bob7dev #0

**Othello Protocol design**

이세린

**Protocol Structure**

|  |  |  |  |
| --- | --- | --- | --- |
| version | state | type | payload(if exist) |

오델로 프로토콜은 다음과 같은 네 부분으로 구성되어 있다.

**(1) version**

-version 은 프로토콜의 생성 id와 같은 역할을 한다. 하나의 프로토콜이 생성되고, 그것이 최적화의 문제나 버그 등과같은 이유로 규격에 있어서 수정이 이루어지면, 클라이언트 역시 수정된 버전의 프로토콜을 사용하도록 해야한다.

-거기엔 두가지 방법이 있는데, 클라이언트의 버전을 체크하여 클라이언트 버전을 업데이트 시키게 하거나 또는 클라이언트로부터 받는 패킷에서 프로토콜 버전을 체크하여 업데이트를 하게 할 수 있다.

-두가지 방법 모두 사용될 수 있으나, 서버가 중지되지 않은 상태에서 여러 버전의 프로토콜을 호환시킬 수 있는 방법으로 버전 필드를 설정할 수 있을 것이라는 생각에서 버전을 도입해 보았다.

-또한 strong heuristic의 장치로써 기능할 수 있을 것으로 기대된다.

**(2) state**

클라이언트의 상태를 나타낼 수 있는 필드이다.

- Handshake(0)

Handshake 상태는 TCP를 사용하는 클라이언트와 서버 간 통신에서, TCP 3-way handshaking 으로 연결이 수립 된 이후에 가상의 또다른 연결을 수립할 수 있도록 한다. 이 가상의 연결은 서버가 허용할 수 있는 클라이언트의 수를 미리 지정해놓고, 그것을 넘기는 클라이언트가 접속을 요청할 때, SERVER\_FULL과 같은 응답으로 클라이언트에게 연결할 수 없음을 알려줄 수 있다.

또한 이 통신을 통해서 연결된 클라이언트에게는 player id를 부여하여, 고유 식별자로서 사용할 수 있도록 한다.

- Match(1)

Match 상태는 클라이언트가 서버와 성공적으로 연결을 했지만, 게임이 시작되지는 않은 상태이다. 이 상태에서 클라이언트는 자신의 player id로서 서버에게 게임을 시작 할 준비가 되었음을 알리고, 서버는 대기중인 다른 player가 있으면 매칭시킨 후 게임이 시작되었음을 알릴 수 있다.

- Play(2)

Play 상태는 게임이 진행중인 상태이다. 이 상태에서는 다양한 command를 서버와 주고받으며 서버의 중개 하에 게임이 진행된다. 게임에서는 다양한 상황이 발생할 수 있고, Play 상태는 게임이 어떠한 이유에서든 종료될 때 까지 유지 될 것이다.

**(3) messages (state + type + payload)**

- Handshake

HELLO\_SERVER(0) , no payload

:TCP연결 수립 후 클라이언트가 서버에게 실질적 연결을 요청하는 메시지로서, payload가 요구되지 않는다.

HELLO\_CLIENT(1)

:클라이언트로부터 HELLO\_SERVER를 받은 서버가 허용할 수 있는 클라이언트 범위 내에서 응답으로서 보내는 메시지. 이 연결은 실질적으로 클라이언트가 게임을 할 수 있는 상태이므로 player\_id를 랜덤으로 부여해 식별자로서 저장해 놓는다.

SERVER\_FULL(2) , no payload

:클라이언트로부터 HELLO\_SERVER를 받았지만, 서버가 더이상 클라이언트를 받아들일 수 없을 때 연결 불가의 메시지로서 전송한다.

-Match

REQUEST(3)

:클라이언트가 서버에 성공적으로 연결된 후, 대기중이다가 게임 플레이를 하고 싶을 때 서버로부터 다른 플레이어와 매칭하기를 요청하는 메시지. 자신이 어떤 플레이어인지 알 수 있도록 player\_id를 payload로 전송한다.

RESPONSE(4)

:매칭 대기중인 클라이언트가 있을 때, 그 중 두 플레이어를 연결한다. 이 때, 클라이언트들은 서로에 대한 정보를 가지고 있어야 하므로, 두 플레이어의 player\_id와 검은돌, 흰돌(게임 순서)을 지정해 payload로 전송한다.

-PLAY

TURN(5)

:서버가 두 클라이언트 모두에게 보내는 패킷으로, 어떠한 player가 수를 둘 차례인지 알려준다. 오델로 게임 특성 상, turn이 불규칙적일 수 있으므로, 서버가 중앙에서 통제하여 알린다. 따라서 해당 턴의 주인인 클라이언트의 player\_id를 payload로 전송한다.

MOVE(6)

:클라이언트가 서버에게 단일로 전송하는 패킷. 각 턴의 주인인 플레이어가 자신이 둔 수의 위치를 서버에 알린다. 자신의 player\_id와 수의 좌표(x,y)를 payload로 전송한다.

ACCEPT(7)

:서버가 클라이언트 모두에게 전송하는 패킷. 이 패킷은 각 턴에 대한 클라이언트의 선택이 서버의 검증을 통과하여 실제로 두 클라이언트 모두에게 반영되는 패킷. 이 패킷은 player\_id와 그 수의 좌표(x,y)를 전송하여, 그 턴의 주인이 아닌 상대 플레이어도 어디에 수를 놓았는지 알 수 있도록 한다.

REJECT(8)

:서버가 클라이언트에게 단일로 전송하는 패킷. 각 턴의 주인이 수를 두었지만 그 값이 불가능한 경우(고의 조작 또는 알 수 없는 어떠한 문제가 발생하여)해당 턴을 거부한다. 이러한 일이 발생할 수 있으므로 플레이어는 자기가 수를 놓았어도 ACCEPT 메시지가 올 때까지 클라이언트에 반영할 수 없고, 타이머는 지속되어야 한다.

GIVE\_UP(9)

:클라이언트가 서버에게 단일로 전송할 수 있는 패킷. 이 패킷은 클라이언트가 게임을 포기하고 싶을 때 요청하며 자신의 player\_id를 payload로 전송해야 한다. 이 패킷을 전송받은 서버는 END\_GAME 패킷으로 각 클라이언트에게 종료를 알린다.

END\_GAME(10)

:서버가 클라이언트 모두에게 전송할 수 있는 패킷. 이 패킷은 게임이 종료되었을 때 우승한 플레이어의 player\_id를 payload로 전송하며, 만약 게임의 승자가 없을 경우 0값을 전송한다.