자료구조 프로젝트 보고서

19-042 박용원, 19-062 윤준서

1. 개요

a. 프로젝트명

온라인 쿼리도 웹사이트 제작

(https://dsquaridor.herokuapp.com/)

(시연영상: https://youtu.be/FnwRrYcm4f0)

b. 프로젝트(시스템) 구성

플레이어 두명이 온라인으로 접속하여 쿼리도 보드게임을 플레이할 수 있는 웹사이트이다. 채팅이 부가적으로 구현되어 있다.

c. 동기 및 목적

최근 몇년동안 부루마블, 루미큐브와 같은 유명한 보드게임들이 웹이나 앱 형태로 제작되어 배포되는 것이 성행하고 있다. 본 프로젝트에서는 멘사 셀렉트 보드게임인 쿼리도를 플레이할 수 있는 웹사이트를 제작하여 쿼리도를 온라인으로 플레이할 수 있게 하고자 하였다. 쿼리도 웹사이트를 제작하면서 이루고자 했던 목표들은 다음과 같다. 먼저 보드게임을 플레이하면서 발생할 수 있는 예외 상황들을 코딩을 통해 처리하면서 다양한 상황에 대한 예외처리 경험과 사고력을 기르고자 하였다. 또한, 새로운 언어인 javascript를 배우고 그 node.js를 활용해 웹사이트를 만들면서 웹에 대한 이해를 높이고자 하였다. 마지막으로, 플레이어 간의 소통을 구현하면서 소켓 프로그래밍에 대한 전반적인 이해를 높이고자 하였다

2. 쿼리도(Quoridor)

- 쿼리도란?

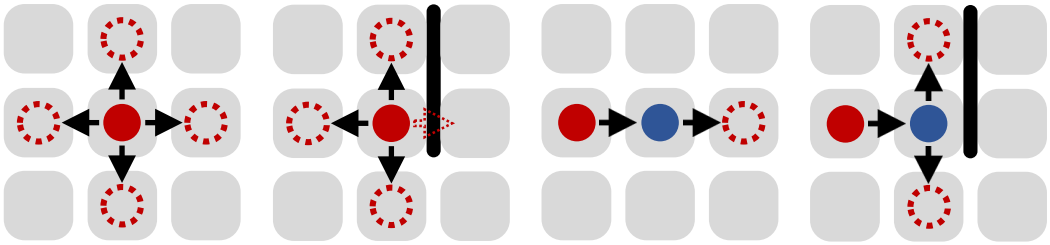
쿼리도(Quoridor)는 이탈리아의 비디오 게임 개발자 미르코 마르케가 만들었으며, 1997년 멘사 셀렉트로 게임으로 선정되었다.

- 쿼리도 규칙

게임은 9x9 판 위에서 진행되며, 각자 10개의 2칸 크기의 장애물을 가지고 시작한다. 자신의 턴에는 한 번의 이동을 하거나, 한 개의 장애물을 배치할 수 있다. 게임의 목표는 자신이 출발한 지점의 정반대에 있는 줄에 도착하는 것이다.

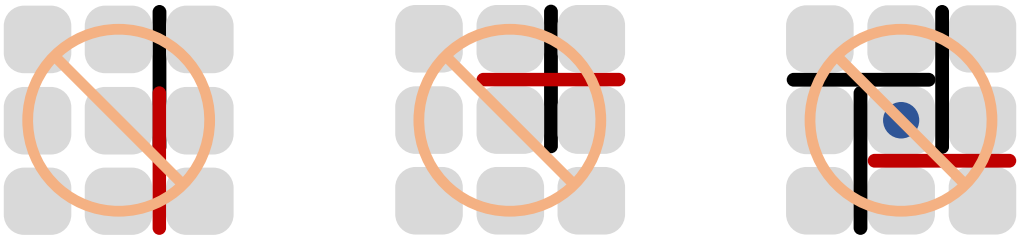
이동의 경우 장애물로 막혀 있지 않은 경우 상하좌우로 이동 가능하며 나와 상대방이 인접하게 위치한 경우 뛰어넘을 수 있다. 또한 뛰어 넘으려 하는 위치가 장애물로 가로막혀 있으면 대각

선으로 이동할 수 있다.



