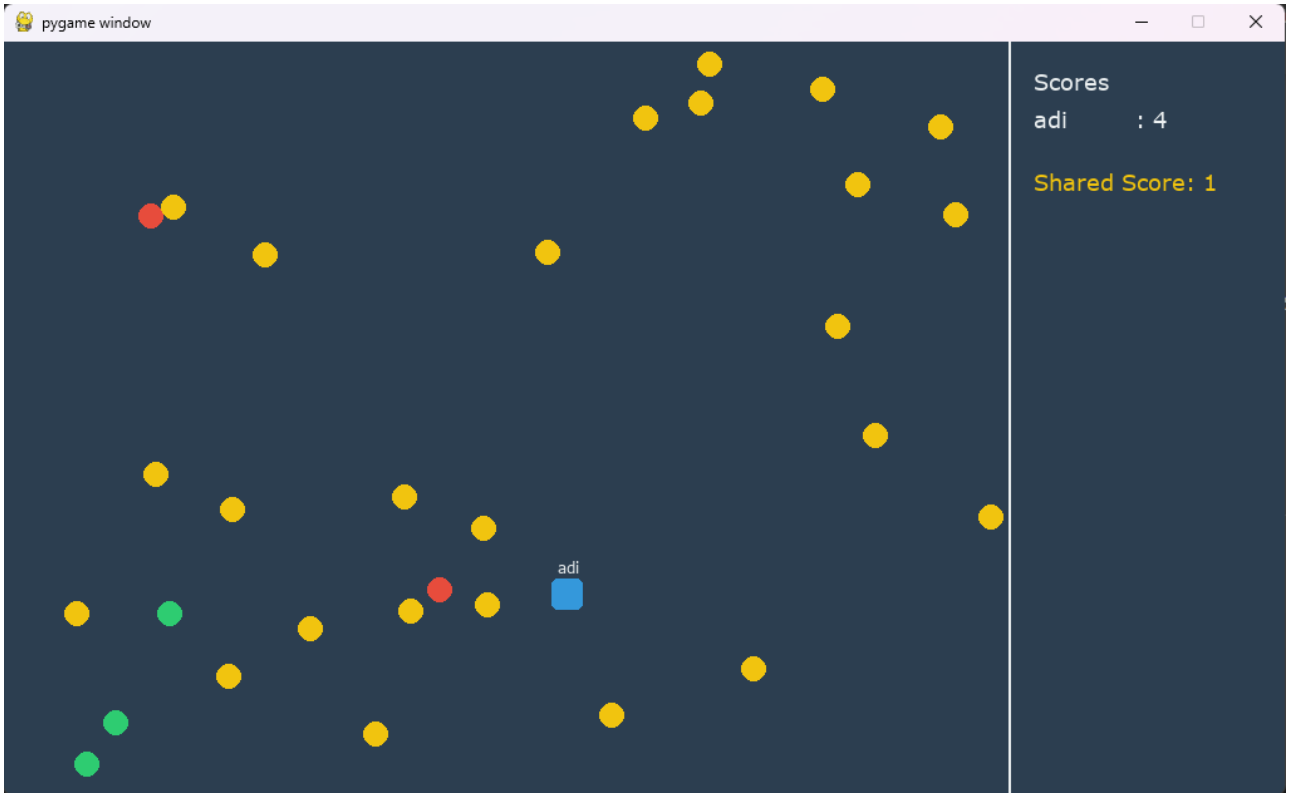
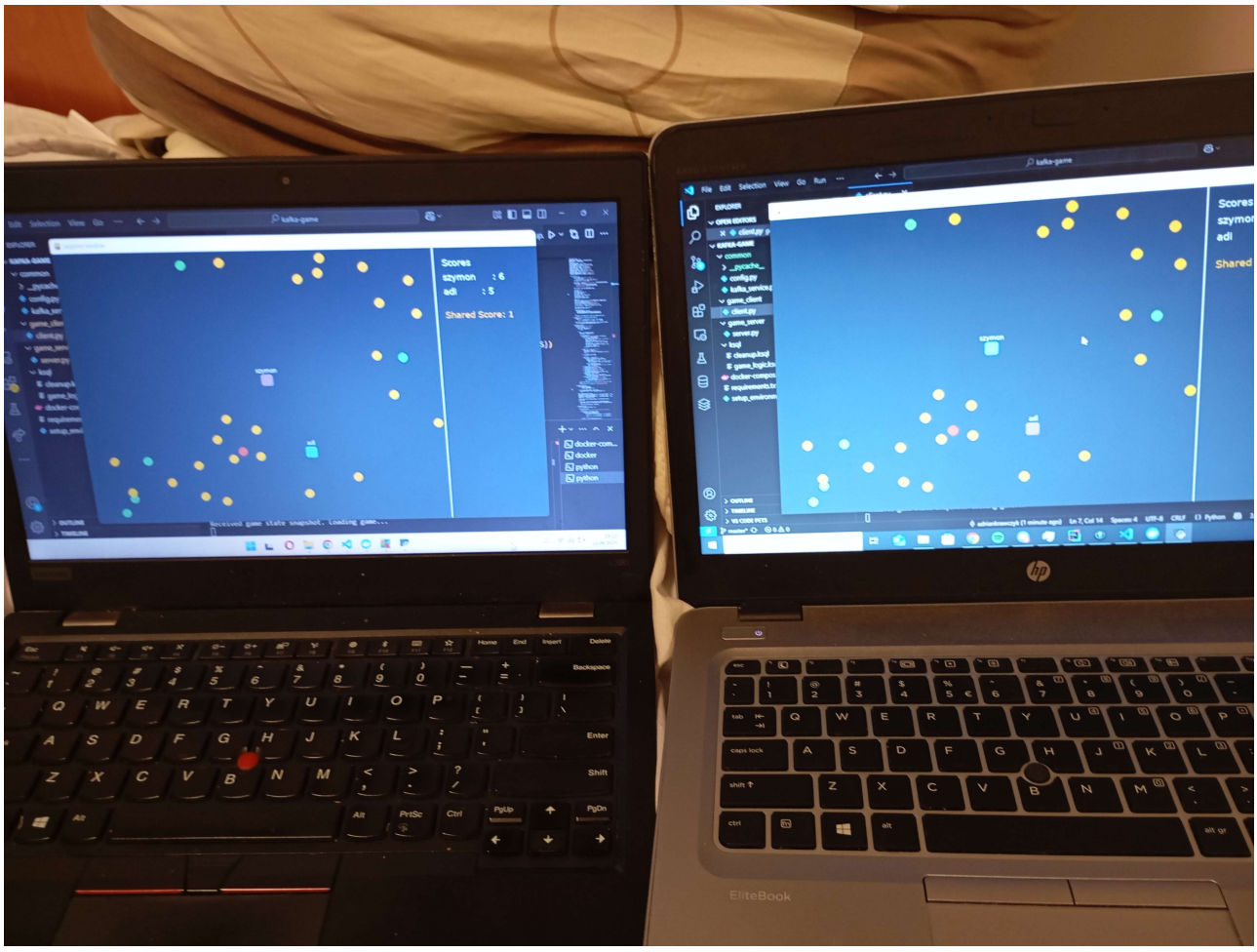
