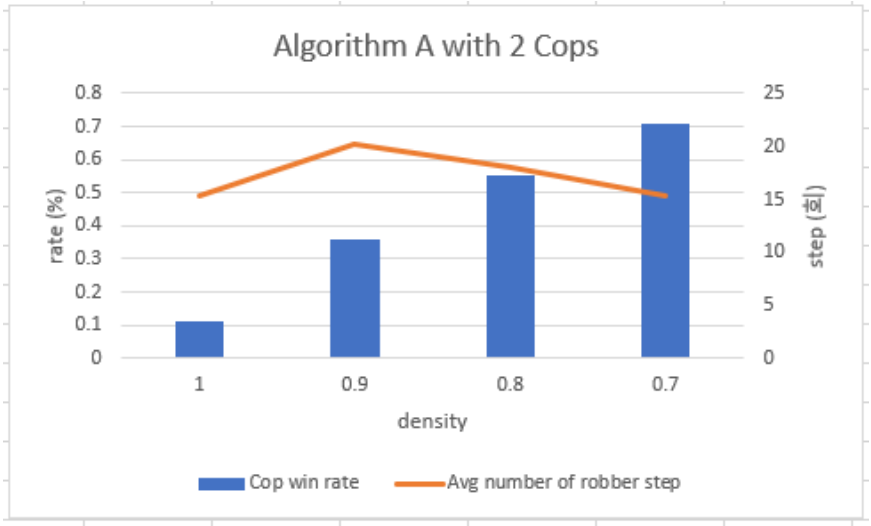
```
function AlgorithmA(Graph):

     create vertex set Q
     source ← position of robber

     for each vertex v in Graph:
         dist[v] ← INFINITY
         prev[v] ← UNDEFINED
         add v to Q
     dist[source] ← 0

     while Q is not empty:
         u ← vertex in Q with min dist[u]
         remove u from Q
         for each neighbor v of u:           // only v that are still in Q
             alt ← dist[u] + 1
             if alt < dist[v]:
                 dist[v] ← alt
                 prev[v] ← u

     return dist[], prev[]
```

<코드> Algorithm A의 Pseudo code

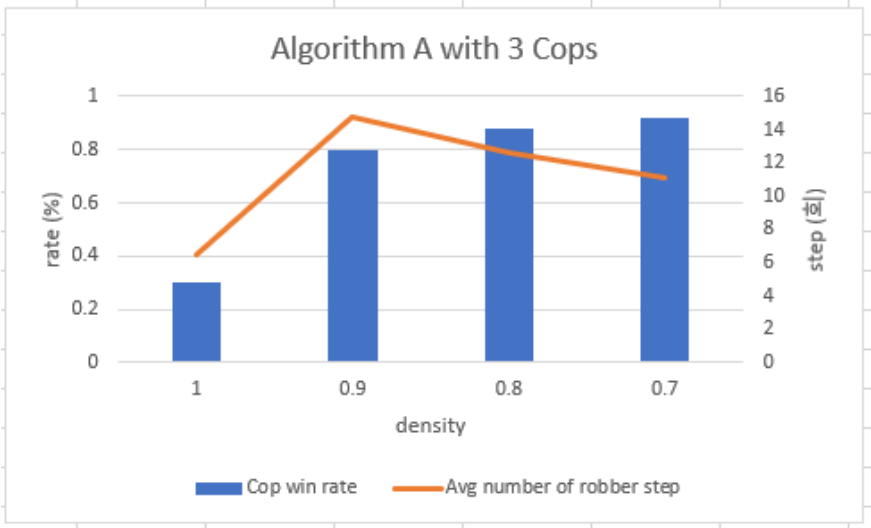
해당 알고리즘의 한계는 매우 명확한데, 여러 개의 Cop의 경로 선택 시 중복된 Edge가 존재할 경우 결국 해당 Cop들이 특정 시점부터 같은 위치에 위치하게 되어 여러 Cop들이 모두 같은 경로를 통해 움직이게 된다는 한계가 존재한다.



