OOP-Term Project

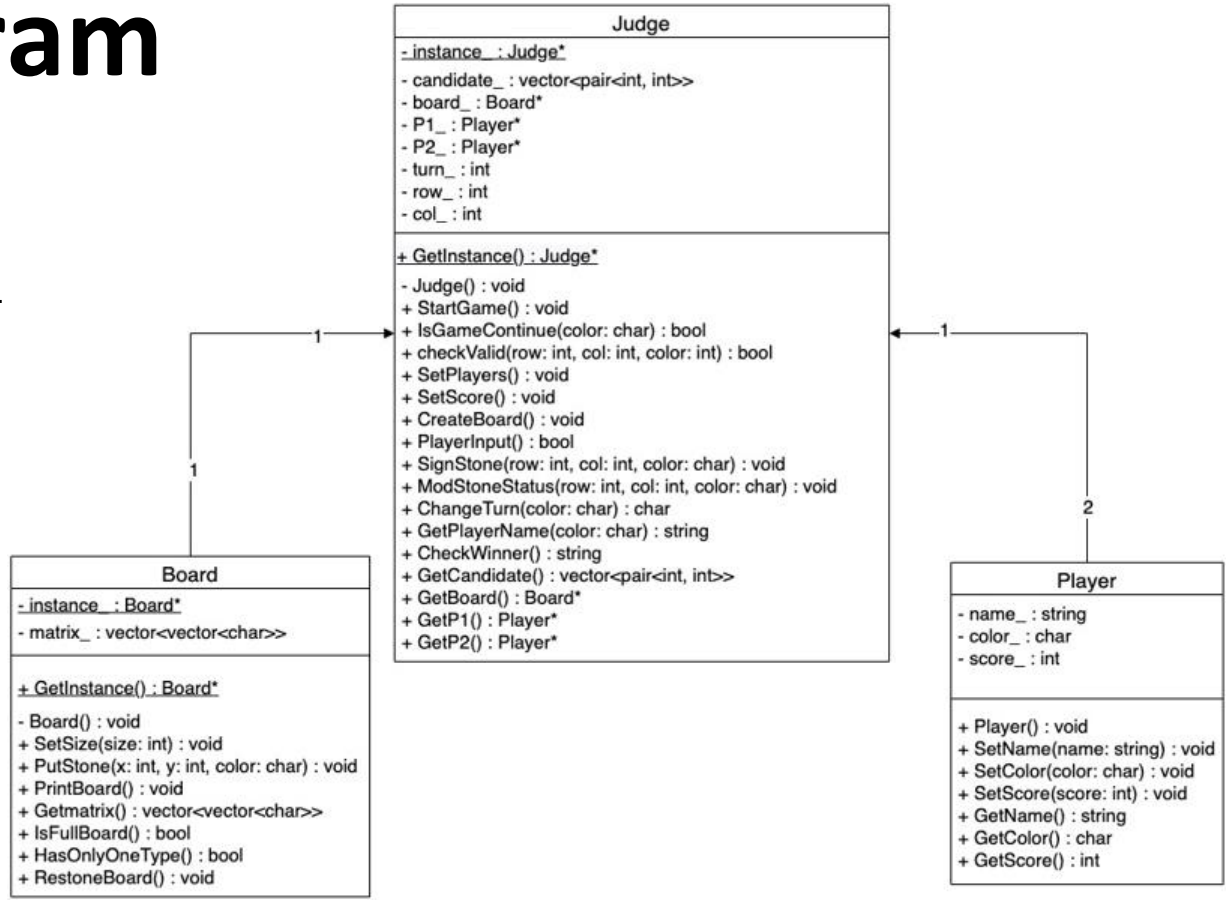
201802155 정주헌
201802167 최원준
202002569 최동현

목차

- Class 소개
- 구현 / 미구현 기능
- 예외 상황 및 처리 방법
- 사용한 디자인 패턴
- 시나리오 진행

Class Diagram

- 총 3개의 클래스로 구성
- **Judge**: 게임의 심판(Judge) 역할을 수행하는 클래스
- **Board**: 게임 판(Board)을 구성하는 클래스
- **Player**: 게임 플레이어의 정보를 담는 클래스
- Judge 객체가 Board, Player 객체를 각각 받아서 게임을 진행



