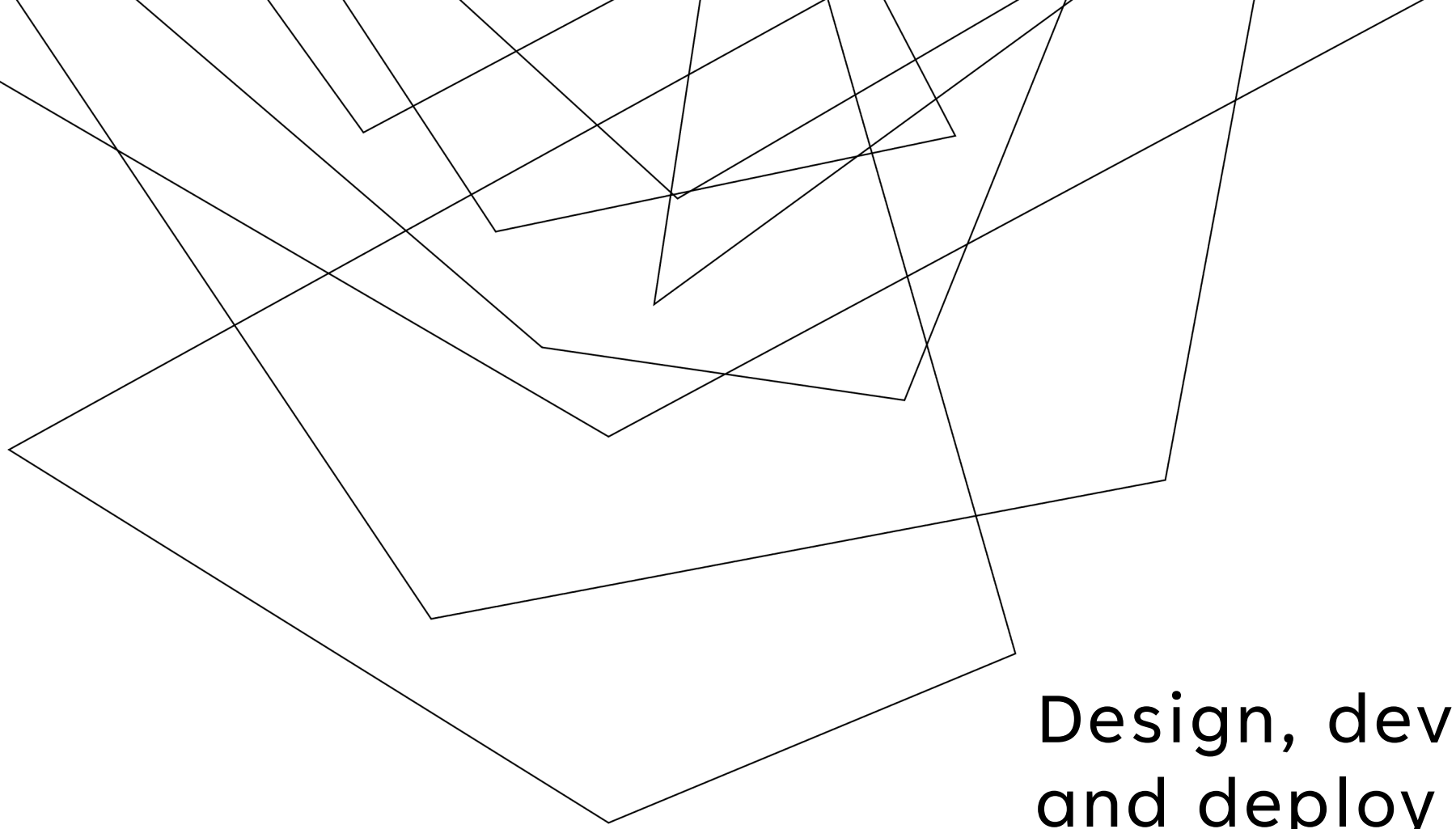


Design, development and deploy of ”RisiKo!” Clone

Squarcialupi Riccardo – Benvenuti Filippo

Distributed peer-to-peer Approach

Why this project?