<그림 1> 쿼리도에서 말의 가능한 이동

장애물은 서로 겹치지 않고 상대방을 목표 지점에 도착하지 못하게 가두지만 않는다면 칸과 칸 사이에 가로 혹은 세로로 어디든 놓을 수 있다.



<그림 2> 쿼리도에서 장애물을 놓을 수 있는 위치

3. 설계 및 제작

- 웹사이트 설계

본 프로젝트에서 설계한 웹사이트가 가져야 하는 특징들은 다음과 같다.

a. 플레이어 두 명이 온라인으로 쿼리도를 플레이 할 수 있어야 한다.

b. 플레이어들은 클릭을 통해 자신의 말을 움직일 수 있어야 한다.

c. 커서를 게임판 위에 올리면 자신이 할 수 있는 선택지를 미리 볼 수 있어야 한다.

d. 플레이어 간에 채팅을 할 수 있게 해야 한다.

- 제작 과정

온라인 쿼리도 웹사이트의 제작 과정은 다음과 같다.

a. python으로 쿼리도 게임(로직) 구현

b. 구현된 게임을 javascript로 번역

c. Socket.io 모듈을 사용하여 플레이어 간의 통신 구현

d. 쿼리도의 board, player 등을 그리는 visualization 및 input manager 구현

e. 채팅 구현

f. Heroku를 통해 웹 애플리케이션 가동

- 역할 분담

  박용원: 웹사이트 제작

  윤준서: 쿼리도 구현

4. 구현

a. 쿼리도 구현

먼저 쿼리도의 게임을 Python을 통해 구현하였다. 게임은 크게 Player 클래스와 Board 클래스로 이루어져 있다.

Player 클래스는 현재의 위치, 현재의 블록 개수, 자신의 위치를 특정 방향으로 옮기는 함수, 블록을 가지고 있는지 알려주는 함수가 있다.

```
class Player:
    def __init__(self, direction):
        assert direction in (0, 1)
        self.block = 10
        self.position = [direction * 8, 4]
        self.moving_dir = (-1) ** direction

    def move(self, direction):
        self.position[0] += direction[0]
        self.position[1] += direction[1]

    def does_have_block(self):
        return self.block != 0
```

<코드 1> Player 클래스

Board 클래스에서는 Player을 가능한 방향으로 움직이는 것, 블록을 가능한 위치에 놓는 것, 턴을 진행하는 것이 필요하다. 판의 사이즈를 나타내는 SIZE 상수, 블록을 놓는 방향을 나타내는 HORIZONTAL과 VERTICAL 상수를 포함한다. 처음 Board를 생성하면 장애물이 어떻게 배열됐는지를 나타내는 8x8의 block_board를 만든다. 처음에는 장애물이 전혀 없기 때문에 모두 None으로 설정했다. block_board는 왼쪽 위를 (0, 0)으로 생각하며, 아래 방향이 첫 번째 좌표, 오른쪽 방향이 두 번째 좌표의 증가를 의미한다.

```
class Board:
    SIZE = 9
```

```
    HORIZONTAL = 0
    VERTICAL = 1

    def __init__(self):
        self.block_board = [[None] * (Board.SIZE - 1) for i in range(Board.SIZE - 1)]
        self.players = [Player(0), Player(1)]

    def is_finished(self):
        return self.players[0].position[0] == 8 or self.players[1].position[0] == 0
```