Class Board

matrix_ : 보드판을 표현하기 위한 벡터

- **static Board* GetInstance()**: Board의 인스턴스 반환
- **Board()**: 생성자, 기본 필드값 설정
- **void SetSize(int size)**: 보드판의 size 설정
- **void PutStone(int x, int y, char color)**: 입력 받은 좌표의 상태를 바꾸기
- **void PrintBoard()**: 보드판을 출력
- **std::vector<std::vector<char> > Getmatrix()**: 보드판을 출력
- **bool IsFullBoard()**: 보드판이 가득 차있는지 여부 확인
- **bool HasOnlyOneType()**: 보드판에 한가지 색만 있는지 확인
- **void RestoreBoard()**: 턴 변경 시 '*'를 '.'로 변환

※ **Getmatrix()** 와 **PrintBoard()**의 차이는 return type
전자는 보드판을 벡터 형태로 반환하고, 후자는 보드판을 출력
※ **IsFullBoard()**, **HasOnlyOneType()**는 Judge에서 게임 종료 판별 시 사용

Class Player

name_, color_, score_: 플레이어 이름, 플레이어의 돌 색깔(B or W), 플레이어의 점수

- **Player()**: 생성자. 기본 필드값 설정
- **void SetName(std::string name)**: 플레이어 이름 설정
- **void SetColor(char color)**: 플레이어의 돌 색깔 설정
- **void SetScore(int score)**: 플레이어의 점수 설정
- **std::string GetName()**: 이름 반환
- **char GetColor()**: 돌 색깔 반환
- **int GetScore()**: 점수 반환

Class Judge

candidate_: 돌을 놓을 수 있는 위치 저장할 벡터
Board 객체 하나, **Player** 객체 둘 포함

- **static Judge* GetInstance()**: Judge의 인스턴스를 반환
- **Judge()**: 생성자, 기본 필드값 설정
- **void StartGame()**: 생성된 인스턴스들을 가지고 게임을 시작하는 함수
- **bool IsGameContinue(char color)**: 파라미터로 받은 색의 게임 진행이 가능한지 여부 반환
- **bool CheckValid(int row, int col, char color)**: 돌을 놓을수 있는지 여부 판단
- **void SetPlayers()**: player들의 정보를 입력받아 저장
- **void SetScore()**: 각 플레이어들의 점수를 계산
- **void CreateBoard()**: 시작할때 보드를 생성
- **bool PlayerInput()**: 플레이어의 입력값을 받음, 유효한 입력인지 확인하기 위해 bool 타입 반환

Class Judge

- **void SignStone(int row, int col, char color)**: board에 돌을 놓는 신호 전송
- **void ModStoneStatus(int row, int col, char color)**: 플레이어의 입력에 따라 돌을 뒤집음
- **char ChangeTurn(char color)**: 턴 변경
- **std::string GetPlayerName(char color)**: Player* 의 이름을 가져옴
- **std::string CheckWinner()**: Player* -> GetScore() 함수로 점수를 받아 승패를 결정
- **std::vector<std::pair<int, int> > GetCandidate()**: candidate 배열 반환
- **Board* GetBoard()**: board 반환
- **Player* GetP1()**: 플레이어1의 인스턴스 반환
- **Player* GetP2()**: 플레이어2의 인스턴스 반환

구현/미구현 기능

- 기본적인 오델로 게임 규칙에 의한 기능은 대부분 구현
- 돌 색깔 표시(B or W), 턴 넘기기, 돌을 놓을 수 있는 위치 표시, 돌 뒤집기 , 점수 표시, 승자/패자 및 무승부 판별 등등 구현
- 사용자 입력을 통해 게임 보드의 크기 설정 구현
- 다음 규칙에 의거한 게임 종료 구현
  - 아래와 같은 조건에 의해 양쪽 모두 더 이상 돌을 놓을 수 없게 되면 게임이 끝나게 된다.
    - 64개의 돌 모두가 판에 가득 찬 경우 (가장 일반적)
    - 어느 한 쪽이 돌을 모두 뒤집은 경우
    - 한 차례에 양 쪽 모두 서로 차례를 넘겨야 하는 경우

