Symbols Game: A Decentralized Multiplayer Tic-tac-toe

DS Project Description, Group 15
Xinyang Chen, Runjie Fan, Sergei Panarin, Axel Wester

Overview

Symbols Game is a decentralized multiplayer tic-tac-toe, where multiple (2-5+) players take turns trying to mark their symbols on a shared board, and win by having multiple symbols in a row.

In this implementation of the game, multiple nodes run the same software, each node maintains its own game state, takes turns coordinating the game, and progresses the game by passing state changes with messages through sockets.

Core Functionalities

Starting a Session

During the discovery phase, one node is designated as "host", and coordinates this phase of the game.

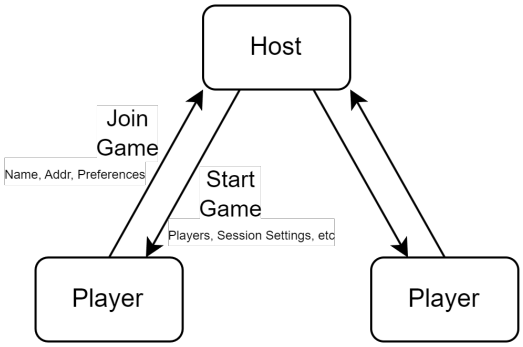

Figure 1: Nodes in the discovery phase

- To join, players connect to the host which has a known address, and announces to the host with their name, address, and other potential player preferences.
- Once everyone has joined, the host can then choose to start the game, where the current game session settings are determined like board size, each player's symbols, players' order in taking turns, etc.
- To announce the game starting, host sends the name and address of all players involved, and the current session settings, to all the participating players.
- Once acknowledged, the game starts, with an empty board, and the first player starting their turn.

Playing a Turn

In this phase, players take turns coordinating the game. The player that is taking the turn drawing is the coordinator of this turn.