<코드 2> Board 클래스

move 함수에서는 입력에 따라 플레이어를 이동시킨다. 규칙에 따라 가능한 움직임이 있기 때문에 먼저 possibl_direction_at을 이용해 장애물로 막히지 않은 곳을 임시로 갈 수 있는 곳으로 지정하고, possible_moves 함수를 통해 상대의 위치에 따라 가능한 움직임으로 리스트를 수정하여 반환한다.

```
    def move(self, player, direction):
        if direction in self.possible_moves(player):
            self.players[player].move(direction)
            return True
        else:
            print("You cannot move to here")
            return False

    def possible_moves(self, player):
        assert player in (0, 1)
        pos = self.players[player].position
        opp_pos = self.players[1 - player].position
        moves = self.possible_direction_at(pos)
        direction = (opp_pos[0] - pos[0], opp_pos[1] - pos[1])
        if direction in moves:
            moves.remove(direction)
            zero = direction.index(0)
            nonzero = 1 - zero
            if Board.SIZE - abs(direction[0]) > opp_pos[0] >= abs(direction[0]) and
Board.SIZE - abs(direction[1]) > opp_pos[1] >= abs(direction[1]):
                to_check = [opp_pos[0] - 0.5 + 0.5 * direction[0], opp_pos[1] - 0.5 +
0.5 * direction[1]]
                if self.block_board[round(to_check[0] + 0.5 *
direction[1])][round(to_check[1] + 0.5 * direction[0])] != nonzero and
self.block_board[round(to_check[0] - 0.5 * direction[1])][round(to_check[1] - 0.5 *
direction[0])] != nonzero:
                    moves.append((direction[0] * 2, direction[1] * 2))
                else:
                    if self.block_board[round(to_check[0] + 0.5 *
direction[1])][round(to_check[1] + 0.5 * direction[0])] != zero and
self.block_board[round(to_check[0] + 0.5 * direction[1]) -
direction[0]][round(to_check[1] + 0.5 * direction[0]) - direction[1]] != zero:
                        moves.append((direction[nonzero], direction[nonzero]))
                    if self.block_board[round(to_check[0] - 0.5 *
direction[1])][round(to_check[1] - 0.5 * direction[0])] != zero and
```

```
self.block_board[round(to_check[0] - 0.5 * direction[1]) -
direction[0]][round(to_check[1] - 0.5 * direction[0]) - direction[1]] != zero:
                    moves.append(((-1) ** nonzero * direction[nonzero], (-1) ** zero
* direction[nonzero]))
        return moves

    def possible_direction_at(self, position):
        dirs = []
        [x, y] = position
        if x >= 1:
            flag = True
            if y < Board.SIZE - 1 and self.block_board[x - 1][y] == Board.HORIZONTAL:
                flag = False
            if y > 0 and self.block_board[x - 1][y - 1] == Board.HORIZONTAL:
                flag = False
            if flag:
                dirs.append((-1, 0))
        if x < Board.SIZE - 1:
            flag = True
            if y < Board.SIZE - 1 and self.block_board[x][y] == Board.HORIZONTAL:
                flag = False
            if y > 0 and self.block_board[x][y - 1] == Board.HORIZONTAL:
                flag = False
            if flag:
                dirs.append((1, 0))
        if y >= 1:
            flag = True
            if x < Board.SIZE - 1 and self.block_board[x][y - 1] == Board.VERTICAL:
                flag = False
            if x > 0 and self.block_board[x - 1][y - 1] == Board.VERTICAL:
                flag = False
            if flag:
                dirs.append((0, -1))
        if y < Board.SIZE - 1:
            flag = True
            if x < Board.SIZE - 1 and self.block_board[x][y] == Board.VERTICAL:
                flag = False
            if x > 0 and self.block_board[x - 1][y] == Board.VERTICAL:
                flag = False
            if flag:
                dirs.append((0, 1))
        return dirs
```

