Bully - readMe

This README outlines the operation and usage of the simulation of the Bully Algorithm using GO. The program simulates synchronization, coordinator election, and various fault scenarios in a distributed system.

Synchronization (reqSync):

1. **Node Sync Request**: A node can request synchronization using the `requestSync` command. The purpose of this is to synchronize its data (local clock) with the coordinator.
2. **Detecting Node Failure**: Periodically, each node sends a SYNC REQ to the coordinator. If a node doesn't receive SYNC RES message from the coordinator within a specified time frame (typically simulated by a fixed timeout), it considers the coordinator as unresponsive or failed.
3. **Action upon Failure Detection**: Once a node detects that the coordinator is unresponsive, it initiates the bully election algorithm to choose a new coordinator. The node that detects the failure might not necessarily become the coordinator; the node with the highest ID that is still alive will become the new coordinator.

Bully Election Algorithm:

1. **Election Initiation**: Any node can initiate an election but it only does so if it believes the coordinator has failed. When initiating election, it sends an ELECTION message to all nodes with higher IDs.
2. **Receiving Election Message**: If a higher ID node receives an ELECTION message, it sends a NACK response to the initiating node to inform it's alive and initiates its own election.
3. **Becoming the Coordinator**: If a node receives no NACK responses (after a timeout) to its ELECTION messages from nodes with higher IDs, it considers itself

the coordinator. It then sends a VICTORY message to all other nodes, announcing its new role.

4. **Responding to the Victory Message**: Nodes that receive the VICTORY message will keep quiet and will not initiate new elections unless they detect the coordinator has failed in a later SYNC REQ attempt.

Compilation and Running

1. **Compilation and running**:
   To compile and run the program, navigate to the directory containing the source code and execute:

```
cd cmd\bully
go build && main.exe (Windows)
go build && ./main (Linux)
```

Upon running, the program enters an interactive CLI mode, allowing you to input commands to control the simulation.

CLI Commands

Here are the available CLI commands:

- `startNode <number>`: Starts the specified number of nodes. For example, `startNode 5` starts nodes 0 to 4.
- `initiateElection <nodeId>`: Triggers the node with the given ID to initiate the election process.
- `killNode <nodeId>`: Kills or makes the node with the given ID unresponsive.
- `requestSync <nodeId>`: Requests synchronization from the node with the specified ID.
- `failNonCoordinatorDuringElection`: A non-coordinator node fails during the election.
- `multiElect`: Initiates multiple elections simultaneously from different nodes.
- `silentLeave`: Simulates a scenario where an arbitrary node silently leaves the network.
- `listNodes`: Lists out all nodes in the simulation with relevant information

Scenarios

1. **Basic Synchronization and Election**:

```
startNode
startNode
startNode
startNode
startNode

initiateElection 0
killNode 4
requestSync 0
```

- **Worst-Case Situation**: The 0th node initiates election in a simulation with 5 nodes (ID 0-4)

```
startNode
startNode
startNode
startNode
startNode

initiateElection 0
```

- **Best-Case Situation**: The 3rd node initiates election in a simulation with 5 nodes (ID 0-4)

```
startNode
startNode
startNode
startNode
startNode

initiateElection 3
```

- **Non-Coordinator Node Fails During Election**:

```
failNonCoordinatorDuringElection
```

- **Multiple Nodes Start Election Simultaneously**:

  ```
  multiElect
  ```

- **Silent Departure of a Node**:

  ```
  silentLeave
  ```

Interpreting the Output

The output is presented in a verbose log format. Each line describes an action or event in the simulation, such as a node initiating an election, sending a message, or becoming a coordinator.