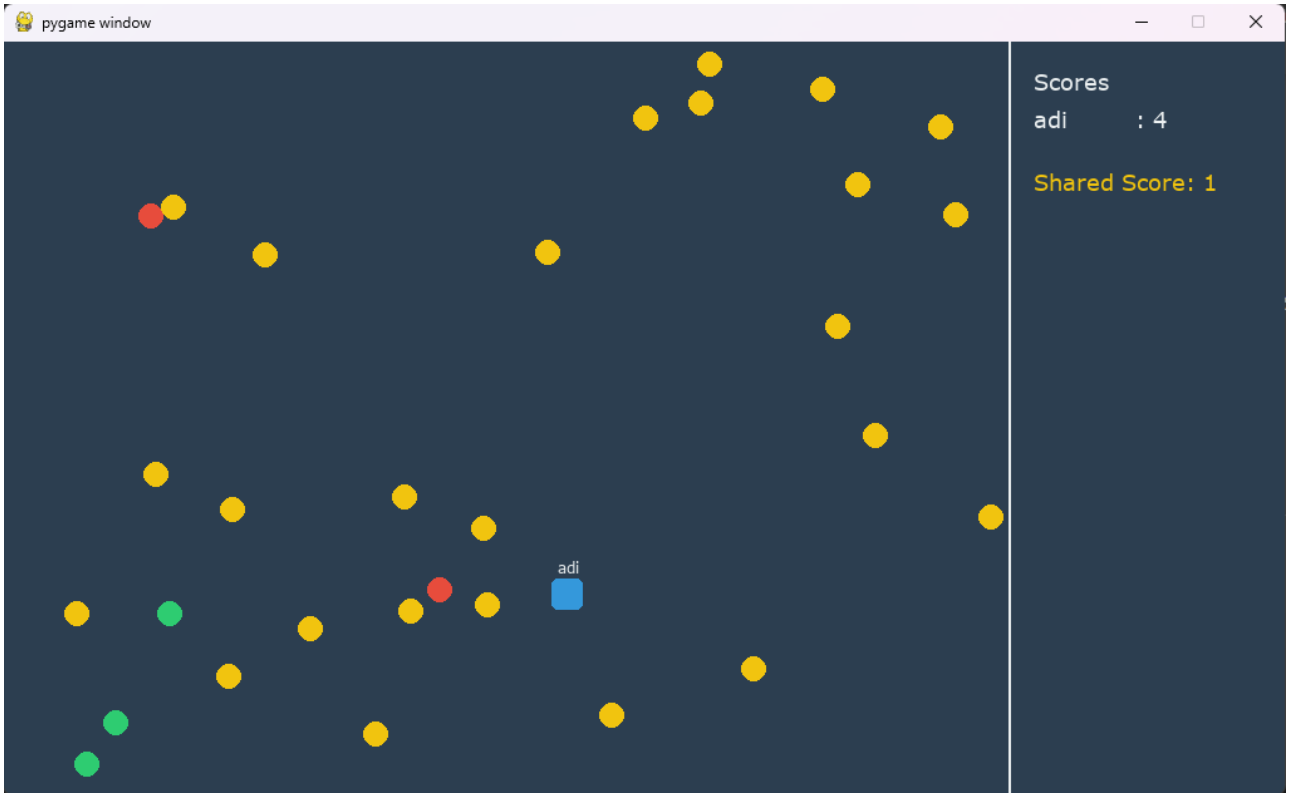
Kafka Game: Real-Time Multiplayer Dot Collector 🎮