<코드 3> 움직임에 관련된 함수

put_block 함수는 장애물을 지정한 위치에 놓는 함수이다. 마찬가지로 장애물을 놓는 규칙이 있기 때문에 _can_put_block_at 함수를 이용하여 어떤 위치에 어떤 방향으로 블록을 놓을 수 있는지 없는지 판단한다. 특히 상대를 가둘 수 없다는 규칙은 _can_reach 함수에서 복제된 보드에 put_block_inst 함수를 사용해 블록을 임시로 배치하고 DFS를 이용해 각 플레이어가 목표 지점에 도달할 수 있는지 판단하였다.

```
    def put_block_inst(self, pos, orientation):
```

```
        if self.block_board[pos[0]][pos[1]] is not None:
            pass
        if pos[orientation] < Board.SIZE - 2 and self.block_board[pos[0] + (orientation
+ 1) % 2][pos[1] + orientation % 2] == orientation:
            pass
        if pos[orientation] > 1 and self.block_board[pos[0] - (orientation + 1) %
2][pos[1] - orientation % 2] == orientation:
            pass
        self.block_board[pos[0]][pos[1]] = orientation

    def _can_reach(self, player, position, orientation):
        inst_board = deepcopy(self)
        inst_board.put_block_inst(position, orientation)
        pos = inst_board.players[player].position
        visited = [pos]
        to_visit = [pos]
        while len(to_visit) > 0:
            pos = to_visit.pop()
            dirs = inst_board.possible_direction_at(pos)
            for direction in dirs:
                next_pos = [pos[0] + direction[0], pos[1] + direction[1]]
                if next_pos not in visited:
                    visited.append(next_pos)
                    to_visit.append(next_pos)
        for i in range(9):
            if [8 - player * 8, i] in visited:
                return True
        return False

    def _can_put_block_at(self, pos, orientation):
        if self.block_board[pos[0]][pos[1]] is not None:
            return False
        if pos[1 - orientation] < Board.SIZE - 2 and self.block_board[pos[0] +
orientation][pos[1] + 1 - orientation] == orientation:
            return False
        if pos[1 - orientation] > 1 and self.block_board[pos[0] - orientation][pos[1] -
1 + orientation] == orientation:
            return False
        if (not self._can_reach(0, pos, orientation)) or (not self._can_reach(1, pos,
orientation)):
            return False
        return True

    def put_block(self, pos, orientation):
        if self._can_put_block_at(pos, orientation):
            self.block_board[pos[0]][pos[1]] = orientation
            return True
        else:
            print("You cannot put it here")
            return False
```

<코드 4> 장애물에 관련된 코드

turn 함수는 콘솔을 통해 받은 parameter 변수에 따라 적절한 동작을 하는 함수이다. 게임을 플

레이할 때 반복되는 턴을 표현한 함수이다.

```
    def turn(self, player, parameter):
        while True:
            if len(parameter) == 2:
                flag = self.move(player, (int(parameter[1]), int(parameter[2])))
                if flag:
                    break
                else:
                    continue
            elif len(parameter) == 3:
                if self.players[player].does_have_block():