<그림> Algorithm A의 취약점



<표 1> Algorithm A 및 2명의 Cops 상황에서의 그리드 밀도에 따른 결과



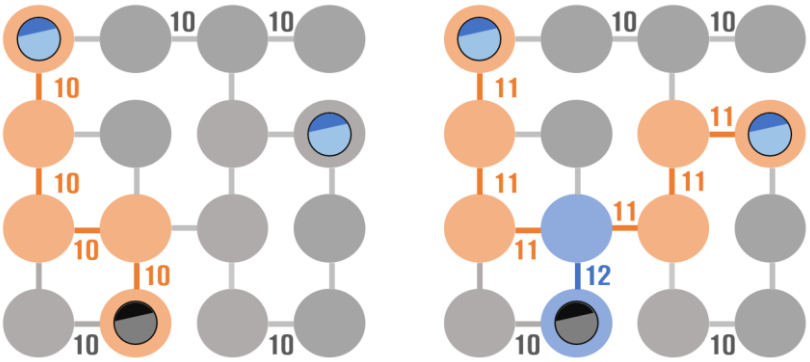
<표 2> Algorithm A 및 3명의 Cops 상황에서의 그리드 밀도에 따른 결과

결과 그래프를 보면 알 수 있듯이, 높은 밀도를 가진 그리드의 Cop 승리 비율이 50% 미만을 밑도는 것을 볼 수 있다. 또한 특이한 점은 Cop 승리 시 평균 이동 횟수가 밀도와 비례하지도, 반비례하지도 않는 모습을 보이고 있는데, 이는 이동 횟수가 Cop과 Robber의 초기 배치 위치에 영향을 크게 받는다는 의미으로 해석할 수 있다.

Algorithm B

Algorithm A의 단점을 보완하기 위해 Algorithm B는 다음과 같이 동작한다.

1. 각 Cop들은 차례대로 도둑까지의 최단 경로를 탐색한다.
2. 각 Edge의 비용은 10이다.
3. 탐색된 최단 경로에 대한 Edge에는 각 1의 비용을 추가한다.



<그림> Algorithm B의 경로 선택 전략

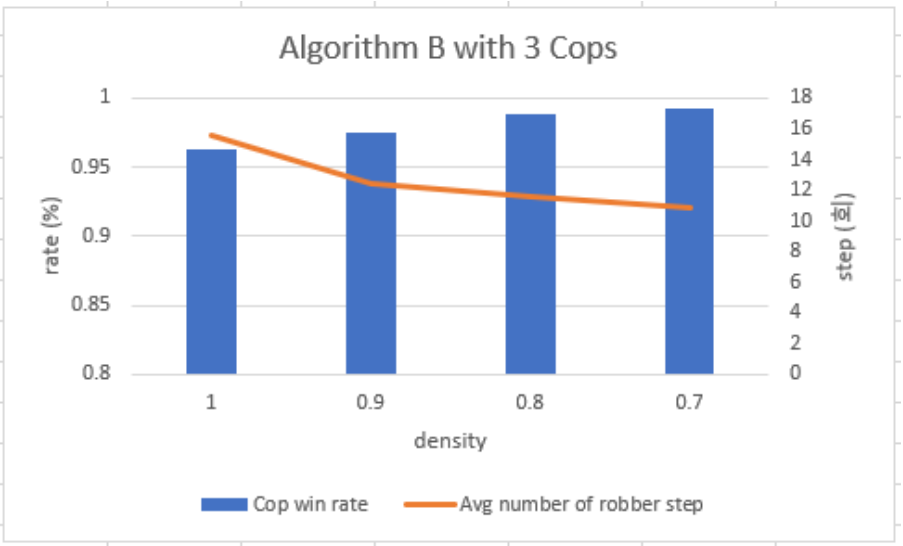
결과적으로 Algorithm B의 경우 Cop의 경로가 과도하게 중복되지 않도록 하여 최대한 각 Cop들의 경로를 다양화하는 결과를 만들어낸다.