  (출처: Wikipedia)
- UI를 구현해보려 했으나 실패 → 터미널에서 실행

예외 상황 및 처리 방법

1. 보드 사이즈 입력 예외 처리
2. 돌 색깔 입력 예외 처리
3. 돌을 놓을 수 없는 위치에 놓았을 경우 예외 처리
4. 게임을 진행하다 돌을 놓을 수 없는 경우 예외 처리

예외 상황 및 처리 방법

1. 보드 사이즈 입력시 예외 처리

- 보드 사이즈 입력을 4 이상 짝수로 제한
- **Judge::CreateBoard()** 함수를 이용해 보드 입력이 홀수이거나 4 미만이면 에러 메시지를 출력하고, 다시 입력받는 방식으로 예외 처리

```
Welcome to Othello

Enter the size of board : 1
Enter an even number of 4 or more : 2
Enter an even number of 4 or more : 3
Enter an even number of 4 or more : 5
Enter an even number of 4 or more : 7
Enter an even number of 4 or more : 101
Enter an even number of 4 or more : 4
Enter Player1's name :
```

```
229  /* CreateBoard : board의 크기를 입력받는 함수 */
230  void Judge::CreateBoard() {
231    std::string input = "";
232    int size = 0;
233
234    std::cout << "Enter the size of board : ";
235    std::cin >> input;
236    size = atoi(input.c_str());
237    while ((size % 2 == 1) || (size < 4)) {
238      std::cout << "Enter an even number of 4 or more : ";
239      std::cin >> input;
240      size = atoi(input.c_str());
241    }
242    (this->board_)->SetSize(size);
243  }
```

예외 상황 및 처리 방법

2. 돌 색깔 입력 예외 처리 (1)

- **Judge::SetPlayers()** 함수를 이용해 입력이 'B' 또는 'W'가 아니면 에러메시지를 출력하고 다시 입력 받음

```
Enter Player1's name : qq
Player1 can choose color (B or W) : E
You have to choose either B or W.
Player1 can choose color (B or W) : be
You have to choose either B or W.
Player1 can choose color (B or W) : b
You have to choose either B or W.
Player1 can choose color (B or W) : w
You have to choose either B or W.
Player1 can choose color (B or W) : B
Enter Player2's name :
```

```
173  /* SetPlayers : 게임 시작 시 초기 플레이어의 이름, 돌 색깔을 설정해주는 함수 */
174  void Judge::SetPlayers() {
175    std::string tmp = "";
176    char p1_color;
177
178    std::cout << "Enter Player1's name : ";
179    std::cin >> tmp;
180    p1_->SetName(tmp);
181    tmp = "";
182    while (true) {
183      std::cout << "Player1 can choose color (B or W) : ";
184      std::cin >> tmp;
185      if ((tmp.front() == 'B') || (tmp.front() == 'W')) {
186        break;
187      } else {
188        std::cout << "You have to choose either B or W." << std::endl;
189      }
190    }
```

예외 상황 및 처리 방법

2. 돌 색깔 입력 예외 처리 (2)

- 플레이어1의 돌 색깔이 입력되면 플레이어 2의 돌 색깔은 자동적으로 반대의 색이 됨

```
194    std::cout<< "Enter Player2's name : ";
195    std::cin >> tmp;
196    p2_->SetName(tmp);
197    if (p1_color == 'B') {
198      p2_->SetColor('W');
199      std::cout << "Player2's color is W." << std::endl;
200    } else {
201      p2_->SetColor('B');
202      std::cout << "Player2's color is B." << std::endl;
203    }
204  }
```

```
Enter Player1's name : qq
Player1 can choose color (B or W) : E
You have to choose either B or W.
Player1 can choose color (B or W) : be
You have to choose either B or W.
Player1 can choose color (B or W) : b
You have to choose either B or W.
Player1 can choose color (B or W) : w
You have to choose either B or W.
Player1 can choose color (B or W) : B
Enter Player2's name : ww
Player2's color is W.

Possible Coordinates (*)
| 0  2 | 1  3 | 2  0 | 3  1 |

\  0  1  2  3
0  .  .  *  .
1  .  B  W  *
2  *  W  B  .
3  .  *  .  .

qq(B) 2 : 2 ww(W)
Player qq's Turn
Enter row and column ex) 2 4 :
```

예외 상황 및 처리 방법

3. 돌을 놓을 수 없는 위치에 놓았을 경우 예외 처리 (1)

