

Tic-Tac-Toe

Distributed Systems project

Demo

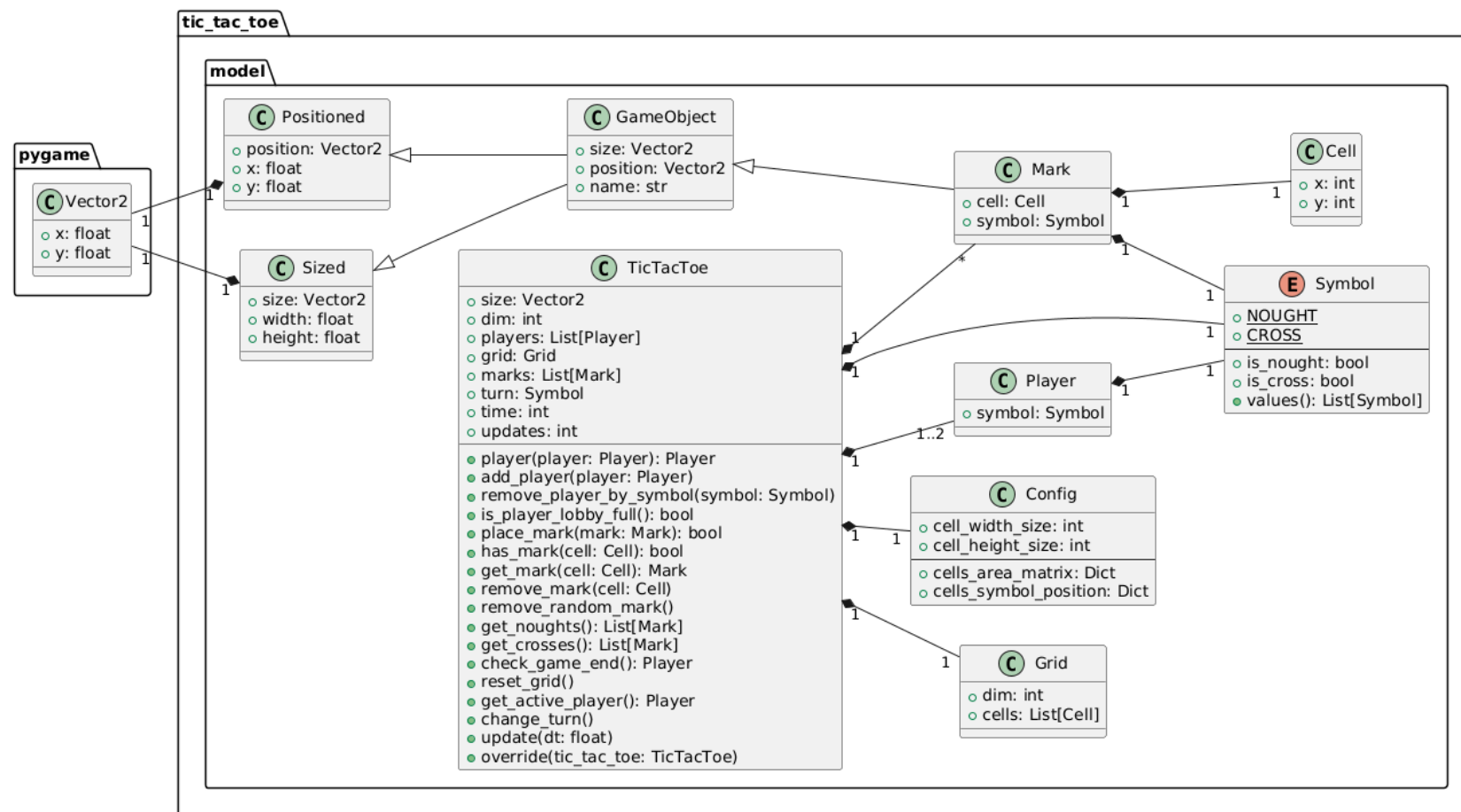
```
C:\Users\Utente\Desktop\FILIPPO\UNIVERSITA\MAGISTRALE\Distributed_S  
ystems\Tic-Tac-Toe>poetry run lobby-coordinator
```

```
C:\Users\Utente\Desktop\FILIPPO\UNIVERSITA\MAGISTRALE\Distributed_S  
ystems\Tic-Tac-Toe>poetry run terminal
```

```
C:\Users\Utente\Desktop\FILIPPO\UNIVERSITA\MAGISTRALE\Distributed_S  
ystems\Tic-Tac-Toe>poetry run terminal
```