Experience dynamic multiplayer action fueled by the power of event-driven architecture! This project showcases the usage of streaming databases for multiplayer games! We used Apache Kafka, which is an open-source distributed event streaming platform and ksqlDB, a streaming SQL engine.

Watch the gameplay:

- 

- 

- 

Overview

This is a simple, proof of concept gameplay made just to show how the streaming databases work in multiplayer game solutions. Players control squares, aiming to collect dots that appear randomly. Each collected dot contributes to an individual score, while special dots can affect a shared team score. When a

player gets 30 points, he wins. The game uses streams to ensure reliable communication between clients and the server. ksqlDB is employed for stateful stream processing, providing a real-time view of the game state. The entire backend infrastructure is containerized with Docker for easy deployment and management.

✨ Key Features

- **Real-Time Multiplayer:** Engage multiple players simultaneously in a dynamic, shared game environment.
- **Event-Driven Core:** Utilizes Apache Kafka for robust, decoupled communication, forming the heart of the game's architecture.
- **Stateful Stream Processing with ksqlDB:** Leverages ksqlDB to transform raw game event streams into tables (`PLAYERS_TABLE`, `DOTS_TABLE`), maintaining an readable game state.
- **Hybrid Communication Strategy:**
  - **Fast Path:** Low-latency `GAME_EVENTS_TOPIC` delivers immediate UI updates to clients.
  - **Authoritative Path:** Server events are fed into ksqlDB to build a consistent game state.
- **Dockerized Ecosystem:** Kafka, Zookeeper, ksqlDB Server, and ksqlDB CLI are managed via Docker for effortless setup.
- **Interactive Pygame Client:** A simple graphical client built with Pygame offers an engaging player experience.
- **Dynamic Scoring Mechanics:** Features individual scores and a shared team score, with different dot types offering varied scoring effects.
- **Clear "Game Over" Condition:** The game concludes decisively when a player's total score (individual + shared) reaches the 30 points.

🏗️ Architecture Deep Dive

The Kafka Game system is a symphony of interconnected components, communicating seamlessly via Kafka topics:

1. Game Clients: * The player's window into the game world, crafted with **Pygame**. * Handles graphical rendering, user input (W,A,S,D or arrow keys), and local prediction for smooth avatar movement. * Publishes player actions (e.g., "join", "move") to the `PLAYER_ACTIONS_TOPIC` in the database. * Subscribes to `GAME_EVENTS_TOPIC` for real-time state changes (other players' positions, dot states, score updates, and game over signals).

2. Game Server (`game_server/server.py`): * The central system of the game. * Consumes player inputs from `PLAYER_ACTIONS_TOPIC`. * Manages player states (positions, scores, activity), dot lifecycles, collision detection, and win condition evaluation. * Produces events to multiple Kafka topics, ensuring data flows correctly to both clients and ksqlDB: * `GAME_EVENTS_TOPIC`: For broadcasting fast-path updates to clients. * `PLAYER_STATE_UPDATES_TOPIC`: For feeding detailed player state into ksqlDB to build the `PLAYERS_TABLE`. * `DOT_EVENTS_TOPIC`: For publishing dot creation and collection events to ksqlDB for the `DOTS_TABLE`.

3. Apache Kafka (Managed by `docker-compose.yml`): * Enables asynchronous communication. * Broker IP and port are configured in `common/config.py`.

4. ksqlDB (Managed by `docker-compose.yml`): * A powerful stream processing engine allowing SQL-like queries on real-time Kafka data. * `ksqldb-server`: Executes the stream processing logic. * `ksqldb-cli`: Provides an interactive shell for defining and querying ksqlDB objects. * Consumes from `PLAYER_STATE_UPDATES_TOPIC` and `DOT_EVENTS_TOPIC`. * The `ksql/game_logic.ksql` script defines: *

Streams: `PLAYER_STATE_UPDATES_STREAM` and `DOT_EVENTS_STREAM` directly mapping to Kafka topics. * **Tables:** `PLAYERS_TABLE` and `DOTS_TABLE` as continuously updated materialized views, representing the canonical, queryable game state.

5. Docker & Docker Compose (`docker-compose.yml`): * Simplifies the deployment and management of the entire backend infrastructure (Zookeeper, Kafka, ksqlDB Server, ksqlDB CLI), ensuring a consistent environment.

Data Flow & Kafka Topics (defined in `common/config.py`):

- `PLAYER_ACTIONS_TOPIC`: Client → Server.
- `GAME_EVENTS_TOPIC`: Server → Client.
- `PLAYER_STATE_UPDATES_TOPIC`: Server → ksqlDB.
  - *ksqlDB Stream:* `PLAYER_STATE_UPDATES_STREAM`
- `DOT_EVENTS_TOPIC`: Server → ksqlDB.
  - *ksqlDB Stream:* `DOT_EVENTS_STREAM`
- `PLAYERS_TABLE_TOPIC` & `DOTS_TABLE_TOPIC`: Internal ksqlDB topics backing the materialized tables.