- **Judge::PlayerInput()** 함수를 이용해 정상적인 위치인지 확인 후 정상적이지 않은 위치면 false를, 정상적인 위치면 true를 반환
- **Judge::StartGame()** 함수 안에 있는 while 문의 반복 조건으로 사용해 정상적인 값이 들어올 때까지 위 함수를 실행

```
244  /* PlayerInput : 플레이어의 입력값을 받는 함수 */
245  bool Judge::PlayerInput() {
246    std::string input_row = "";
247    std::string input_col = "";
248    std::cout << "Enter row and column ex) 2 4 : ";
249    std::cin >> input_row >> input_col;
250
251    this->row_ = atoi(input_row.c_str());
252    this->col_ = atoi(input_col.c_str());
253    std::vector<std::vector<char> > matrix = board_->Getmatrix();
254    int size = matrix.size();
255
256    if ((this->row_ >= 0) && (this->row_ < size) && (this->col_ >= 0)
257    && (this->col_ < size) && (matrix[this->row_][this->col_] == '*')) {
258      return true;
259    }
260    return false;
261  }
```

```
while (!(this->PlayerInput())) {
  // 후보군이 아닌 다른 값들을 받을 경우 예외처리
  std::cout << "It's not a right choice." << std::endl;
}
```

예외 상황 및 처리 방법

3. 돌을 놓을 수 없는 위치에 놓았을 경우 예외 처리 (2)

주어진 * 자리에 놓지 않았을 경우

```
Possible Coordinates (*)
| 0  2 | 1  3 | 2  0 | 3  1 |

\  0  1  2  3
0  .  .  *  .
1  .  B  W  *
2  *  W  B  .
3  .  *  .  .

qq(B) 2 : 2 ww(W)
Player qq's Turn
Enter row and column ex) 2 4 : 0 0
It's not a right choice.
Enter row and column ex) 2 4 : 1 3

Possible Coordinates (*)
| 0  1 | 0  3 | 2  3 |
```

주어진 보드 밖의 위치에 놓았을 경우

```
Possible Coordinates (*)
| 0  1 | 0  3 | 2  3 |

\  0  1  2  3
0  .  *  .  *
1  .  B  B  B
2  .  W  B  *
3  .  .  .  .

qq(B) 4 : 1 ww(W)
Player ww's Turn
Enter row and column ex) 2 4 : 4 4
It's not a right choice.
Enter row and column ex) 2 4 : 0 1

Possible Coordinates (*)
| 0  0 | 1  0 | 2  0 | 3  0 |
```

예외 상황 및 처리 방법

4. 게임을 진행하다 돌을 놓을 수 없는 경우 예외 처리 (1)

- 아래와 같은 조건에 의해 양쪽 모두 더 이상 돌을 놓을 수 없게 되면 게임이 끝나게 된다.
  - 64개의 돌 모두가 판에 가득 찬 경우 (가장 일반적)
  - 어느 한 쪽이 돌을 모두 뒤집은 경우
  - 한 차례에 양 쪽 모두 서로 차례를 넘겨야 하는 경우

(출처: Wikipedia)

- 게임을 진행하다 보면 위 두 경우에 의해 게임이 중간에 끝날 수 있음
- 어느 한 쪽이 돌을 모두 뒤집은 경우는 **Judge::HasOnlyOneType()** 함수로 예외 처리
- 한 차례에 양 쪽 모두 서로 차례를 넘겨야 하는 경우는 **candidate_** 벡터의 크기가 0일 때 즉, 놓을 수 있는 위치가 하나도 없을 때 예외 처리

예외 상황 및 처리 방법

4. 게임을 진행하다 돌을 놓을 수 없는 경우 예외 처리 (2)

- 어느 한 쪽이 돌을 모두 뒤집은 경우

```
85  /* IsGameContinue : 게임을 진행할 수 있는지 여부를 확인하는 함수 */
86  bool Judge::IsGameContinue(char color) {
87    // 한 차례에 양쪽 모두 서로 차례를 넘겨야 하는 경우
88    std::vector <std::vector<char>> matrix = board_->Getmatrix();
89    int size = matrix.size();
90
91    if (board_->IsFullBoard()) { return false; }
92    // 돌을 놓을 수 있는 공간이 없을 경우
93    if (board_->HasOnlyOneType()) { return false; }
94    // 한가지 색의 돌만 존재하는 경우
```

```
test1(B) 9 : 3 test2(W)
Player test1's Turn
Enter row and column ex) 2 4 : 3 5

Possible Coordinates (*)
|

\  0  1  2  3  4  5
0  .  .  .  .  .  .
1  .  B  B  B  .  .
2  .  B  B  B  .  .
3  .  B  B  B  B  B
4  .  B  .  .  .  .
5  .  B  .  .  .  .

Oops! Player test2 has no space to put stone.
It's Player test1's turn again.

Possible Coordinates (*)
|


test1(B) 13 : 0 test2(W)
Congratulations! Player test1 wins!
wanpile@MacBookPro Othello %
```