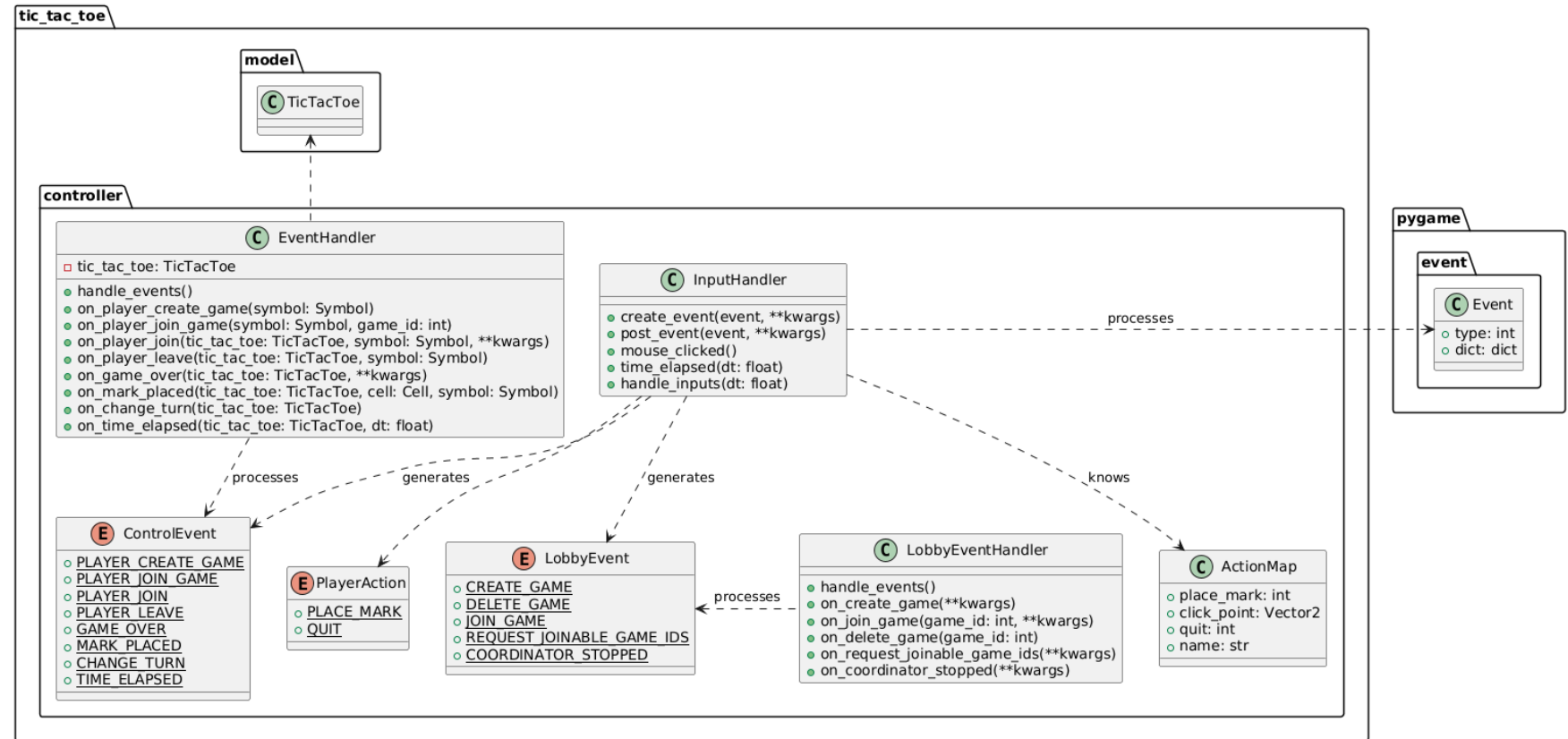
Local architecture

- **Model-View-Controller (MVC);**
- the **TicTacToe** entity represents the global game state.