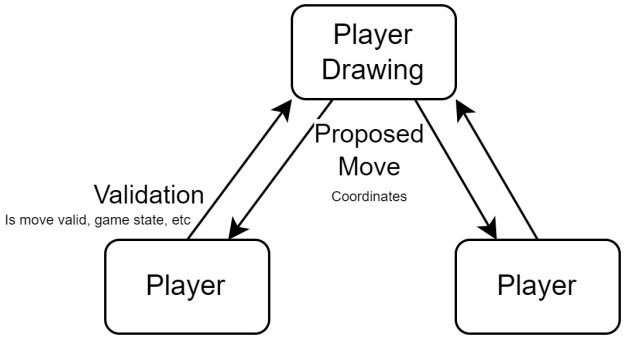

Figure 2: Nodes when Playing a Turn

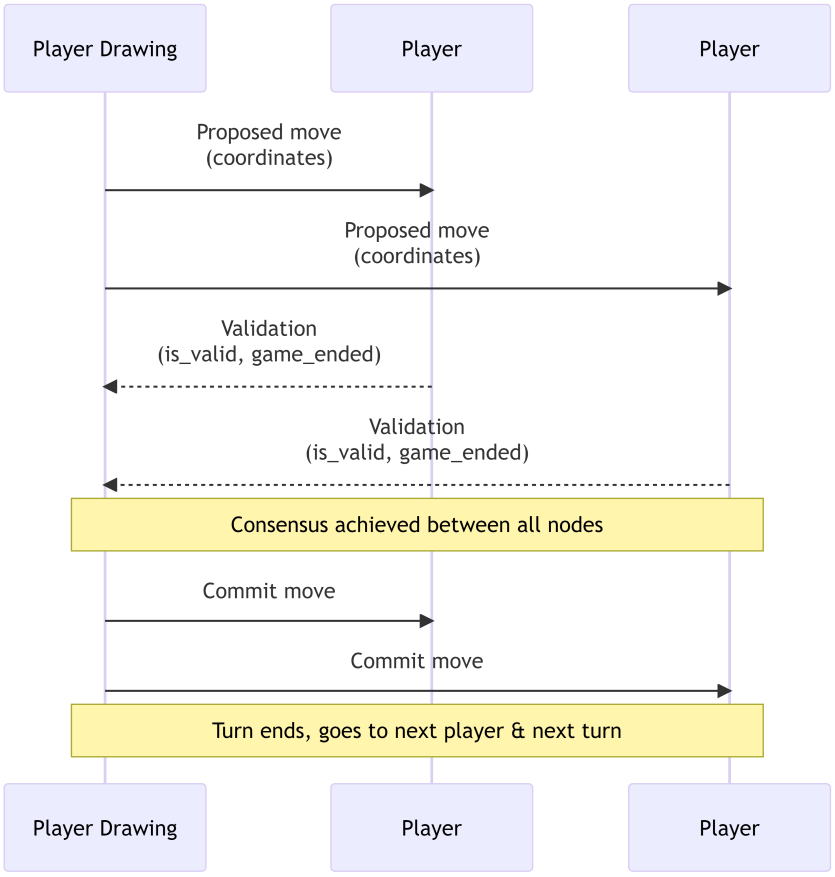

Figure 3: Sequence of Messages within a Turn

- The players that are not drawing are waiting for the player drawing to make a move.
- Once the player drawing has decided on a move, they can broadcast their proposed move to all other players.

- All other players can then validate the proposed move, and also sees if the game has ended.
- Other players then reply back to the player drawing with their validation results, and once a consensus is made, the proposed move is considered accepted.
- All players update their internal board state to refelect the latest move made in this turn.
- Next turn begins, the next players becomes the player drawing, and continues until the game ends.
- Game ends when all players agree that the game has ended.

Fault Detection

We mainly consider the following two faliure scenarios when we're in-game:

- One (or more) messages lost
- One node fails (disconnects without notice)

With the messages lost scenario, we currently hard-fail that to be treated the same as a complete node faliure. With node faliures, we immediately end the game, as a reduction in player count will defeat goals of the main gameplay loop. In later stages of the project, we might experiment on different ways of recovering from these two faliure scenarios more gracefully.

If a node fails in-game, depending on which node it is:

- If it's the player drawing, aka the player making and coordinating the turn: faliure would seem like it's taking too long for the player to draw. We can detect this with a timeout.
  - The time for the human player to decide on the move is also included within a timeout, so a longer timeout might be preferable.
- If it's any other player: faliure would seem like it's taking too long for for players to validate the proposed move.
  - We might prefer a shorter timeout here, as it's only going to be network RTT + some very light compute doing the validations.

Shared Game State

The main game state is composed of the following components:

The game board, which contains all previous moves made by players.

- Every node maintains this log seperately, and only updates are broadcasted between nodes.

- To update the game board, player-in-turn proposes a new move to all other players and wait for a validation. Once consensus on validation is reached, this move is committed on all nodes

Which player's turn it is.

- At the start of the game, a fixed cyclical order is chosen
- When a move achieves consensus and is committed to the game board, it goes to the next player in the order

Game state, which shows if the game is still going, or it has already ended. Game state changes when players reach consensus during validation.

Architectural Decisions

The nodes are to be implemented in Python. Communication between nodes are done with sockets, and messages encoded with JSON.

Message Semantics

Joining a game (players -> host):

```
{
    "action": "join_game",
    "name": "some_player",
    "address": "127.0.0.1:65535",
    "preferences": {}
}
```

Response (host -> player): simple ACK with ID

Starting a game (host -> players):

```
{
    "action": "start_game",
    "players": [
        {"id": 1, "name": "im_the_host", "address": ":", "symbol": "x"},
        {"id": 2, "name": "some_player", "address": ":", "symbol": "o"},
        {"id": 3, "name": "another_player", "address": ":", "symbol": "+"}
    ],
    "board_size": 4,
    "turn_order": [2, 3, 1],
    "session_settings": {},
}
```

Making a move (player drawing -> others):

```
{
    "action": "propose_move",
    "location": [3, 2]
}
```

Validation response (other players -> player drawing):

```
{
    "action": "validate_move",
    "valid": true,
    "game_result": null,
    "winning_player": null
}
```

Committing (player drawing -> others): simple ACK

Ending game (player drawing -> others):

```
{
    "action": "end_game",
    "reason": "win_or_tie",
    "game_result": "win",
    "winning_player": 1
}
```