다음은 Algorithm B의 Pseudo code이다. 각 edge의 비용을 저장하는 ***EdgeCostMap***이 추가된 모습을 볼 수 있다.

```
function AlgorithmB(Graph, EdgeCostMap, Cop):
      // ...
      // Same with AlgorithmA
      // ...
```

```
    for each edge e in Cop.path:
        EdgeCostMap[e] ← EdgeCostMap[e] - 1

    while Q is not empty:
        u ← vertex in Q with min dist[u]
        remove u from Q
        for each neighbor v of u:
            alt ← dist[u] + EdgeCostMap[(u, v)] + 10
            else if alt < dist[v]:
                dist[v] ← alt
                prev[v] ← u

    Cop.path ← []
    track ← prev[dest]
    do:
        v0 ← track
        v1 ← prev[track]
        e ← (v0, v1)
        add (v0, v1) to Cop.path
        EdgeCostMap[e] ← EdgeCostMap[e] + 1
        track ← prev[track]
    while track is not source

    return dist[], prev[]
```

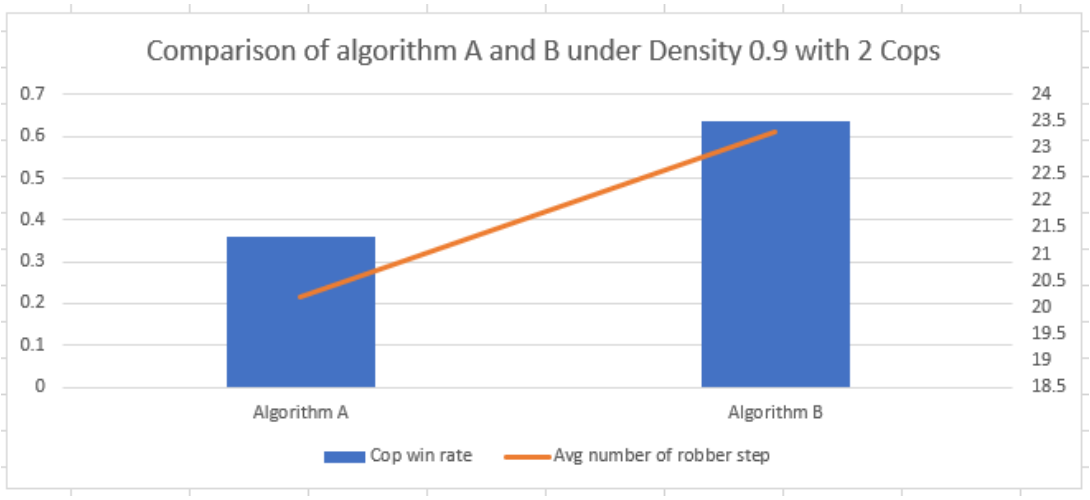
<코드> Algorithm B의 Pseudo code



<표> Algorithm B 및 3명의 Cops 상황에서의 그리드 밀도에 따른 결과

각 Cop들의 경로에 대해 가중치만 부여하여도 이미 3 Cops 가 존재하는 상황에서는 모든 밀도에

서 95% 이상의 승리 비율을 보여주고 있다. 또한 주목할 점은 Algorithm A의 경우 밀도와 잡히는데 소요되는 스텝의 수 간의 상관 관계를 찾아보기 힘들었지만, Algorithm B의 경우 밀도와 잡히는데 소요되는 스텝의 수가 비례하는 것을 볼 수 있다.