Distributed architecture

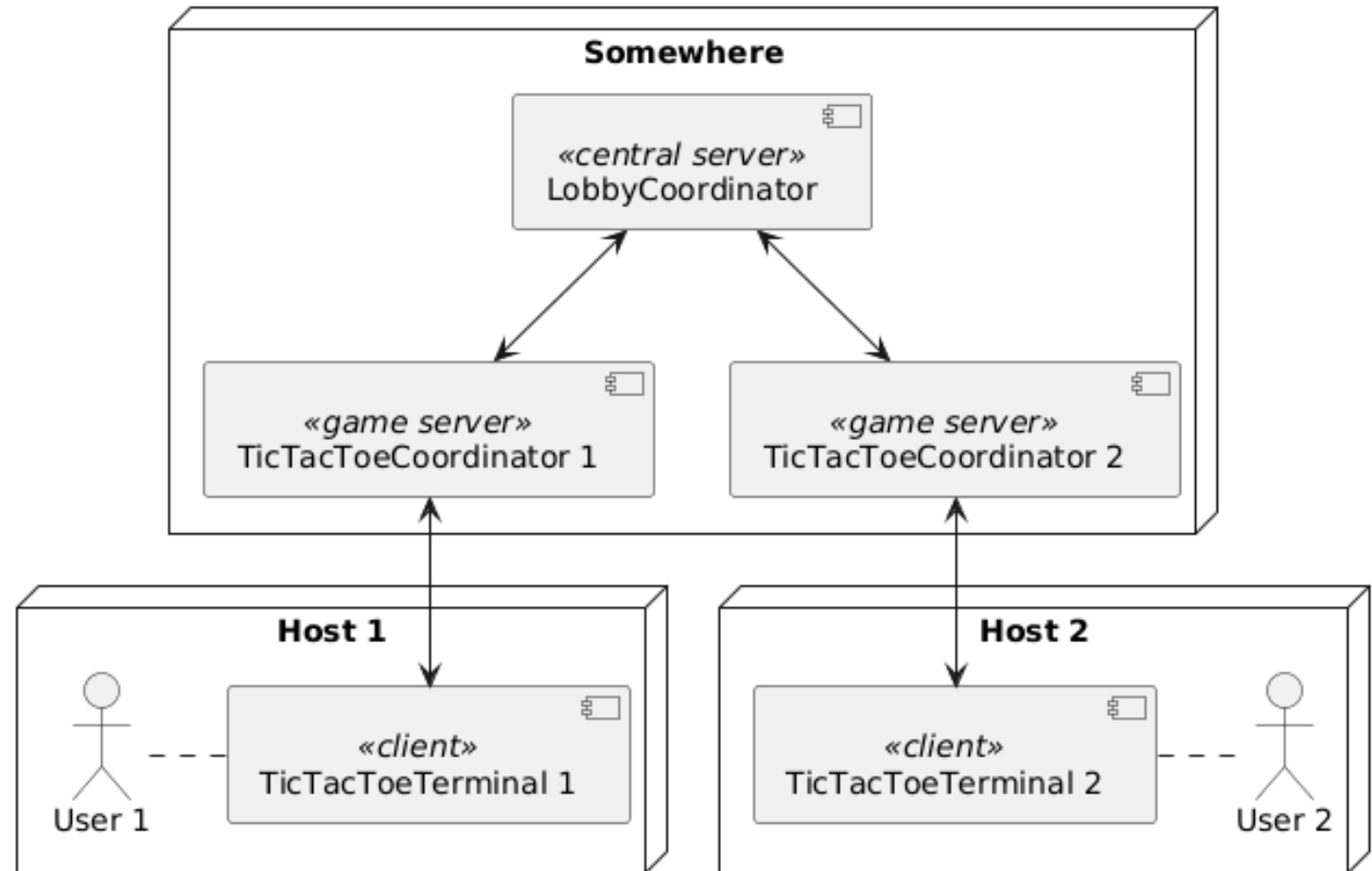
- **Event-Based Client-Server;**
- the controller handles events and inputs;
- events are divided into **ControlEvent** and **LobbyEvent** types;