                    flag = self.put_block((int(parameter[1]), int(parameter[2])),
int(parameter[3]))
                    if flag:
                        self.players[player].block -= 1
                        break
                else:
                    print("No blocks")
```

<코드 5> turn 함수

show 함수는 코드가 제대로 짜였는지 확인하기 위해 실제 게임 화면이 어떤 상태인지 표시해주는 함수이다.

```
    def show(self):
        boarder = " ╵╴└╷│┌┴┘─┐┤┬┼"
        strings = [["*", " "] * (Board.SIZE - 1) + ["*"] if i % 2 == 0 else [" "] * (2 *
Board.SIZE - 1) for i in range(2 * Board.SIZE - 1)]
        strings[self.players[0].position[0] * 2][self.players[0].position[1] * 2] = "1"
        strings[self.players[1].position[0] * 2][self.players[1].position[1] * 2] = "2"
        for i in range(2 * Board.SIZE - 1):
            for j in range(2 * Board.SIZE - 1):
                x = i // 2
                y = j // 2
                boarder_shape = 0
                if i % 2 == 1 and j % 2 == 1:
                    if self.block_board[x][y] == Board.HORIZONTAL:
                        boarder_shape += 10
                    if self.block_board[x][y] == Board.VERTICAL:
                        boarder_shape += 5
                    if x + 1 < Board.SIZE - 1 and self.block_board[x + 1][y] ==
Board.VERTICAL:
                        boarder_shape += 4
                    if x - 1 >= 0 and self.block_board[x - 1][y] == Board.VERTICAL:
                        boarder_shape += 1
                    if y + 1 < Board.SIZE - 1 and self.block_board[x][y + 1] ==
Board.HORIZONTAL:
                        boarder_shape += 2
                    if y - 1 >= 0 and self.block_board[x][y - 1] == Board.HORIZONTAL:
                        boarder_shape += 8
                    strings[i][j] = boarder[boarder_shape]
                elif i % 2 == 1:
```

```
                    if x < Board.SIZE - 1 and y < Board.SIZE - 1 and
self.block_board[x][y] == Board.HORIZONTAL:
                        boarder_shape += 10
                    if x < Board.SIZE - 1 and y > 0 and self.block_board[x][y - 1] ==
Board.HORIZONTAL:
                        boarder_shape += 10
                    strings[i][j] = boarder[boarder_shape]
                elif j % 2 == 1:
                    if x < Board.SIZE - 1 and y < Board.SIZE - 1 and
self.block_board[x][y] == Board.VERTICAL:
                        boarder_shape += 5
                    if x > 0 and y < Board.SIZE - 1 and self.block_board[x - 1][y] ==
Board.VERTICAL:
                        boarder_shape += 5
                    strings[i][j] = boarder[boarder_shape]
        for i in range(2 * Board.SIZE - 1):
            for j in range(2 * Board.SIZE - 1):
                q = "  "
                if strings[i][j] in "╶└┌├─┴┬┼" and strings[i][j] in "╶└┌├╴┘─┴┐┤┬┼":
                    q = "──"
                print(strings[i][j], end=q)
            print()
```
<코드 6> show 함수