예외 상황 및 처리 방법

4. 게임을 진행하다 돌을 놓을 수 없는 경우 예외 처리 (3)

- 한 차례에 양 쪽 모두 서로 차례를 넘겨야 하는 경우

```
testqq(W) 7 : 6 testw2(B)
Player testqq's Turn
Enter row and column ex) 2 4 : 2 0

Possible Coordinates (*)
|

\  0  1  2  3
0  W  W  W  W
1  W  W  W  W
2  W  W  W  W
3  B  .  .  B

Oops! Player testw2 has no space to put stone.
It's Player testqq's turn again.

Possible Coordinates (*)
|


testqq(W) 12 : 2 testw2(B)
Congratulations! Player testqq wins!
```

```cpp
/* StartGame : 게임 시작 및 진행을 하는 함수.
 main에서 직접적인 게임 진행을 해준다. */
void Judge::StartGame() {
  std::cout << "Welcome to Othello\n" << std::endl;
  this->CreateBoard();
  this->SetPlayers();
  std::vector<std::vector<char>> matrix = board_->Getmatrix();
  int size = matrix.size();
  char turn_flag = 'B';
  // black 돌이 먼저 시작하므로 초기값으로 'B' 설정
  while (true) {
    std::cout << std::endl;

    if (!(this->IsGameContinue(turn_flag))) {
      // 게임이 진행될 수 있는지 판단하는 부분
      break;
    }
    if (this->candidate_.size() == 0) {
      // 후보군이 없으므로 턴이 넘어가야함.
      board_->PrintBoard();
      std::cout << "\nOops! Player " << this->GetPlayerName(turn_flag)
      << " has no space to put stone." << std::endl;

      turn_flag = this->ChangeTurn(turn_flag);
      std::cout << "It's Player " << this->GetPlayerName(turn_flag)
      << "'s turn again." << std::endl;
      this->SetScore();
      continue;
    }
```

디자인 패턴

- **class Judge, class Board**에 각각 **싱글톤 패턴** 적용

↓

- 두개로 각각 호출될 수 있는 Judge와 Board를 하나씩만 호출함으로써 메모리 낭비(Memory Leak)를 방지
- 하나의 Judge, Board 인스턴스만을 사용하였기에 **p1_, p2_** 간의 데이터 공유를 쉽게 함

```
9  Judge* Judge::instance_ = nullptr;
10 /* GetInstance : Judge의 sigleton pattern 적용을 위한 함수 */
11 Judge* Judge::GetInstance() {
12   if (instance_ == nullptr) { instance_ = new Judge(); }
13   return instance_;
14 }
15 Judge::Judge() {
16   this->row_ = 0;
17   this->col_ = 0;
18   this->turn_ = 0;
19   Player* p1 = new Player();
20   Player* p2 = new Player();
21   this->p1_ = p1;
22   this->p2_ = p2;
23   this->board_ = Board::GetInstance();
24 }
```

```
4  Board* Board::instance_ = nullptr;
5
6  /*GetInstance : Singleton pattern을 적용한 함수*/
7  Board* Board::GetInstance() {
8    if (instance_ == nullptr) { instance_ = new Board(); }
9    return instance_;
10 }
11
12 Board::Board() {}
```

시나리오 진행

1. 프로그램 실행 및 보드 사이즈 입력
2. Player1, Player2의 이름 및 색깔을 각각 입력
3. 돌을 놓을 좌표를 입력
4. 턴을 번갈아 가면서 돌을 놓기
5. 게임이 끝나면 승자가 결정됨

시나리오 진행

1. 프로그램 실행 및 보드 사이즈 입력

```
yorker@DESKTOP-1GS4JVK:/mnt/c/Users/user/OODLab/Othello-final$ ./main
Welcome to Othello

Enter the size of board : 8
```