💻 Technology Stack

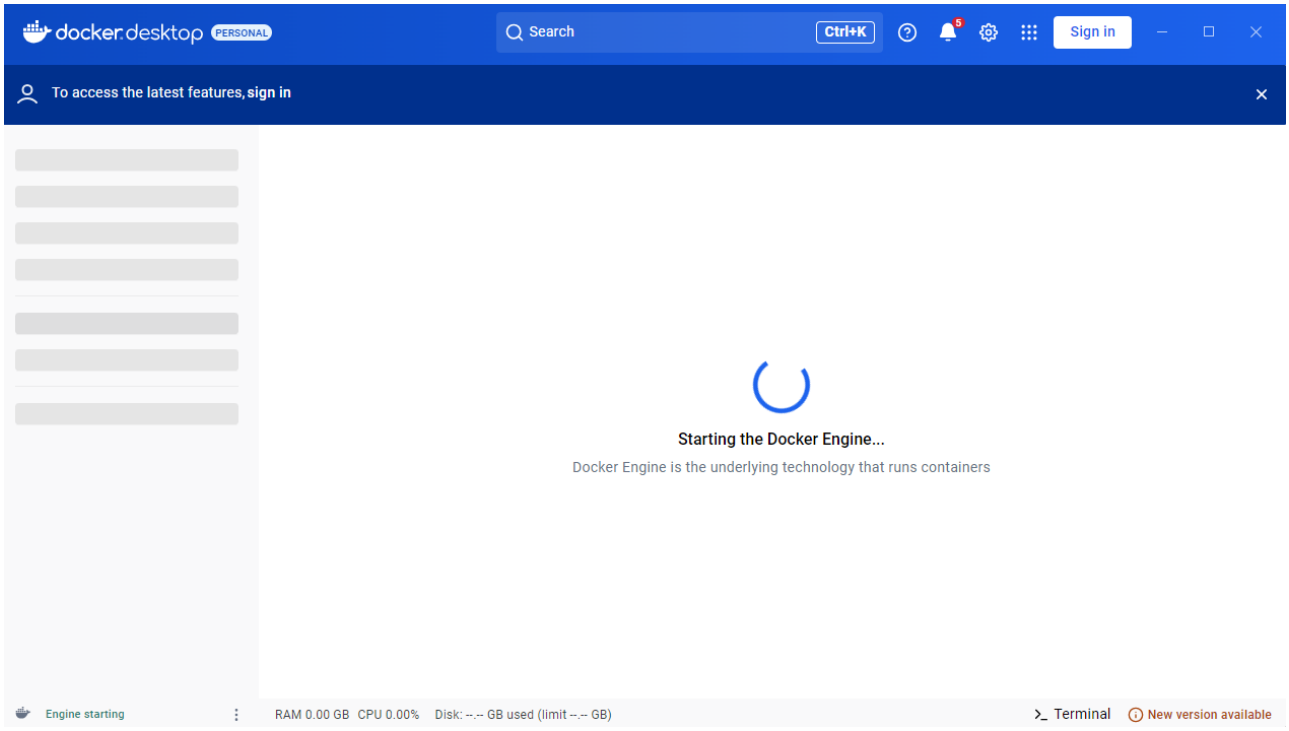
- **Backend & Game Logic:** Python 3
- **Client GUI:** Pygame
- **Messaging Backbone:** Apache Kafka
- **Stream Processing:** ksqlDB
- **Containerization:** Docker, Docker Compose
- **Kafka Python Client:** `confluent-kafka`

🛠️ Setup and Installation Guide

Prerequisites:

- Python 3.8+ and `pip`

- Docker Engine & Docker Compose (Docker Desktop is recommended)



Steps:

1. **Clone the Repository:**

```
git clone <your-repository-url>
cd kafka-game
```

2. **❗CRUCIAL: Configure Kafka Broker IP❗** Open `common/config.py`. You **MUST** update `KAFKA_BROKER_IP` to your machine's Local Area Network (LAN) IP address. This IP must be reachable by Docker containers and other clients on your network for multiplayer functionality.

```
# common/config.py
KAFKA_BROKER_IP = "YOUR_MACHINE_LAN_IP_ADDRESS" # Example: "192.168.1.105"
KAFKA_BOOTSTRAP_SERVERS = f"{KAFKA_BROKER_IP}:9092"
```

3. **Install Python Dependencies:** (Specified in `requirements.txt`)

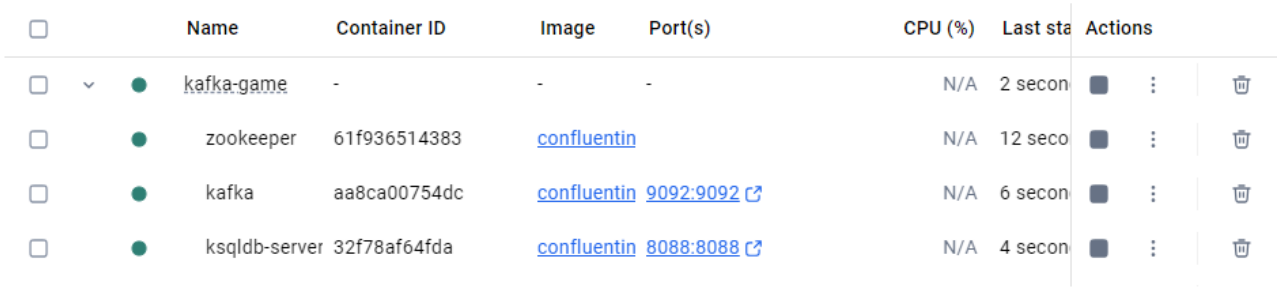
```
pip install -r requirements.txt
```

4. **Launch Dockerized Infrastructure:** From the project root:

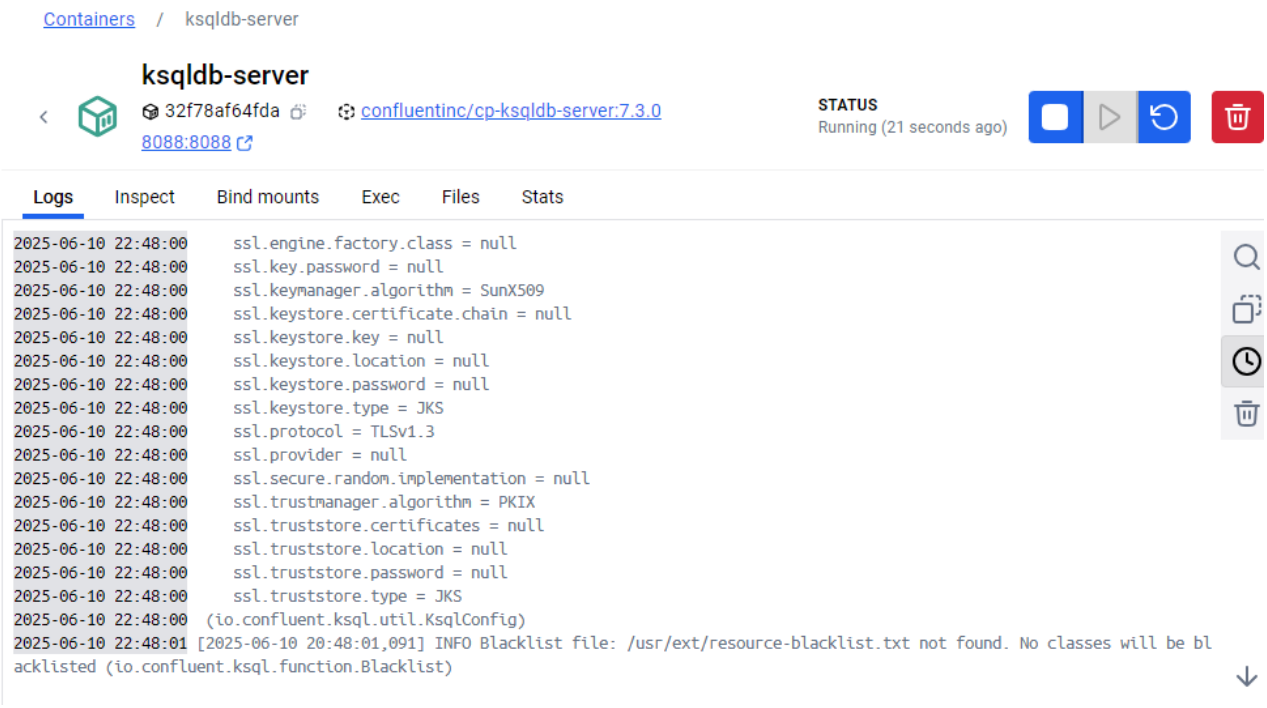
```
docker-compose up -d
```

```
PS C:\Users\Admin\Desktop\moje\4 SEM\BAZY\kafka-game> docker-compose up -d
[+] Running 4/5
 ✓ Network kafka-game_default  Created                                    0.6s
 - Container zookeeper         Starting                                   7.6s
 ✓ Container kafka             Created                                    0.6s
 ✓ Container ksqldb-server     Created                                    0.5s
 ✓ Container ksqldb-cli        Created                                    0.8s
```

Verify containers are running via Docker Desktop:



Optionally, check ksqlDB server logs for successful startup:



5. **Initialize Kafka Topics:** This script (`setup_environments.py`) creates the necessary Kafka topics.

```
python setup_environments.py
```

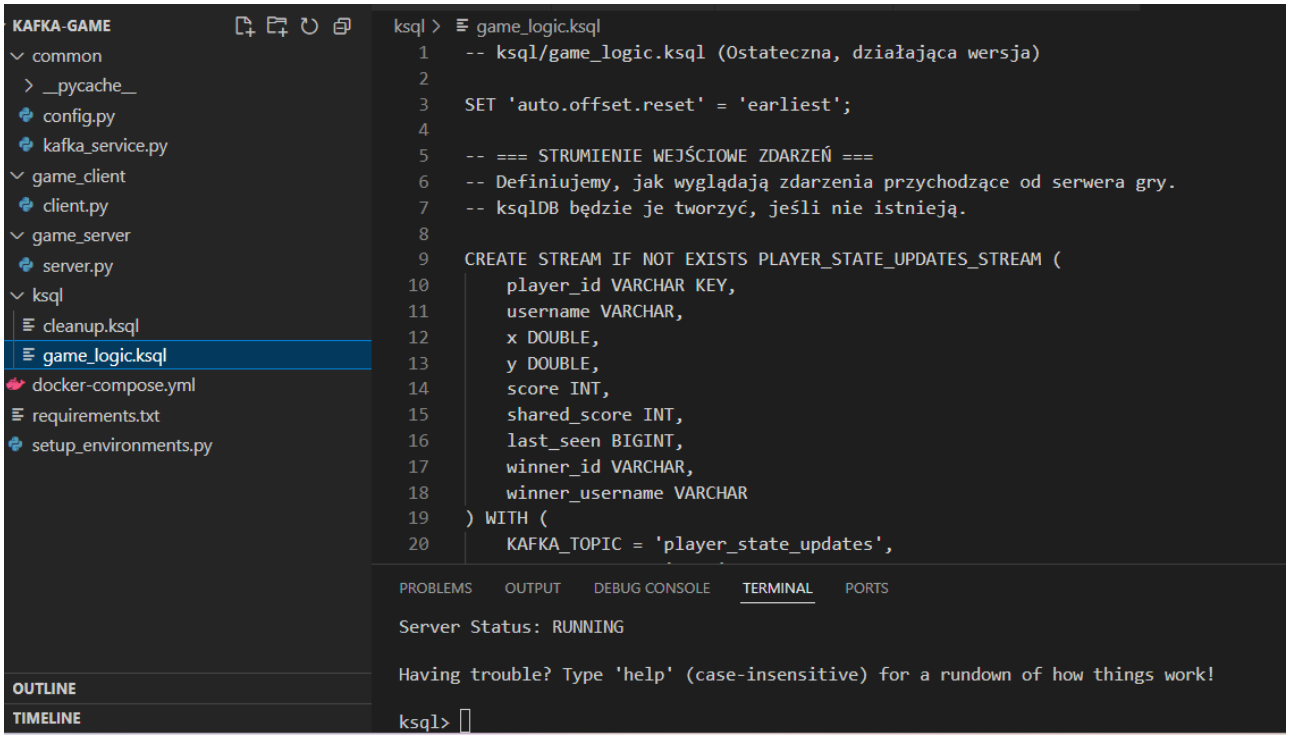
6. **Bootstrap ksqlDB Streams & Tables:** a. Access the ksqlDB CLI: `bash docker exec -it ksqldb-cli ksql http://ksqldb-server:8088`

```
PS C:\Users\Admin\Desktop\moje\4 SEM\BAZY\kafka-game> docker exec -i ksqldb-cli ksql http://ksqldb-ser
ver:8088
```