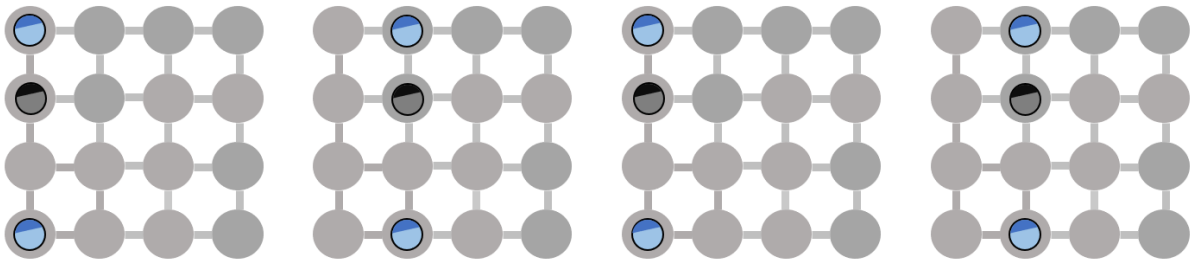


<표> 밀도 0.9 및 2명의 Cop 환경에서 Algorithm A와 B의 결과 비교

표는 2명의 Cop과 0.9의 밀도에서 Algorithm A와 B의 결과를 비교하고 있다. Algorithm A에 비해 Algorithm B에서 Cop의 승리 확률이 확연하게 상승한 것을 확인할 수 있다.

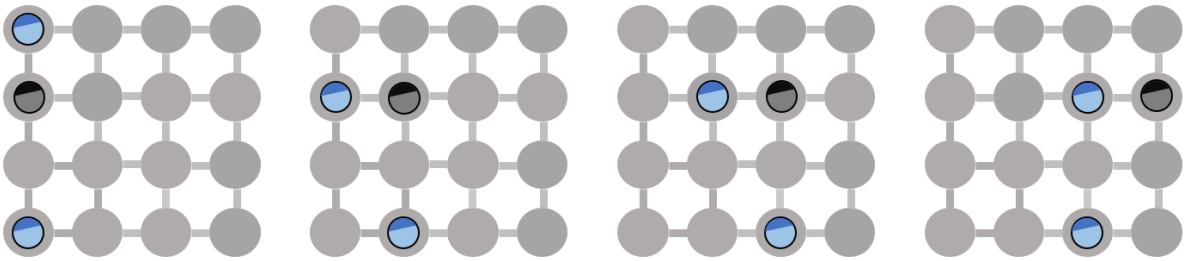
3명의 Cop이 존재하는 경우에서 Algorithm B는 1.0의 밀도에서도 96%라는 높은 승리 확률을 기록하고 있다. 그러나 2명의 Cop이 존재하는 경우에서는 38%라는 여전히 50% 미만의 낮은 승리 확률을 기록하고 있다.

2명의 Cop이 존재하는 상황에서 Algorithm B가 가지는 취약점 중 하나는 다음과 같은 문제이다.



<그림> Algorithm B의 취약점

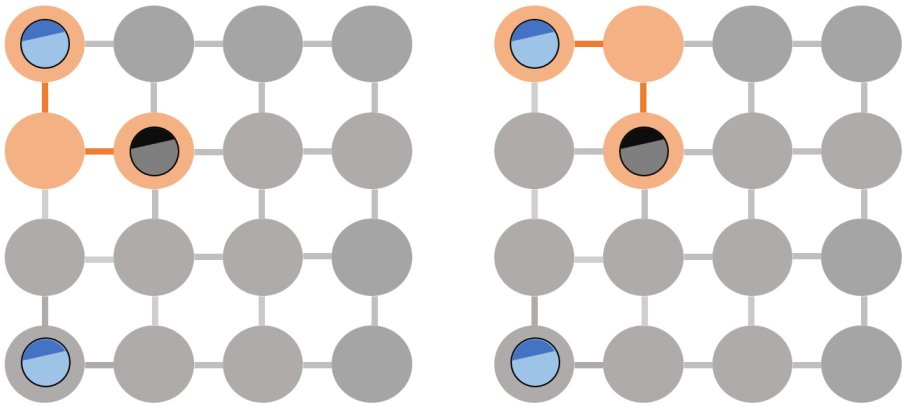
그림은 Robber가 좌우로 반복해서 움직이는 상황에서 2명의 Cop 역시 모두 좌우로 반복적으로 움직이는 상황을 보여주고 있다. 위의 상황이 계속 반복되어 결국 Robber의 승리로 끝나는 경우가 가장 큰 문제로 작용하였다.



<그림> Algorithm B의 개선 방향

Algorithm C

Algorithm B의 문제를 해결하기 위해 Cop은 최대한 Robber를 구석으로 몰아가도록 움직임을 변경할 필요가 있다.