game 함수는 게임이 끝날 때까지 콘솔로 입력을 받아 턴을 바꾸며 게임을 진행시키는 함수이다. turn 함수를 만들기 이전에 game 함수를 만들어 turn 함수를 사용하지는 않았다.

```
    def game(self):
        print("move dx dy or block x y orientation")
        print("ex) move 0 1 / block 3 7 1")
        current_player = 0
        while not self.is_finished():
            self.show()
            print()
            while True:
                print('Player ', end='')
                print(current_player + 1, end='')
                string = input("\'s turn : ")
                l = string.split(" ")
                if len(l) == 3:
                    flag = self.move(current_player, (int(l[1]), int(l[2])))
                    if flag:
                        break
                    else:
                        continue
                elif len(l) == 4:
                    if self.players[current_player].does_have_block():
                        flag = self.put_block((int(l[1]), int(l[2])), int(l[3]))
                        if flag:
                            self.players[current_player].block -= 1
                            break
                    else:
                        print("No blocks")
```

```
            current_player = 1 - current_player
            print()
            print()
        self.show()
        print()
        print("Player ", end='')
        print(2 - current_player, end='')
        print(' win')
```

<코드 7> game 함수

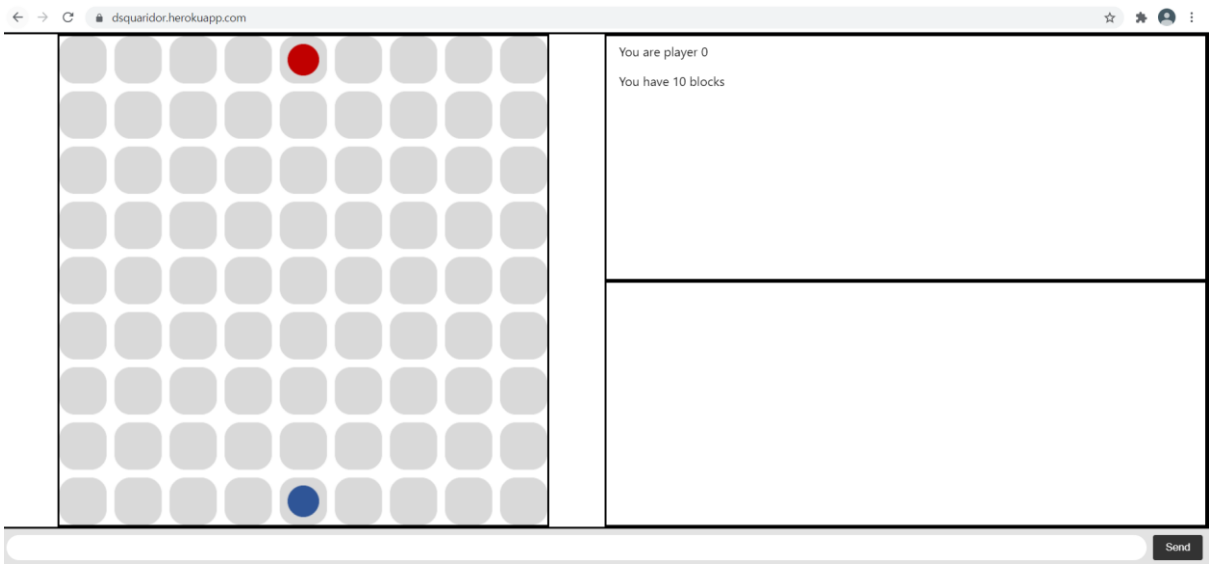
b. 웹사이트 구현

온라인 쿼리도 웹사이트의 제작은 HTML, CSS, JS를 사용하여 이루어졌다. 웹 앱 제작을 위해 js의 플랫폼인 node.js와 그 프레임워크인 express.js를 사용하였고, 플레이어 간의 통신을 구현하기 위해 socket.io 모듈을 사용하였다.

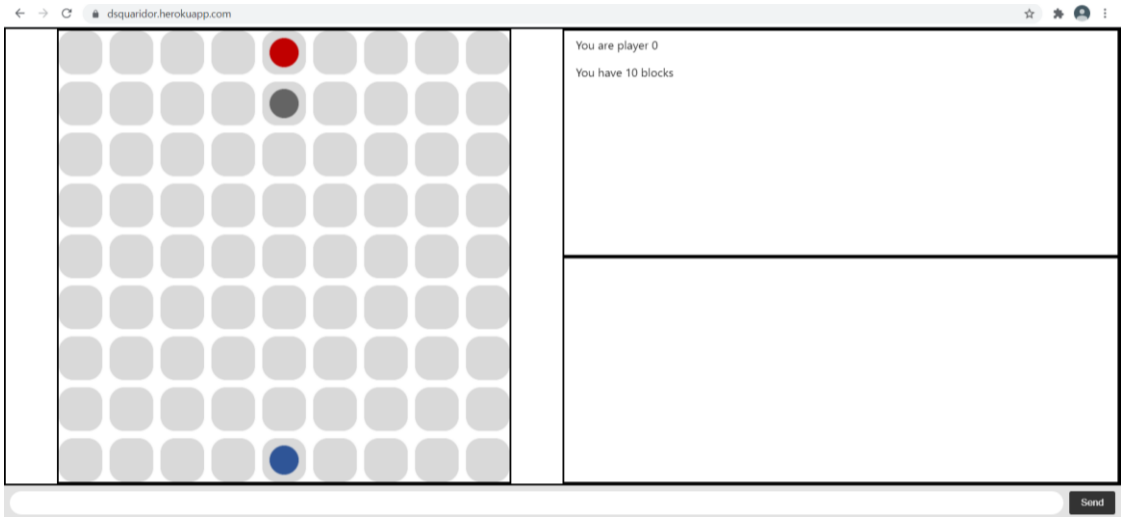
플레이어들은 socket.io의 socket.emit() method를 이용하여 자신이 말을 어떻게 움직일지 서버로 송신한다. 또, 상대방이 말을 어떻게 움직였는지 확인하기 위해 socket.on() method를 사용하였다.