You'll be greeted by the ksqlDB CLI:

```
PS C:\Users\Admin\Desktop\moje\4 SEM\BAZY\kafka-game> docker exec -i ksqldb-cli ksql h
ttp://ksqldb-server:8088
Jun 10, 2025 9:07:03 PM org.jline.utils.Log logr
WARNING: Unable to create a system terminal, creating a dumb terminal (enable debug lo
gging for more information)
ksqlDB, Copyright 2017-2022 Confluent Inc.

CLI v7.3.0, Server v7.3.0 located at http://ksqldb-server:8088
Server Status: RUNNING

Having trouble? Type 'help' (case-insensitive) for a rundown of how things work!

ksql>
```

b. Execute the ksqlDB Logic: Open `ksql/game_logic.ksql`.



Copy the entire content of `ksql/game_logic.ksql` and paste it into the ksqlDB CLI. Press Enter. This script defines how ksqlDB processes game data:

```
SET 'auto.offset.reset' = 'earliest';

-- === STRUMIENIE WEJŚCIOWE ZDARZEŃ ===
-- Definiujemy, jak wyglądają zdarzenia przychodzące od serwera gry.
-- ksqlDB będzie je tworzyć, jeśli nie istnieją.

CREATE STREAM IF NOT EXISTS PLAYER_STATE_UPDATES_STREAM (
    player_id VARCHAR KEY,
    username VARCHAR,
    x DOUBLE,
    y DOUBLE,
    score INT,
    shared_score INT,
    last_seen BIGINT,
```

```
    winner_id VARCHAR,
    winner_username VARCHAR
) WITH (
    KAFKA_TOPIC = 'player_state_updates',
    VALUE_FORMAT = 'JSON',
    PARTITIONS = 1, REPLICAS = 1
);

CREATE STREAM IF NOT EXISTS DOT_EVENTS_STREAM (
    id VARCHAR KEY,
    x INT,
    y INT,
    type VARCHAR
) WITH (
    KAFKA_TOPIC = 'dot_events',
    VALUE_FORMAT = 'JSON',
    PARTITIONS = 1, REPLICAS = 1
);

-- === TABELE STANU (ZMATERIALIZOWANE WIDOKI) ===
-- To jest "read model" - stan, który będą czytać klienci.

CREATE TABLE IF NOT EXISTS PLAYERS_TABLE WITH (KAFKA_TOPIC='players_table_topic',
VALUE_FORMAT='JSON') AS
SELECT
    -- Poprawny sposób na włączenie klucza do wartości rekordu
    player_id AS PLAYER_ID,
    LATEST_BY_OFFSET(username) AS USERNAME,
    LATEST_BY_OFFSET(x) AS X,
    LATEST_BY_OFFSET(y) AS Y,
    LATEST_BY_OFFSET(score) AS SCORE,
    LATEST_BY_OFFSET(shared_score) AS SHARED_SCORE,
    LATEST_BY_OFFSET(last_seen) AS LAST_SEEN,
    LATEST_BY_OFFSET(winner_id) AS WINNER_ID,
    LATEST_BY_OFFSET(winner_username) AS WINNER_USERNAME
FROM PLAYER_STATE_UPDATES_STREAM
GROUP BY player_id
EMIT CHANGES;

CREATE TABLE IF NOT EXISTS DOTS_TABLE WITH (KAFKA_TOPIC='dots_table_topic',
VALUE_FORMAT='JSON') AS
SELECT
    id AS ID,
    LATEST_BY_OFFSET(x) AS X,
    LATEST_BY_OFFSET(y) AS Y,
    LATEST_BY_OFFSET(type) AS TYPE
FROM DOT_EVENTS_STREAM
GROUP BY id
EMIT CHANGES;
```