- The adoption of a peer-to-peer model in tabletop gaming presents a myriad of opportunities for the implementation of sophisticated algorithms, which we actively pursued and integrated into our project.

Project Architecture:

- Java-based P2P Client
- Python RestFul WebServer:
  - Needed for discover and share players IPs

WebServer Comunication

API-REST:

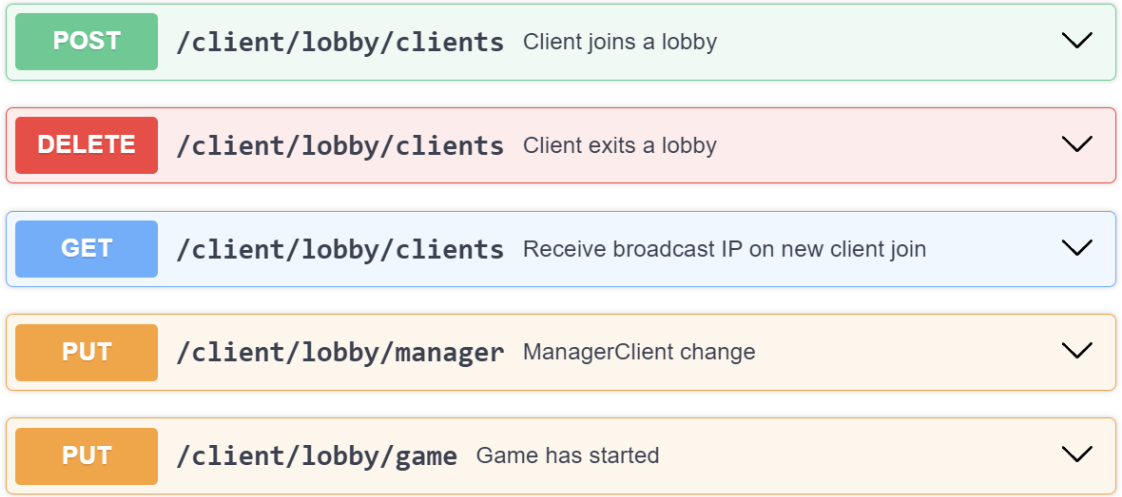
- Flask: framework for fast development
- Serialize/Deserialize JSON
- Informations exposed via URI
- Information saved in a simple data struct

GET	`/server/lobbies`	Get lobbies
POST	`/server/lobbies`	Create a new lobby
PUT	`/server/lobby/{id}`	Connect to a lobby
DELETE	`/server/lobby/{id}`	Exit lobby (decrement number of players)
DELETE	`/server/lobby/{id}/numberOfPlayer`	Exit lobby (decrement number of players)
PUT	`/server/lobby/{id}/managerClientIp`	Update manager client information from the selected lobby

P2P Client Lobby Comunication

API-REST:

- Vert.X: java monolithic comunication framework
- Jackson: Serialize/Deserialize JSON
- Shared state across clients
- Elected Manager among clients share infos with WebServer



P2P Client Game Comunication

PUT	`/client/game/territories`	Receive broadcast territories of another client
PUT	`/client/game/turn/finish`	Get notify a client has finished his turn
PUT	`/client/game/territory/armiesWithConqueror`	Get notify when a territory is co
PUT	`/client/game/card`	Get notify when bonus from State Card have been used
PUT	`/client/lobby`	Lobby close
PUT	`/client/game/battle/offense`	Get notify if an offense move is starting
PUT	`/client/game/battle/defense`	Get notify of the defense response
PUT	`/client/game/stateCard`	Get notify when someone draws a State Card

