Guandan Platform Documentation

1. Project Deployment

1.1 Java Version Deployment

Requirements: JDK 17 or higher

Single-threaded Mode (Default):

```
java -jar target/guandan-java-1.0.0.jar
```

Multi-threaded Mode:

```
java -Dguandan.cluster.mode=true -Dguandan.cluster.workers=4 -jar target/guandan-java-1.0.0.jar
```

Specifying a Port:

```
java -Dserver.port=8182 -jar target/guandan-java-1.0.0.jar
```

1.2 Native Executables

- **Windows**: Double-click to run `guandan.exe`
- **Linux**: Execute `./guandan-linux` (requires execution permissions: `chmod +x guandan-linux`)

Port Usage:

- Single-process mode: 3000 (HTTP), 8181 (WebSocket)
- Cluster mode: 8181 (load balancer) + 8182, 8183, 8184, etc. (worker processes)

---

2. WebSocket Connection

Connection Address: `ws://127.0.0.1:8181`

Data Format: JSON (UTF-8 encoding)

Basic Message Structure:

```
{
    "type": "MESSAGE_TYPE",
    "data": {
        // Message data
    }
}
```

---

3. Client-to-Server Messages

3.1 Create Room

Message Type: `CREATE_ROOM`

Message Format:

```
{
    "type": "CREATE_ROOM",
    "data": {
        "userId": "user1",
        "round": 1,
        "seatNum": 0
    }
}
```

Parameters:

- `userId`: User ID (string, must be unique for each player)
- `round`: Number of game rounds (integer)
- `seatNum`: Seat number (optional, 0-3, auto-assigned if not specified)

Response Format:

```
{
    "type": "CREATE_ROOM",
    "code": 200,
    "data": {
        "roomId": 1,
        "userNum": 0
    }
}
```

3.2 Join Room

Message Type: `JOIN_ROOM`

Message Format:

```
{
    "type": "JOIN_ROOM",
    "data": {
        "userId": "user2",
        "roomId": 1,
        "seatNum": 1
    }
}
```

Parameters:

- `userId`: User ID (string)
- `roomId`: Room ID (integer, returned when creating a room)
- `seatNum`: Seat number (optional, 0-3)

Response Format:

```
{
    "type": "JOIN_ROOM",
    "code": 200,
    "data": {
        "roomId": 1,
        "userNum": 1
    }
}
```

Note: The game starts automatically when the room reaches 4 players.

3.3 Play Action

Message Type: `PLAY`

Message Format:

```
{
    "type": "PLAY",
    "data": {
        "roomId": 1,
        "player": 0,
        "act": ["PASS", "PASS", "PASS"]
    }
}
```

Parameters:

- `roomId`: Room ID (integer)
- `player`: Player seat number (integer, 0-3)
- `act`: Action tuple `[Pattern, Rank, CardList]`, must be selected from the `actionList` provided by the server

3.4 Tribute Action

Message Type: `TRIBUTE`

Message Format:

```
{
    "type": "TRIBUTE",
    "data": {
        "roomId": 1,
        "player": 0,
        "act": ["tribute", "tribute", ["D2"]]
    }
}
```

Parameters:

- `roomId`: Room ID (integer)
- `player`: Player seat number (integer)
- `act`: Tribute action, must be selected from the `actionList` provided by the server

3.5 Return Tribute Action

Message Type: `PAYTRIBUTE`

Message Format:

```
{
    "type": "PAYTRIBUTE",
    "data": {
        "roomId": 1,
        "player": 0,
        "tributePos": 3,
        "tribute": "S2",
        "act": ["back", "back", ["H2"]]
    }
}
```

Parameters:

- `roomId`: Room ID (integer)
- `player`: Player seat number (integer)
- `tributePos`: Seat number of the player who paid tribute (integer, obtained from server message)
- `tribute`: The card offered as tribute (string, obtained from server message)
- `act`: Return tribute action, must be selected from the `actionList` provided by the server

Note: `tributePos` and `tribute` must exactly match the values received in the request message.

---

4. Server-to-Client Messages

4.1 Notification Messages (notify)

Notification messages are broadcast to all players.