5. 실행 화면

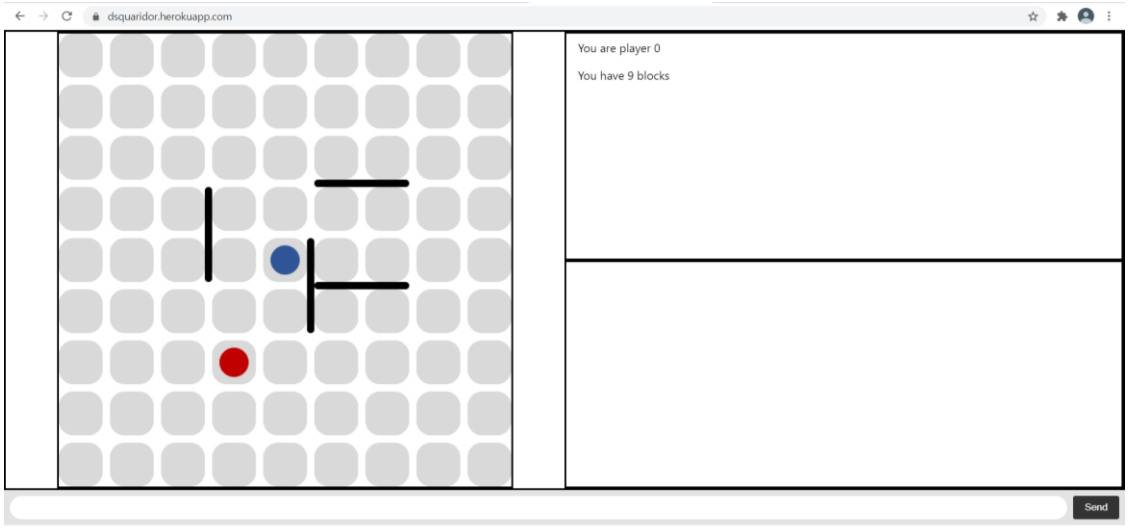
웹사이트에 접속하면 다음 화면을 볼 수 있다.



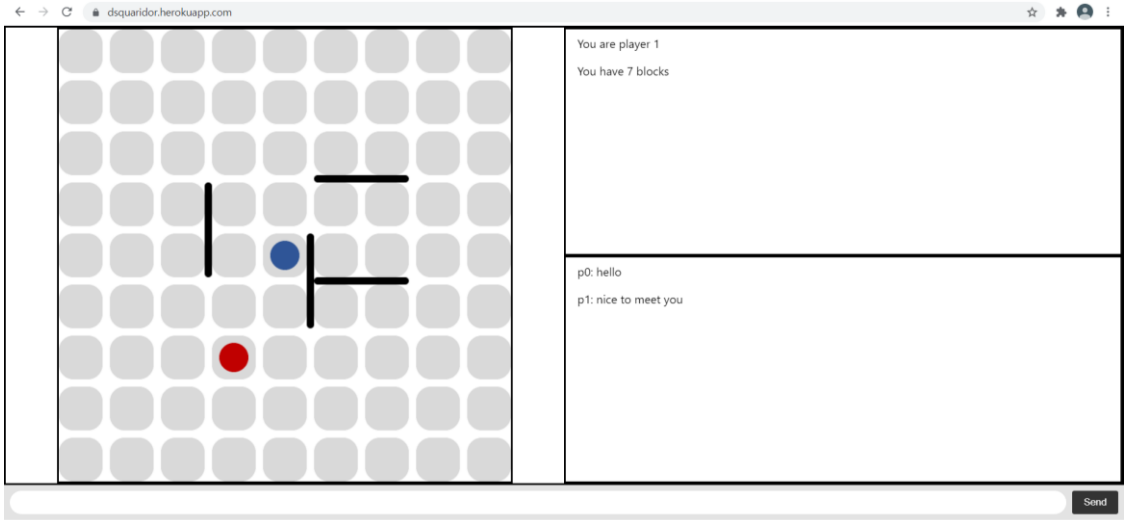
말을 옮기고자 하는 위치에 커서를 올리면 옅은 색깔로 그 칸이 색칠되면서 미리보기가 제공된다. 만약 그 칸이 옮길 수 없는 칸이거나 자신의 차례가 아닐 때에는 미리보기가 제공되지 않는다.



상대방과 번갈아가며 차례를 이어나간다. 말을 옮기거나 블록을 설치할 수 있다.



채팅을 통해 상대방과 소통할 수 있다.



게임이 종료되면 다음과 같은 화면이 나타난다. Regame 버튼을 누르면 게임을 다시 시작할 수 있다.

6. Reference

- https://expressjs.com/ko/
- https://socket.io/
- https://nodejs.org/ko/
- https://www.heroku.com/
- https://en.wikipedia.org/wiki/Quoridor