<그림> 같은 비용을 가지는 여러 개의 경로가 존재하는 상황

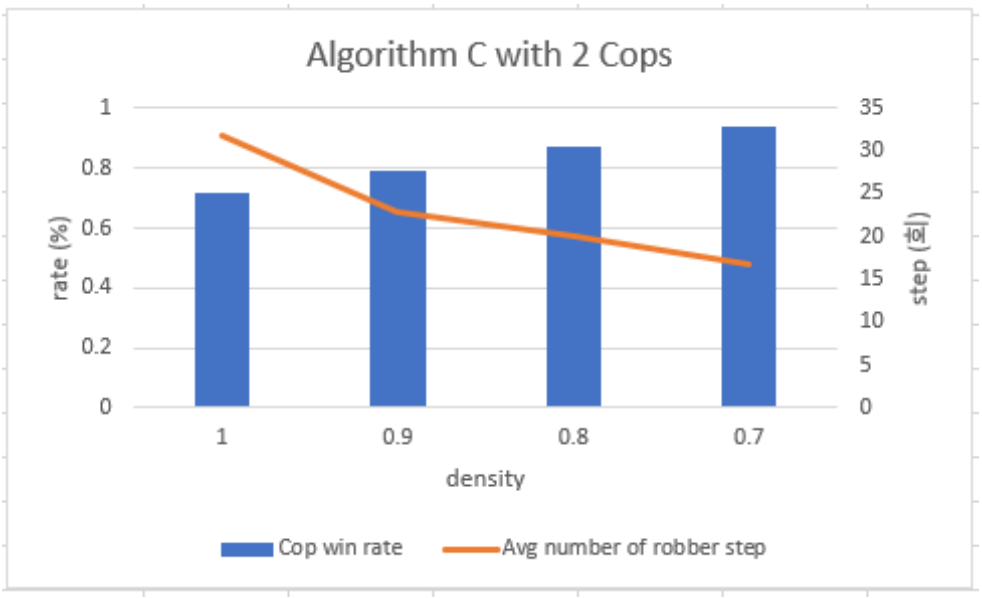
위 동작을 구현하는 방법은 여러가지가 있지만, 가장 간단한 방법은 경로를 선택하는데 있어 랜덤 요소를 가미하는 것이다. <그림>을 보면, Cop은 2가지의 동일한 비용을 가진 가장 짧은 경로를 가지고 있다. 기존 Dijkstra 알고리즘을 이용한다면 어떻게 알고리즘을 구현했는가에 차이가 존재할 뿐 항상 두 경로 중 하나의 경로만 선택하게 될 것이다. 결국 동일한 상황 하에서 동일 비용의 선택 가능한 경로가 여러 개 존재하더라도 항상 같은 경로를 선택하게 됨으로써 문제가 발생하게 되는데, 이 문제를 해결하기 위해 동일 비용의 경로가 N개 존재할 경우 해당 경로의 선택 확률을 $\frac{1}{N}$로 두어 매번 다른 경로를 선택하도록 만들어 위에 제시된 문제를 해결하였다.

다음은 Algorithm C의 Pseudo code이다. Algorithm B에서 변경된 일부분만을 보여주고 있다.

```
while Q is not empty:
        u ← vertex in Q with min dist[u]

        remove u from Q

        for each neighbor v of u:
```

```
            alt ← dist[u] + EdgeCostMap[(u, v)] + 10
            if alt == dist[v]:
                chance ← random value between 0 ~ 1.0
                if chance > 0.5:
                    dist[v] ← alt
                    prev[v] ← u
            else if alt < dist[v]:
                dist[v] ← alt
                prev[v] ← u
```