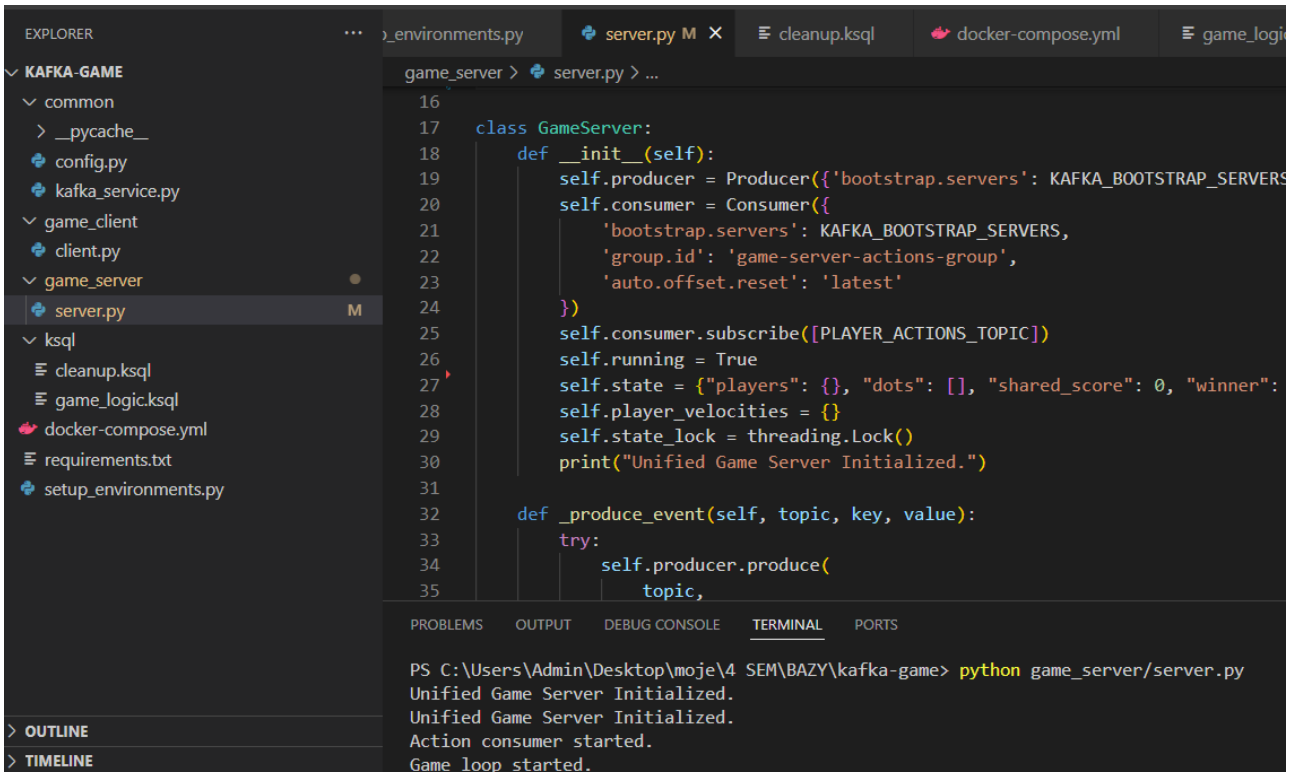
Verify Table Creation: In the ksqlDB CLI: `ksql SHOW TABLES;` You should see `PLAYERS_TABLE` and `DOTS_TABLE`.

```
ksql> ksql> SHOW TABLES;

 Table Name     | Kafka Topic          | Key Format | Value Format | Windowed

 DOTS_TABLE     | dots_table_topic     | KAFKA      | JSON         | false
 PLAYERS_TABLE  | players_table_topic  | KAFKA      | JSON         | false
```

▶ How to Run the Game

1. **Ensure Docker services are active** (`docker-compose up -d`).
2. **Confirm Kafka topics & ksqlDB setup is complete.**
3. **Start the Game Server (`game_server/server.py`):** In a new terminal:

```
python game_server/server.py
```