Infrastructure

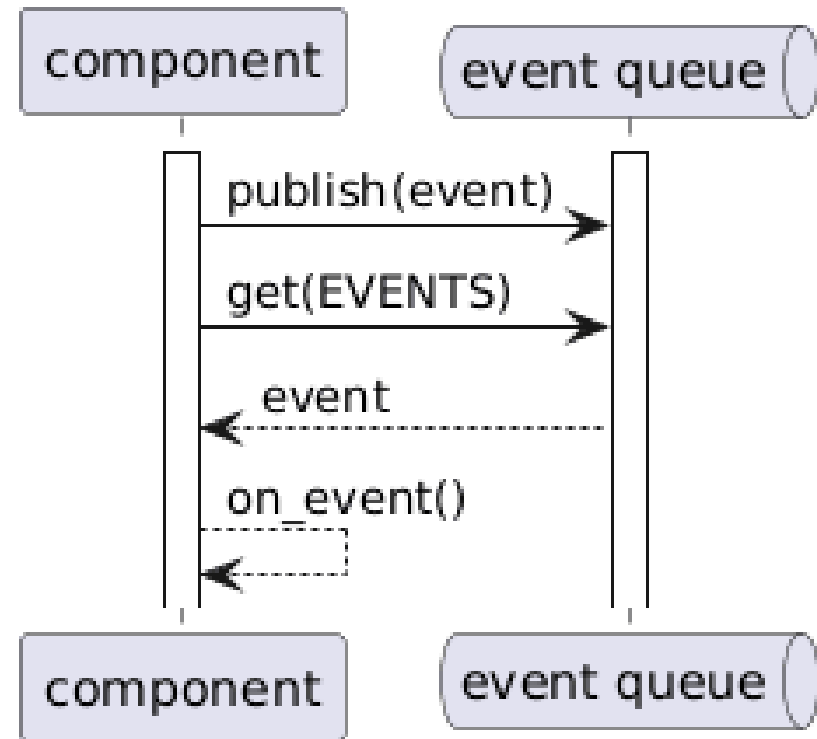
- Each player is a client, which runs a **TicTacToeTerminal**;
- the central server is a **LobbyCoordinator**;
- the game servers are **TicTacToeCoordinator** and each one manages a game.

- The central server is a singleton;
- game servers run on the same machine, but as separate processes from the central server;
- clients can be distributed on different hosts, as long as they are able to reach the central server.