4.1.1 Game Start

Identifier: `"type": "notify", "stage": "beginning"`

Message Format:

```
{
    "type": "notify",
    "stage": "beginning",
    "handCards": ["S2", "H2", "C2", ...],
    "myPos": 1
}
```

Fields:

- `handCards`: Hand cards list (string array)
- `myPos`: Player seat number (integer, 0-3)

4.1.2 Play Notification

Identifier: `"type": "notify", "stage": "play"`

Message Format:

```
{
    "type": "notify",
    "stage": "play",
    "curPos": 1,
    "curAction": ["Single", "2", ["S2"]],
    "greaterPos": 1,
    "greaterAction": ["Single", "2", ["S2"]]
}
```

Fields:

- `curPos`: Current player's seat number (integer)
- `curAction`: Current player's action (tuple)
- `greaterPos`: Seat number of player with the highest action (integer, -1 when all players PASS)
- `greaterAction`: Highest action (tuple, null when all players PASS)

4.1.3 Tribute Notification

Identifier: `"type": "notify", "stage": "tribute"`

Message Format:

```
{
    "type": "notify",
    "stage": "tribute",
    "result": [[0, 3, "S2"]]
}
```

Fields:

- `result`: Tribute result list (2D array), format: `[Giver Seat, Receiver Seat, Tribute Card]`

4.1.4 Anti-Tribute Notification

Identifier: `"type": "notify", "stage": "anti-tribute"`

Message Format:

```
{
    "type": "notify",
    "stage": "anti-tribute",
    "antiNums": 2,
    "antiPos": [0, 2]
}
```

Fields:

- `antiNums`: Number of players performing anti-tribute (integer)
- `antiPos`: List of anti-tribute player seat numbers (integer array)

4.1.5 Return Tribute Notification

Identifier: `"type": "notify", "stage": "back"`

Message Format:

```
{
    "type": "notify",
    "stage": "back",
    "result": [[3, 0, "S2"]]
}
```

Fields:

- `result`: Return tribute result list (2D array), format: `[Returner Seat, Receiver Seat, Return Card]`

4.1.6 Episode Over

Identifier: `"type": "notify", "stage": "episodeOver"`

Message Format:

```
{
    "type": "notify",
    "stage": "episodeOver",
    "order": [0, 1, 2, 3],
    "curRank": "A",
    "restCards": [[3, ["C2"]]]
}
```

Fields:

- `order`: Finishing order (integer array)
- `curRank`: Current rank (string)
- `restCards`: Remaining cards list (2D array), format: `[Seat Number, Card List]`

4.1.7 Game Over

Identifier: `"type": "notify", "stage": "gameOver"`

Message Format:

```
{
    "type": "notify",
    "stage": "gameOver",
    "curTimes": 1,
    "settingTimes": 1
}
```

Fields:

- `curTimes`: Current number of completed games (integer)
- `settingTimes`: Configured number of games (integer)

4.1.8 Game Result

Identifier: `"type": "notify", "stage": "gameResult"`

Message Format:

```
{
    "type": "notify",
    "stage": "gameResult",
    "victory": 0,
    "victoryRank": ["A", "K"]
}
```

Fields:

- `victory`: Winning team (integer, 0 for seats 0 and 2, 1 for seats 1 and 3)
- `victoryRank`: Final ranks of both teams (string array)

4.2 Action Request Messages (act)

Action request messages are targeted at the specific player who needs to perform an action.

4.2.1 Play Request

Identifier: `"type": "act", "stage": "play"`

Message Format:

```
{
    "type": "act",
    "handCards": ["S2", "H2", ...],
    "publicInfo": [
        {"rest": 22},
        {"rest": 23},
        {"rest": 23},
        {"rest": 27}
    ],
    "selfRank": "K",
    "oppoRank": "9",
    "curRank": "K",
    "stage": "play",
    "curPos": 2,
    "curAction": ["Bomb", "A", ["HA", "HA", "CA", "DA"]],
    "greaterAction": ["Bomb", "A", ["HA", "HA", "CA", "DA"]],
    "greaterPos": 2,
    "actionList": [
        ["PASS", "PASS", "PASS"],
        ["Bomb", "9", ["H9", "H9", "C9", "D9"]],
        ...
    ],
    "indexRange": 21
}
```

Fields:

- `handCards`: Current player's hand cards (string array)
- `publicInfo`: Public information (object array), `rest` indicates remaining card count
- `selfRank`: Own team's rank (string)
- `oppoRank`: Opponent team's rank (string)
- `curRank`: Current rank (string)
- `curPos`: Current player's seat number (integer)
- `curAction`: Current player's action (tuple)
- `greaterAction`: Highest action (tuple)
- `greaterPos`: Seat number of player with highest action (integer)

- `actionList`: Available action list (tuple array)
- `indexRange`: Maximum index value (integer, index range 0 to indexRange, inclusive)

Response: Send `PLAY` message, `act` must be selected from `actionList` (by index: `actionList[selectedIndex]`)

4.2.2 Tribute Request

Identifier: `"type": "act", "stage": "tribute"`

Message Format:

```
{
    "type": "act",
    "handCards": ["H3", "D3", ...],
    "selfRank": "2",
    "oppoRank": "9",
    "curRank": "9",
    "stage": "tribute",
    "actionList": [["tribute", "tribute", ["D2"]]],
    "indexRange": 0
}
```

Fields:

- `handCards`: Current player's hand cards (string array)
- `selfRank`: Own team's rank (string)
- `oppoRank`: Opponent team's rank (string)
- `curRank`: Current rank (string)
- `actionList`: Available tribute action list (tuple array)
- `indexRange`: Maximum index value (integer)

Response: Send `TRIBUTE` message, `act` must be selected from `actionList`

4.2.3 Return Tribute Request

Identifier: `"type": "act", "stage": "back"`

Message Format:

```
{
    "type": "act",
    "handCards": ["H2", "S3", ...],
    "selfRank": "5",
    "oppoRank": "9",
    "curRank": "9",
    "stage": "back",
    "tributePos": 3,
    "tribute": "S2",
    "actionList": [
        ["back", "back", ["H2"]],
```

```
        ["back", "back", ["S3"]],
        ...
    ],
    "indexRange": 11
}
```

Fields:

- `handCards`: Current player's hand cards (string array)
- `selfRank`: Own team's rank (string)
- `oppoRank`: Opponent team's rank (string)
- `curRank`: Current rank (string)
- `tributePos`: Seat number of player who paid tribute (integer)
- `tribute`: The card offered as tribute (string)
- `actionList`: Available return tribute action list (tuple array)
- `indexRange`: Maximum index value (integer)

Response: Send `PAYTRIBUTE` message, must include `tributePos` and `tribute` (matching received values), `act` must be selected from `actionList`

---

5. Data Format Specification

5.1 Card Representation

Cards are represented as strings of length 2: `{Suit}{Rank}`

Suits: S (Spade), H (Heart), C (Club), D (Diamond)
Ranks: A, 2-9, T (10), J, Q, K, B (Small Joker), R (Big Joker)

Examples: `"S2"` (Spade 2), `"HQ"` (Heart Q), `"DT"` (Diamond 10), `"SB"` (Small Joker), `"HR"` (Big Joker)

5.2 Pattern Representation

Patterns are represented as tuples: `[Pattern, Rank, CardList]`

Pattern Types: Single, Pair, Trips, ThreePair, ThreeWithTwo, TwoTrips, Straight, StraightFlush, Bomb, FourKings, tribute, back, PASS

Examples:

- `["Single", "5", ["D5"]]`: Single Diamond 5
- `["Pair", "4", ["H4", "C4"]]`: Pair of 4s
- `["PASS", "PASS", "PASS"]`: Pass
- `["tribute", "tribute", ["D5"]]`: Tribute Diamond 5

5.3 Seats and Teams

Seat Numbers: 0, 1, 2, 3 (fixed 4 seats)
Team Assignment: Team 0 (seats 0 and 2), Team 1 (seats 1 and 3)

---

6. Important Notes

1. **Action Selection**: Actions must be selected from the `actionList` provided by the server, selected by index: `selectedAction = actionList[selectedIndex]`
2. **Action Format**: The selected action must exactly match an entry in `actionList` (including card order)
3. **Connection Disconnection**: Automatic reconnection is not supported; a new connection must be established
4. **Return Tribute Restriction**: Cards returned in tribute cannot be greater than 10 (J, Q, K, A, etc. are invalid)