PUT	`/client/game/win`	Get notify if a Client has won the game
GET	`/manager/game/territory`	Get a state card from the Manager
PUT	`/client/game/territory/armies`	Get notify when armies number change
POST	`/client/game/dice/throw`	Get notify a client wants to throw a dice
PUT	`/client/game/dice/confirm`	Receive dice confirm intention
PUT	`/client/game/dice/share`	Get dice share from other clients
PUT	`/client/game/armies/update`	Get broadcast when army is placed
PUT	`/client/game/order`	Get random turns order
PUT	`/client/game/player`	Get notify when a player has left

Testing

JUnit5 Testing:

- API communications tests
- Components state tests

ThunderClient Testing:

- Lightweight version of Postman being used for API initial declaration and development
- Useful tool for fast manual testing
- Post-production usage for benchmarking API performances

Docker - Deployment

Deployment:

- Easily distributed across OS
- Each Endpoint has its own container:
  - Clients
  - Server

Docker-compose for running:

- Deploy 3 clients and 1 server
- Distinct version for Windows and Linux/MacOS

Docker-compose for testing:

- Deploy 1 server and 3 gui-less clients

Docker - X11 GUI

Natively docker cannot display a graphics interface, in our project the X Window System (X11/Xming depend on the OS) was used to bypass this limitation.

Main features:

- Provides the basic framework for a GUI environment
- X’s network protocol is based on X command primitives, allows both 2D and 3D operations by an X client application which might be running on a different computer.

NON-TRIVIAL ALGORITHMS
DICES THROW

Two phases:

- ***Throw init***:
  - Generate random number
  - Generate random key
  - Generate hash of number with key
  - Send hash to all clients
  - Receive random number as response
  - *Clients shares random numbers*
- ***Dice confirm***:
  - Sum all random numbers
  - Send random dice and random key
  - Clients recompute hash of random number with the key provided
  - If match dice throw is eligible

Note that no client can alonely cheat on the dice throw, even if more clients decide to cheat together, they won't be able to decide the final random number.

NON-TRIVIAL ALGORITHMS
GAME TURN ORDERING

Sender Side:

- Generate random turn order (shuffledList).
- Store locally in randomOrderList with client IPs.
- Asynchronously transmit orders via HTTP PUT.
- Handle responses and Return a Future<Void> upon completion or failure.

Receiver Side

- Receive incoming HTTP PUT requests and extract IP and random order from the request body.
- Order Processing:
  - Update local random order list if orderFound is false.
  - Check if random orders from all clients are the same.
    - If not, trigger cleanup and retransmit with a randomly picked order.
    - If orders are the same, set orderFound to true.
  - Handling Order Found:
    - Print order found message.
    - Resend order to all clients for synchronization.
    - Initiate turn if it’s the first player’s turn; otherwise, enter waiting phase.
  - Optional: Simulate resending for redundancy.

WHAT WE LEARNED:

Dockerfile Expertise:

- Valuable insights from Dockerfile creation:
  - Optimizing container builds.
  - Minimizing image sizes.
  - Streamlining application deployment.

Network Programming Challenges

- Encounter with inherent difficulties:
  - Heisenbugs highlighting complexity.
  - Insights into the challenges of network programming.

Vertx Considerations:

- Comprehensive framework with myriad features.
- Potential drawbacks for simpler projects:
- Monolithic nature.
- Significant learning curve.

Service Scaling:

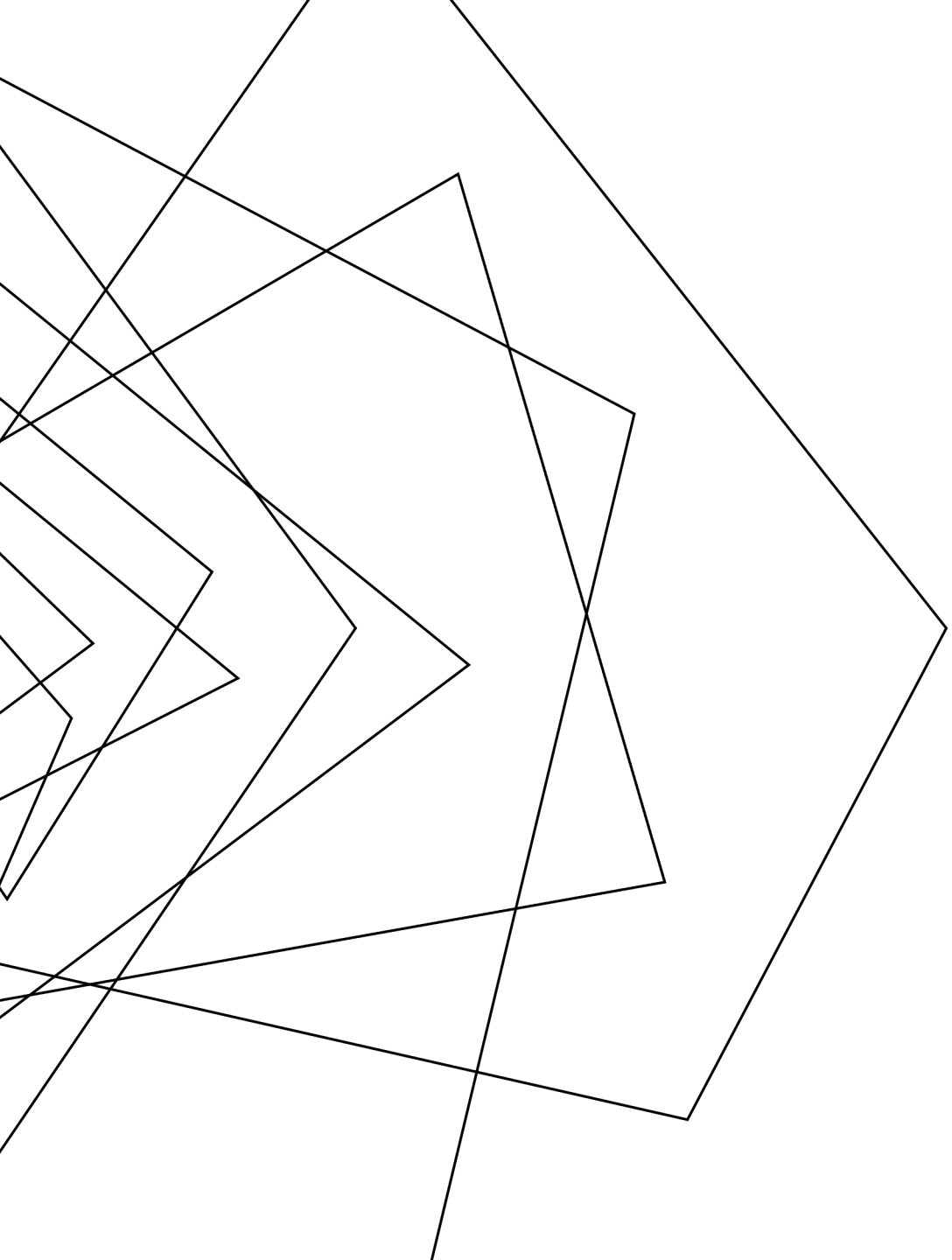
- Importance of creating larger services in 'compose.yaml.'
- Optimizing resource utilization for enhanced system performance.

Environment Variable Management:

- Effective strategies for handling environment variables.
- Ensuring seamless configuration and adaptability in different deployment scenarios.

Shared Volumes:

- Implementation of shared volumes for:
- Data persistence.
- Efficient communication between system components.