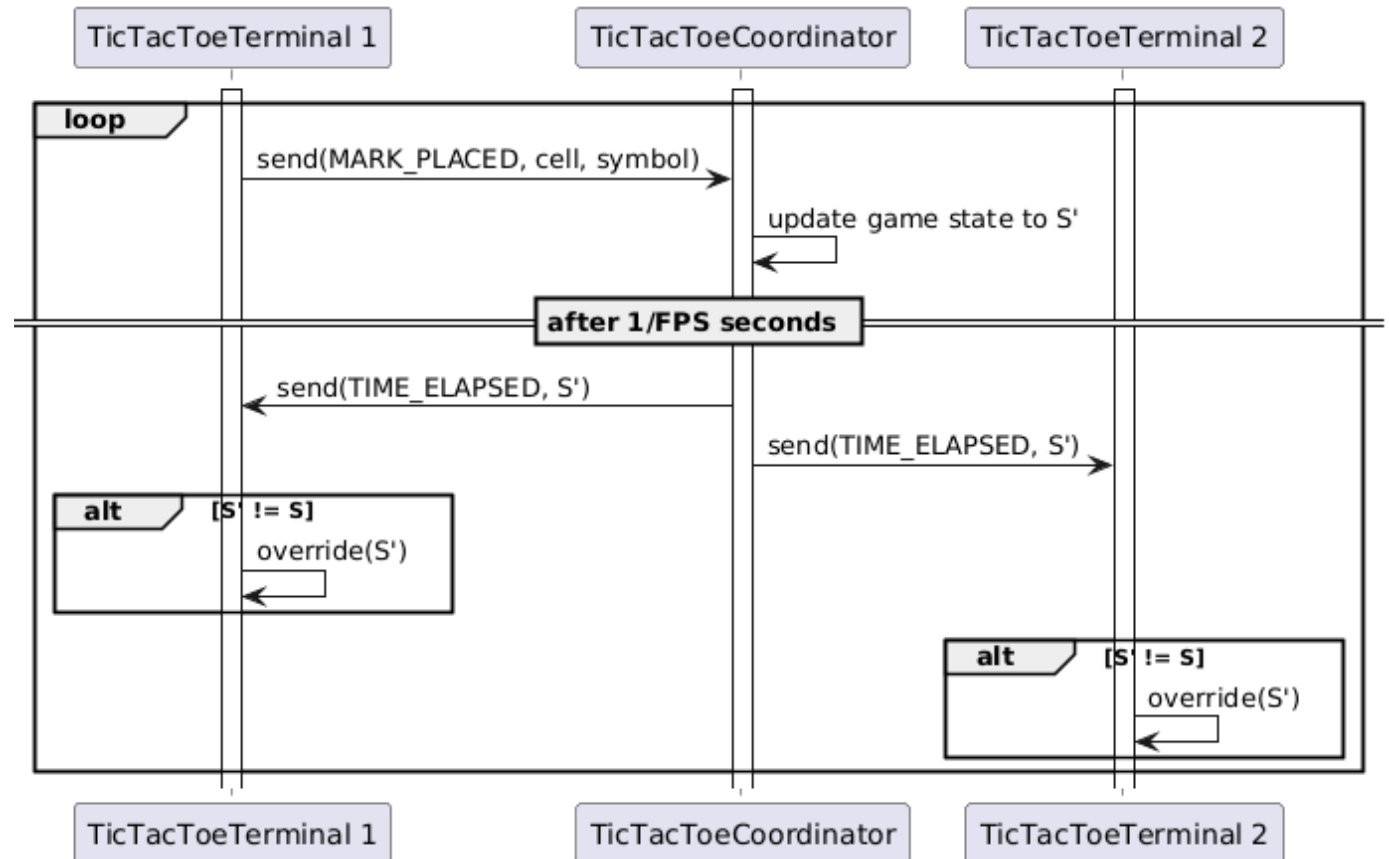
Interaction: intra-component

- **Pygame** event queue;
- analogous for clients and servers;
- event queue populated by the component itself;
- loop in which:
 - event handler checks for new events in the queue;
 - processes one at a time calling the corresponding handler.



