Monopoly - CS3305

Docs for our Monopoly project

View on GitHub (https://github.com/crnbrdrck/Monopoly)

Monopoly (/Monopoly/) API (API) GUI (GUI) Player (Player) Server (Server)

API

Our API between server and client is based on a simple JSON structure

```
{
    "command": "COMMAND_NAME",
    "values": {
        "value_name": "value"
    }
}
```

For port values used, see the Server (Server) documentation

Note: Success / Failure

When a message below says **Returns: Success / Failure**, expect the following JSON message

```
{
    "command": "COMMAND",
    "values": {
        "status": "1" for success else "0"
    }
}
```

where COMMAND is replaced with the command that was sent initally

Game Discovery Commands

These commands are used by Clients to find and join open games

Create

```
{
    "command": "CREATE",
    "values": {
        "game": "Monopoly",
        "username": str host_username,
        "password": str password or None
    }
}
```

- This will normally be sent to the localhost, but it allows for externally located servers also (later)
- If no password is used, password will be None
- Else the password should be encrypted using the following: `sha256(password.encode()).hexdigest()`
- The Server will use the socket object obtained from accepting this connection to add to the map
- The game value must be specified as Monopoly so the server will not accidentally be created for other games
- **Returns: Success / Failure**

Poll

```
{
    "command": "POLL",
    "values": {
        "game": "Monopoly"
    }
}
```

- This is used to discover any open games on the network.
- This is the only message that will be sent and received using UDP, since you cannot broadcast with TCP
- The value is important to determine that the correct game is being polled for
- **Returns: GAME from every open game on network**

Join

```
{
    "command": "JOIN",
    "values": {
        "game": "Monopoly",
        "username": str username,
        "password": str password or None
    }
}
```

- This will be sent to a server found using the POLL command
- If no password is used, password will be None
- Else the password should be encrypted using the following: `sha256(password.encode()).hexdigest()`
- **Returns: Success / Failure**

Client-to-Server Commands

These commands are used to pass user input to the server to control the state of the game

Quit

```
{
    "command": "QUIT",
    "values": {}
}
```

- Instructs the Server that this Client wishes to quit from the game
- **Returns QUIT**

Start

```
{
    "command": "START",
    "values": {}
}
```

- Instructs the server to start the game
- Anyone can send this message, it will only work if there are 2 or more players joined
- **Returns: START / Failure**

Roll

```
{
    "command": "ROLL",
    "values": {}
}
```

- Instructs the server to roll a dice for the client that sends the request
- **Returns: ROLL**

Buy

```
{
    "command": "BUY",
    "values": {
        "buy": 1 or 0 for YES or NO
    }
}
```

- Replies to the Server's BUY? with whether they want to purchase the property they are on or not
- Will be updated later to include support for houses / hotels
- **Returns: BOUGHT**

Sell

```
{
    "command": "SELL",
    "values": {
        "tiles": [int tile1, int tile2, ..., int tilen]
    }
}
```

- Instruct the server to mortgage the properties identified by the tiless tile1 to tilen
- Will be expanded later to include support for houses / hotels
- **Returns: PAY**

Chat

```
{
    "command": "CHAT",
    "values": {
        "text": str text
    }
}
```

- Instruct the Server to pass on a chat message to all clients
- The server will automatically attach things like the username of the sender
- **Returns: CHAT**

End Turn

```
{
    "command": "END",
    "values": {}
}
```

- Informs the Server that the current player's turn is now over

Server-to-Client Commands

These commands are used to inform clients of an update to the state

Game

```
{
    "command": "GAME",
    "values": {
        "game": {
            "players": [str username for Player in game],
            "password": bool has_password
        }
    }
}
```

- This message is sent as a response to a `POLL` request
- The Client can use these messages to build up a list of currently open games on the network

Start

```
{
    "command": "START",
    "values": {
        "players": {int player.id: str player.name for player in players
}
        "local": int local_player_id
    }
}
```

- Sent in response to the host sending a `START` request
- Informs the Clients that the game has started
- Informs all clients of the ids of the players in the game for update purposes
- Specifies the id of the local player also for ease

Turn

```
{
    "command": "TURN",
    "values": {
        "player": int player_id
    }
}
```

- Inform all the clients whose turn it is

Roll

```
{
    "command": "ROLL",
    "values": {
        "roll": [int dice, int dice]
    }
}
```

- Informs a Client of their Roll value when they send a roll request
- Sends both dice value to inform the Client if they got a double

Buy Request

```
{
    "command": "BUY?",
    "values": {}
}
```

- Asks the Player whose turn it is whether or not they'd like to buy the property they are standing on

Bought

```
{
    "command": "BOUGHT",
    "values": {
        "player": int player_id,
        "tile": int tile
    }
}
```

- Informs all clients that the Player player_id bought the property at position tile

Sold

```
{
    "command": "SOLD",
    "values": {
        "player": int player_id,
        "tiles": [int tile1, int tile2, ..., int tilen]
    }
}
```

- Informs all clients that player player_id has sold the properties at positions tile1 to tilen

Go To

```
{
    "command": "GOTO",
    "values": {
        "player": int player_id,
        "tile": int tile
    }
}
```

- Instruct clients that the player player_id has moved to tile

Jailed

```
{
    "command": "JAIL",
    "values": {
        "player": int player_id
    }
}
```

- Instructs the clients that the player player_id has been sent to jail, or freed from jail

Pay

```
{
    "command": "PAY",
    "values": {
        "player_from": int player_id or None,
        "player_to": int player_id or None,
        "amount": int amount
    }
}
```

- Instructs clients that player from has paid amount to to
- Either from or to can be None, indicating a payment from / to the Bank
- Only one of these can be None in any one payload

Card

```
{
    "command": "CARD",
    "values": {
        "text": str text,
        "is_bail": bool is_bail
    }
}
```

- Sends the text of a Chance / Community Chest card that a client has landed on to the client
- The actual mechanism of the card will be handled by the server
- If is_bail is True, then the client will be awarded a Get out of jail free card (maybe implement later?)

Quit

```
{
    "command": "QUIT",
    "values": {
        "player": int player_id
    }
}
```

- Tells all other clients that a Player has left the game

Chat

```
{
    "command": "CHAT",
    "values": {
        "player": str username or None,
        "text": str message
    }
}
```

- Sends a chat message to all players
- If username is None, the message is directly from the server
- Else it is from the player named username

Game Over

```
{
    "command": "GAMEOVER",
    "values": {}
}
```

- Informs all clients that the game is now over

Implementation

- We intend to have a method in our server to handle all of these messages in separate threads to keep the server running quickly.
- We will separate our communication and logic into a Server and Board class respectively.
- The Server will make use of methods in the Board to handle messages sent from Clients
- The Server will also have chat functionality built in, to be handled solely by the Server

- The Server can also use the chat functionality to inform the Players of events

Back to Top