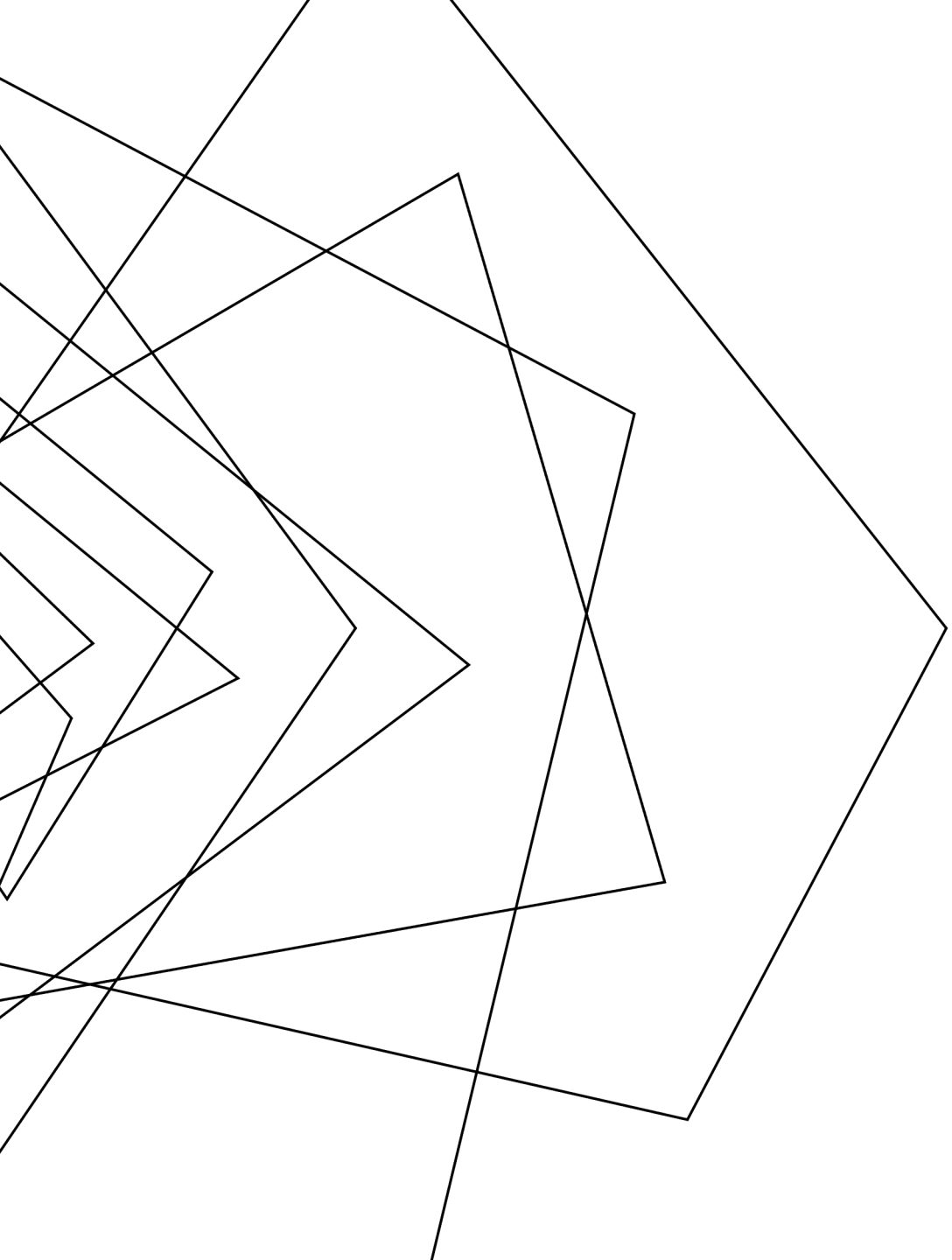
FUTURE WORKS

Major:

- Incognito cards drawing
- Handle client crash, improve stability during rare scenarios:
  - Timeout
  - Keep Alive packets

Minor:

- Adding live chat inside games and lobby
- Add GPL license



THANKS FOR:

La pazienza