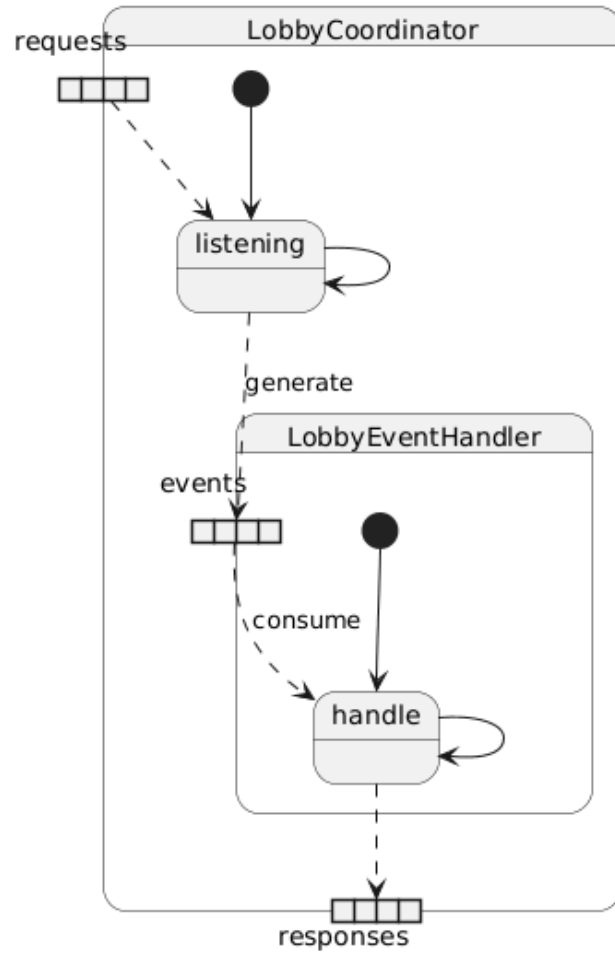
Interaction: inter-component

- Messages exchange through the network;
- in the figure shown the game state update interaction;
- interaction sequence:
 - the client handles the input and sends the event to the game server;
 - the game server updates its game state;
 - every $1/FPS$ seconds the game server sends a *TIME_ELAPSED* event to both clients with the game state;
 - the clients override their game state if different.