2. Player1, Player2의 이름 및 색깔을 각각 입력

```
Enter the size of board : 8
Enter Player1's name : JuHeon
Player1 can choose color (B or W) : B
Enter Player2's name : WonJoon
```

시나리오 진행

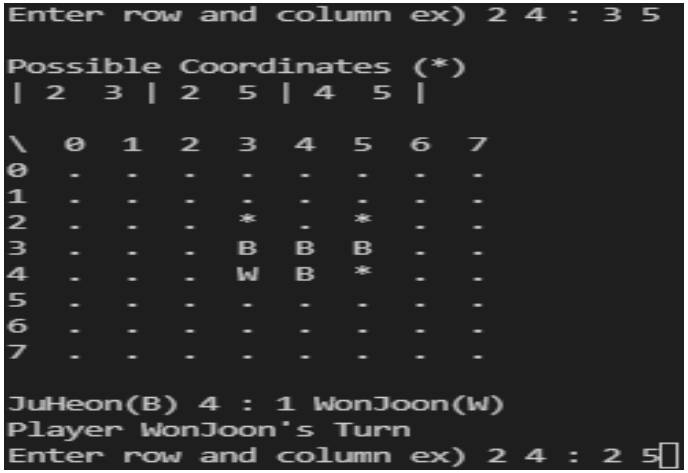
3. 돌을 놓을 좌표를 입력

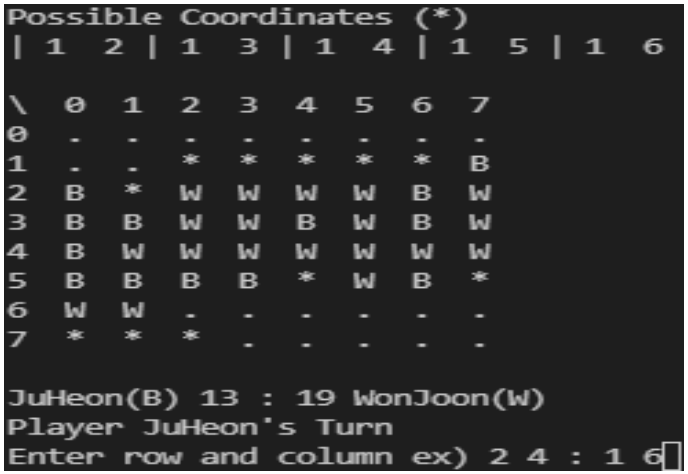
- **2**까지 완료하면 돌을 놓을 수 있는 위치(**Possible Coordinates**), 보드 판의 상태, 현재 점수를 알려주고 어떤 플레이어의 차례인지 안내함
- 해당 플레이어는 자신의 돌을 놓을 위치를 **Possible Coordinates**에서 선택해 입력

시나리오 진행

4. 턴을 번갈아 가면서 돌을 놓기

- **3**까지 완료하면 돌을 놓을 수 있는 위치(**Possible Coordinates**), 보드 판의 상태, 현재 점수를 똑같이 알려주고 플레이어의 턴이 바뀌었음을 안내함
- 해당 플레이어는 자신의 돌을 놓으면서 게임을 진행
- 이 일련의 과정을 쭉 반복





시나리오 진행

5. 게임이 끝나면 승자가 결정됨

- 앞 과정들을 진행하고 게임이 끝나면 플레이어의 최종 점수를 토대로 승패를 결정
- 승자의 이름과 함께 축하 메시지를 출력하며 게임 종료

```
Possible Coordinates (*)
| 0  3 |

\  0  1  2  3  4  5  6  7
0  B  B  B  *  B  B  B  W
1  B  B  W  B  B  B  W  W
2  B  W  B  W  B  W  W  W
3  B  W  B  B  W  B  B  W
4  B  W  W  B  W  W  B  W
5  B  W  B  W  B  W  B  W
6  B  B  B  B  W  B  W  W
7  B  B  B  B  B  B  B  W

JuHeon(B) 38 : 25 WonJoon(W)
Player WonJoon's Turn
Enter row and column ex) 2 4 : 0 3


JuHeon(B) 33 : 31 WonJoon(W)
Congratulations! Player JuHeon wins!
```