<코드> Algorithm C의 Pseudo code

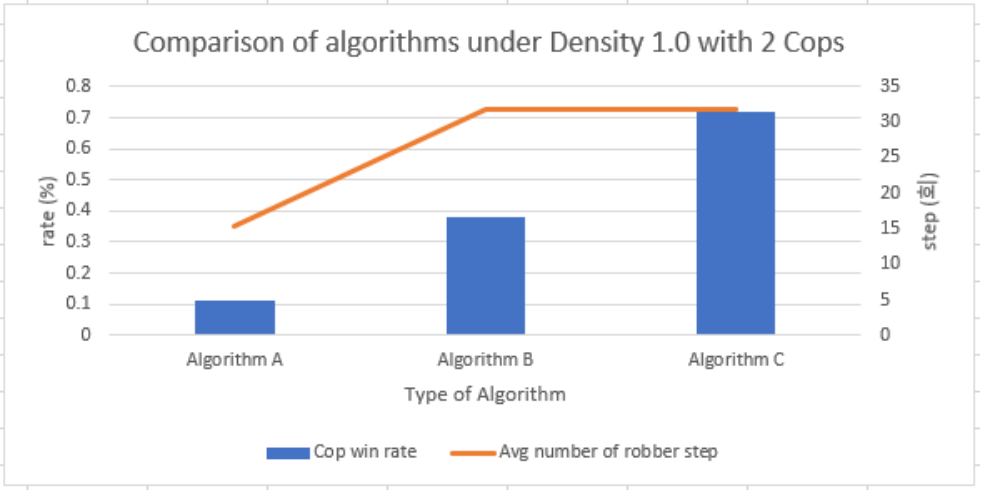


<표> Algorithm C 및 2명의 Cops 상황에서의 그리드 밀도에 따른 결과

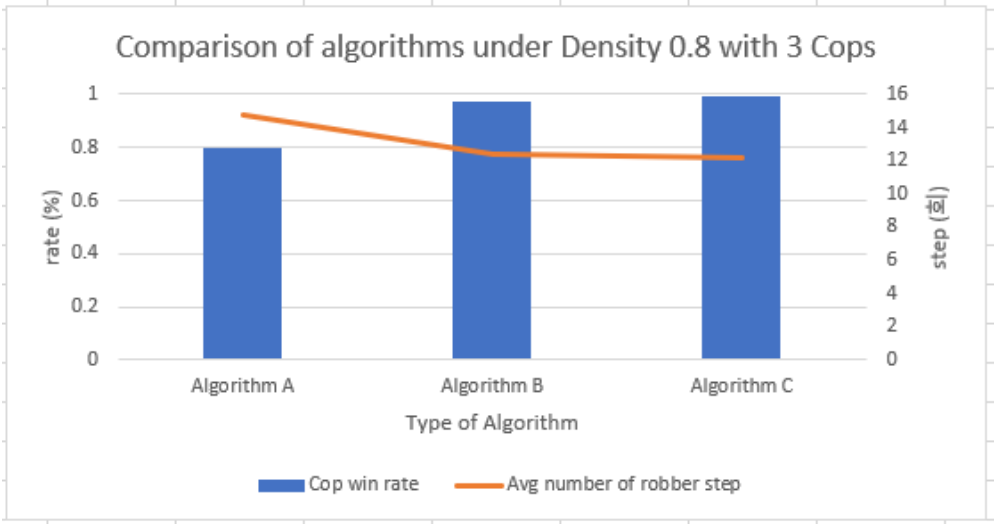
<표>는 2명의 Cop이 존재하는 상황에서 Algorithm C의 결과를 나타내고 있다. 모든 밀도에서 승률 70% 이상을 기록하여 준수한 결과를 보여주고 있는 것을 확인할 수 있다. 또한 밀도가 낮아질 수록 Robber를 잡는데 필요한 횟수가 적어지는 모습을 볼 수 있다.

Overall Result

해당 과제에서는 효율적인 Cop의 승리를 위한 알고리즘을 A, B, C 단계로 나누어 설명하였다. 가장 기본적인 Dijkstra 알고리즘인 Algorithm A부터 시작하여 각 Edge에 가중치를 부과하여 Algorithm A를 개선한 Algorithm B, 그리고 마지막으로 랜덤성을 추가하여 Algorithm B를 개선한 Algorithm C까지 살펴보았다.