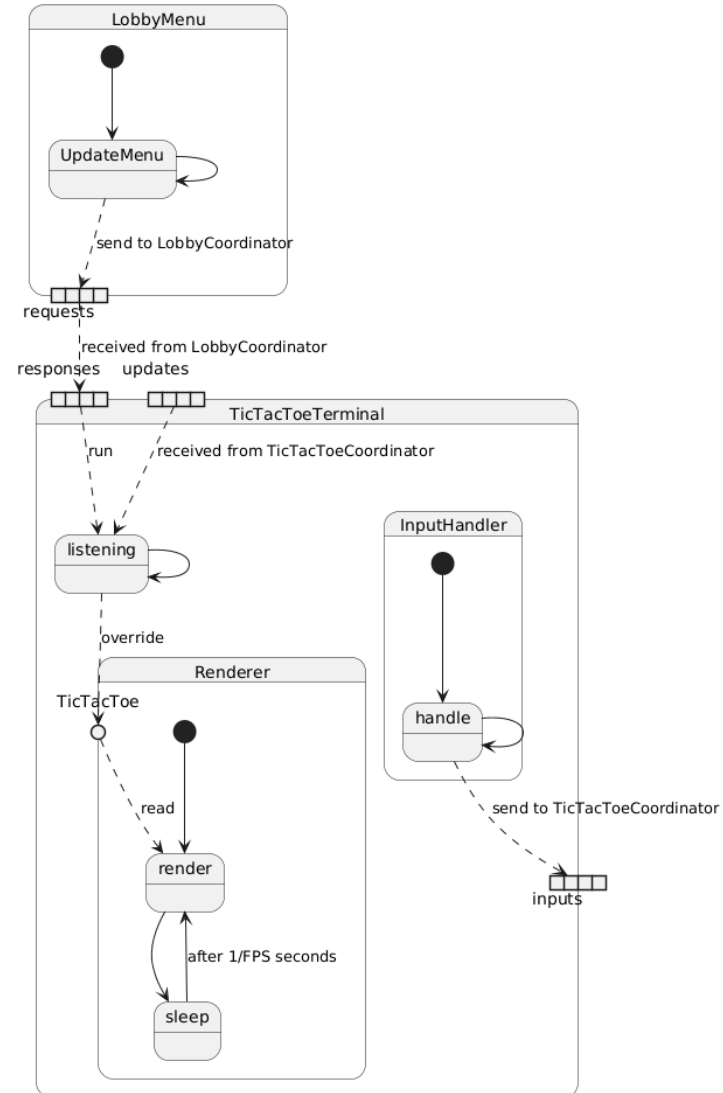


Behaviour

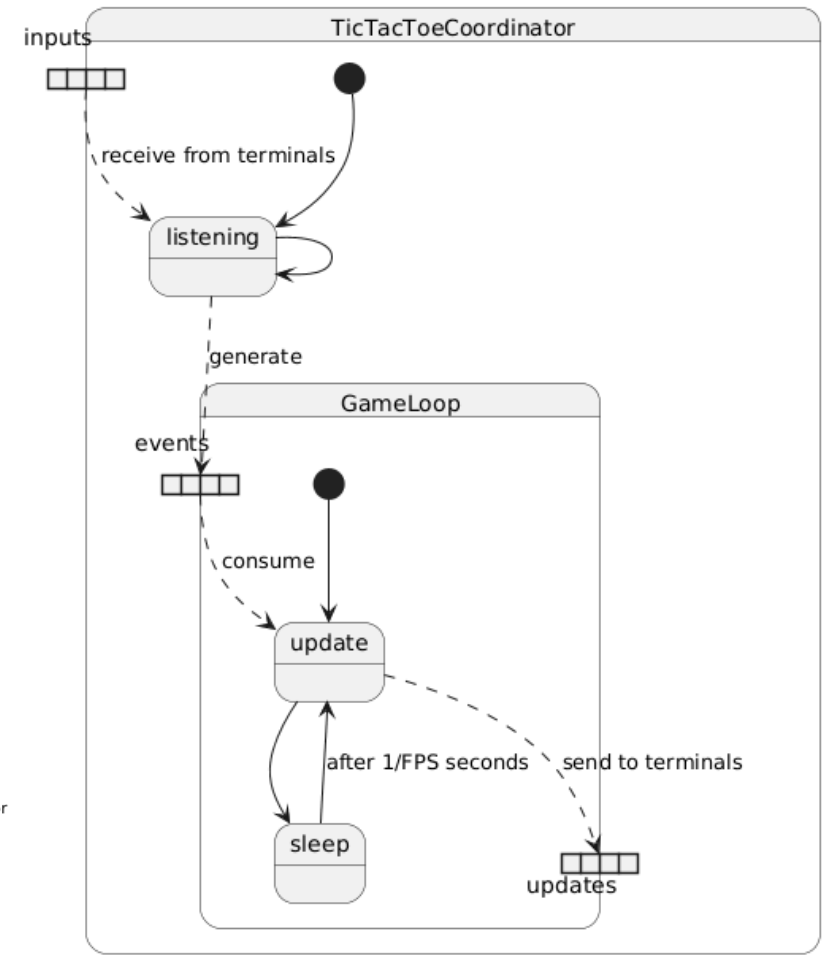
Central server



Client



Game server



Distributed properties

Data and consistency

- No persistent data storage;
- temporary *JSON* file on central server for list of joinable games;
- list of running game servers in central server's memory;
- game server is the authoritative source of game state.

Fault tolerance

- Only replication is game state between game server and clients;
- no heart-beating or retry mechanism;
- checking continuously for new messages, so error received in case of disconnection.

Availability

- In case of network partitioning the system will prefer consistency over availability (*CAP theorem*), because of:
 - turn-based nature of the game;
 - not very high frequency of interactions;
- clients terminated and game ended in case of game server disconnection.

Security

- No need for cryptographic schemas, authentication and authorization mechanisms, because of:
 - lack of persistent and sensitive data;
 - absence of user management.

Implementation

- **TCP** as network protocol, chosen over UDP for:
 - high reliability;
 - turn-based nature of the game gives importance to consistency of game state;
 - package loss is more critical than latency;
- **JSON** for in-transit data representation;
- the central server sends back to the client:
 - the game server address to connect to, in case of creation or joining of a game request;
 - the list of joinable game in case of that request;
- chat between players as a *daemon* thread in the client.

Project references

- Repository:
<https://github.com/FilippoPracucci/Tic-Tac-Toe.git>
- Forum topic:
<https://virtuale.unibo.it/mod/forum/discuss.php?d=195999#p268655>