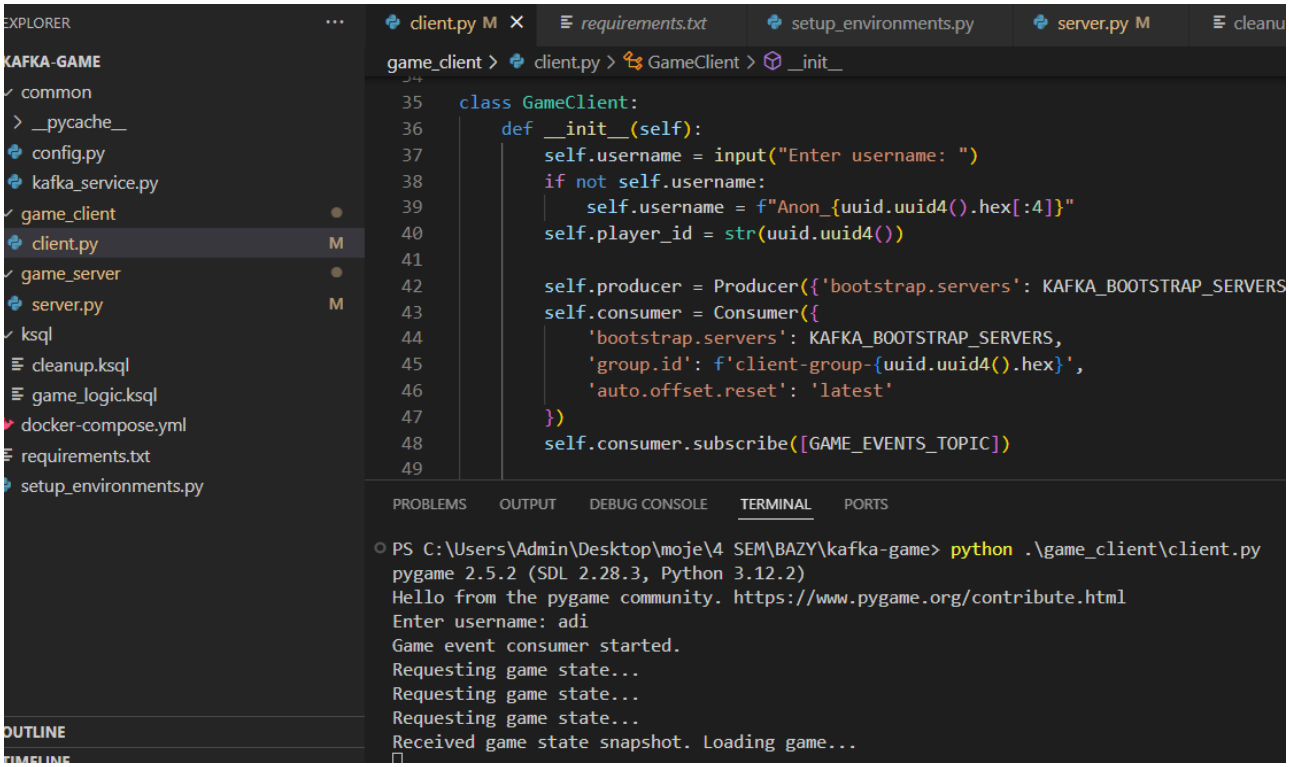
Watch for logs indicating player joins:

```
PS C:\Users\Admin\Desktop\moje\4 SEM\BAZY\kafka-game> python game_server/server.py
Unified Game Server Initialized.
Action consumer started.
Game loop started.
New player joining: szymon (ID: 02352413-f402-49ee-9600-bc727f7f997d)
Received 'join' from szymon. Sending state snapshot.
Received 'join' from szymon. Sending state snapshot.
New player joining: adi (ID: ba21f9a1-14f4-44e3-98b2-18c2d825ed3a)
```

4. **Launch Game Clients (`game_client/client.py`):** For each player, open a new terminal (on the same or different machines on the network) and run:

```
python game_client/client.py
```

Enter a username when prompted:



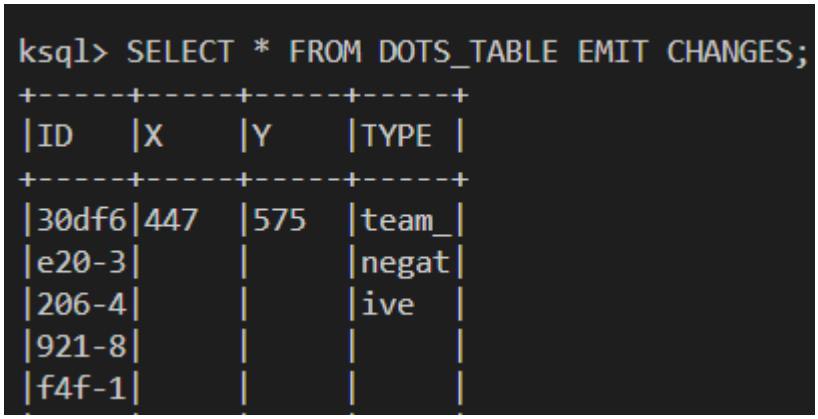
5. **Enjoy the Game!** Use `W, A, S, D` or `Arrow Keys` to move. Collect dots, watch the scores, and aim for victory!

🔍 Inspecting Live Game State with ksqlDB

Peek into the authoritative game state directly via ksqlDB:

1. Connect to ksqlDB CLI: `docker exec -it ksqldb-cli ksql http://ksqldb-server:8088`
2. Run live queries: View dot states:

```
SELECT * FROM DOTS_TABLE EMIT CHANGES;
```



View player states:

```
SELECT * FROM PLAYERS_TABLE EMIT CHANGES;
```

```
SELECT * FROM PLAYERS_TABLE EMIT CHANGES;
```

PLAYER_ID	USERNAME	X	Y	SCORE	SHARED_SCORE	LAST_SEEN	WINNER_ID	WINNER_USERNAME
99ae8	adi	460.6	426.7	7	4	17495	null	null

🛑 Stopping the Environment

To gracefully shut down and remove all Docker containers, networks, and volumes:

```
docker-compose down -v --remove-orphans
```

```
PS C:\Users\Admin\Desktop\moje\4 SEM\BAZY\kafka-game> docker-compose down -v --remove-orphans
[+] Running 5/5
 ✓ Container ksqldb-cli          Removed                                                    1.1s
 ✓ Container ksqldb-server       Removed                                                    1.0s
 ✓ Container kafka               Removed                                                    1.7s
 ✓ Container zookeeper           Removed                                                    0.2s
 ✓ Network kafka-game_default    Removed                                                    1.4s
```

We hope you enjoy playing, exploring, and learning from the Kafka Dot Collector Game!