<표> 밀도 1.0, 2 Cops 상황에서의 각 Algorithm의 결과 비교



<표> 밀도 1.0, 3 Cops 상황에서의 각 Algorithm의 결과 비교

위 두 <표>는 각 상황에서의 Algorithm A, B, C의 결과를 비교하고 있다. 표에서 볼 수 있듯이 Algorithm A, B, C 순으로 결과가 얼마나 개선되는지 확인할 수 있다. 또한 <표>는 밀도 1.0 과 2 명의 Cop 상황에서의 결과를 비교하고 있는데, 승률이 각 11%, 38% 로 저조한 Algorithm A, B와 달리 Algorithm C는 72%의 승률이라는 준수한 결과를 보여주고 있는 것을 볼 수 있다.

반면 3명의 Cop이 존재하는 상황에서 Algorithm A는 확연히 저조한 모습을 보여주고 있지만, Algorithm B와 C는 비슷한 결과를 보여주고 있는 것을 확인할 수 있다. 따라서 Algorithm C는 2 명의 Cop 상황에서 발생하는 문제를 효율적으로 해결하였다고 해석할 수 있다.

멘토링 결과 반영

해당 과제는 ㈜페이보리 의 멘토링 하에 진행되었다. 멘토링 내용은 다음과 같다.

- AI COM과 대결할 수 있는 알고리즘 구현

  논문을 바탕으로 하여 Cop 알고리즘을 A, B, C 세 단계로 개발하였다.

- 기본 UI들을 활용하여 깔끔한 디자인 개발

  기존 디자인을 수정하여 깔끔하고 모던한 디자인을 적용하도록 하였다.

결과 및 발전 가능성

결과

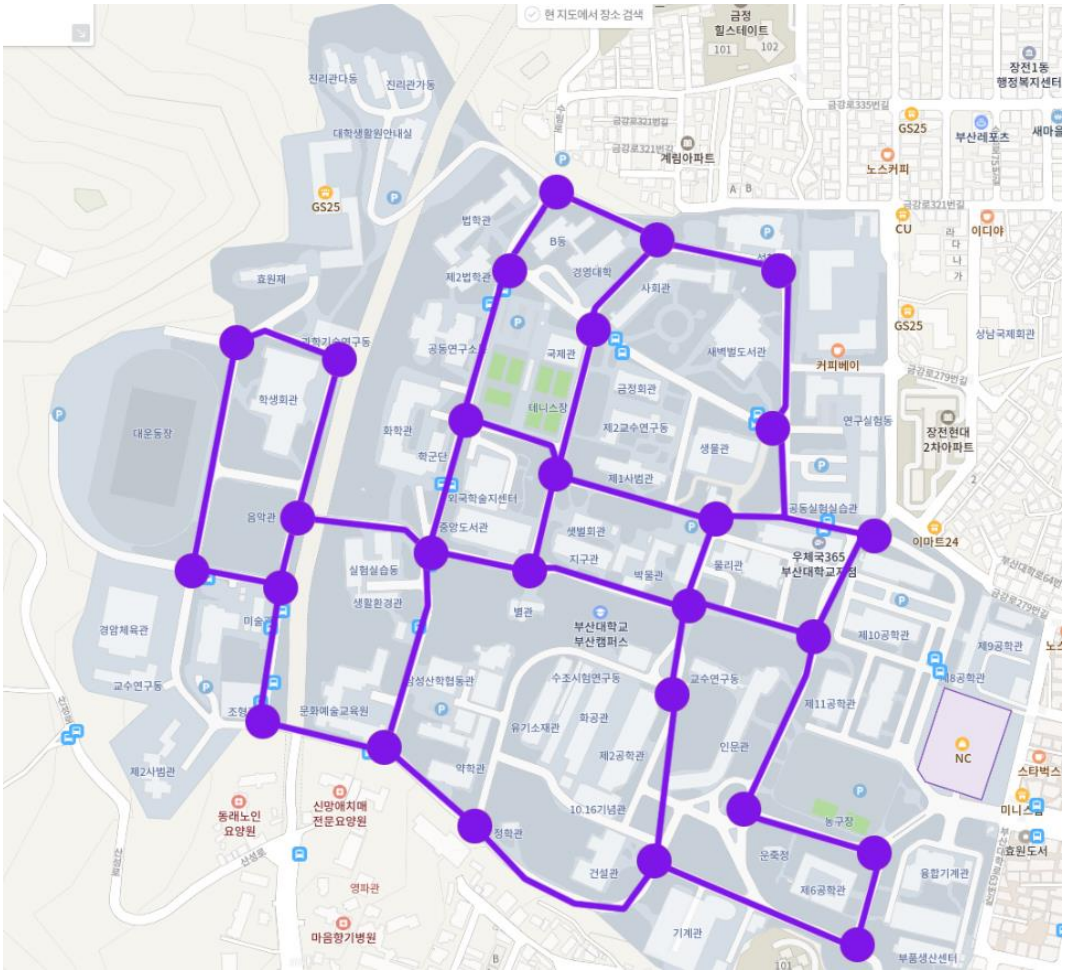
해당 과제에서는 Cops and Robbers Game Theory를 기반으로 해당 게임 이론을 온라인 상을 통하여 실제 사용자들끼리 대전을 할 수 있도록 Cosp and Robbers Game을 구현하였다. 이를 통해 사용자들이 Cops and Robbers 게임 이론에 대해 이해하고, 직접 게임을 플레이하면서 어떤 상황 또는 어떤 전략을 이용하여야 효율적으로 Cop 또는 Robber가 승리할 수 있는지 생각해볼 수 있도록 하였다.

또한 해당 과제에서 Cop에 대한 단계적으로 발전된 3가지 알고리즘을 제시함으로써 시뮬레이션을 통하여 해당 알고리즘이 그래프의 밀도 및 Cop의 수에 따라 어느 만큼 효율적으로 Cop-Win 상황으로 이끄는지 분석해 보았다.

발전 가능성

실제 지도상에서의 게임

해당 과제에서의 그래프는 별 다른 의미 없는 N x M 그리드를 통해 게임을 진행하고 있다. 이를 응용하여 실제 지도를 그래프로 표현하고 해당 그래프 상에서 게임을 진행할 수 있다면 현실 세계에서 실제로 경찰과 도둑이 쫓고 도망치는 모습을 시뮬레이션 할 수 있을 것으로 기대된다.
예를 들어 아래 <그림>은 부산대학교 지도를 그래프로 나타낸 모습을 보여준다.



<그림> 부산대학교 지도를 그래프로 표현한 모습

Neural network 적용

해당 게임은 바둑처럼 그리드 상에서 진행이 되므로 Convolutional Neural Network를 적용할 수 있을 것으로 기대된다. AlphaGo와 마찬가지로 각 지점에 필요한 Feature들을 제공하여 학습시킨다면 기대 할 만한 결과가 나오지 않을까 기대된다. 다음은 CNN을 적용할 경우 제공되는 Feature의 예상 목록이다.

Feature	# of planes	Description
Agent type	3	Cop / Robber / Empty
Edge State	16 (2^4)	상/하/좌/우 이웃과의 연결 상태
Turns since	8	현재까지 진행된 턴 횟수

<표> Cops and Robbers CNN 적용에 필요한 Input features

추진체계 및 일정

월	6		7					8				9			
주	3	4	1	2	3	4	5	1	2	3	4	1	2	3	4
개발방향 및 일정확인															
전체 설계															
클라이언트 구현															
서버구현															
전략 알고리즘 분석개발															
수정 및 보완															
최종발표 준비															

구성원 역할

임학천	게임서버 구현	완료
	COM 구현	완료
	전략 알고리즘 분석, 개발	완료
송태영	클라이언트 인게임 구현	완료
	전략 알고리즘 분석, 개발	완료
배선우	전체 구조 설계	완료
	클라이언트 설계	완료
	전략 알고리즘 분석, 개발	완료

참고문헌

M. Aigner, M.Fromme (1982), A Game of cops and robbbers

Wikipedia, Pursuit-evasion
https://en.wikipedia.org/wiki/Pursuit-evasion

PBS Infinite Series, The Cops and Robbers Theorem | Infinite Series
https://www.youtube.com/watch?v=9mJEu-j1KT0