How to execute:

Start server first:

java -jar Server.jar

Then you start a client:

java -jar Client.jar ARG1 ARG2

ARG1: Port
ARG2: Can be whatever you want. It will activate that „special settings" get sent to the client regulary

Examples: *java -jar Client.jar 4000 1* or *java -jar Client.jar 4000*
Note that the server port is 5000. Don't use the same.

Play with Rebels:

java -jar Rebels.jar ARG1 ARG2

ARG1: Port
ARG2: Number of Rebels

Example: java -jar Rebels.jar 6000 20
Note that it will create 20 rebel clients starting from port 6000. The port number increases with every client.

Discussion:
The Game itself seems to work fine, even though there are probably some bugs left, which I haven't discovered yet. To make a design document first was new to me, but it turned out great. I was able to implement all my ideas directly and had almost no changes to make.

I tried to balance the game settings to make it as playable as possible in such a short time. All the game settings I used in my code are hardcoded in a separate java file called Parameters.java.

If I had the chance to invest more time I would implement a simple database to store all the generated data to get rid of the lists and locks. The user interface is only text-based, which made it quite impossible to keep it updated automatically. The user has to update the view by himself. (Button 5).

Changes:

Below you can find a list of function/ideas I wanted to implement according to my design document and how I implemented them.

Planned	Implemented
FEC Method: Triple Redundant	Implemented. Sequence number stays the same to know it is a redundant message.
Packet loss to simulate network problems	A random number generator simulates a packet loss. The standard loss is 10% (can be changed).
Bots simulate opponents	The class rebel_logic.java generates clients, which use a simple logic to interact with the server.
Text-based user interface or graphical	There was not enough time to implement a gui. The text-based interface is very simple. It is not possible to update it automatically. The user has to update it by himself.
Infinite number of players but there is a max number of players per map	For the default parameters 15 players per map are able to join. If a new player joins, a new map gets created. Infinite maps are possible. The bigger the map, the more players can join.
Client tries to connect 3 times, before stopping.	Client sends in total three registration messages till the client stops trying.
When clients disconnect, their forts get taken over by rebels.	Their forts stay in the game, but the „rebels" don‘t produce anything. Simply because in this version of the game the rebels are acutal clients and no AI which is implemented on the server-side.
Exception handling on the client side	The client checks every time if he has enough resource to produce anything. If not, the server won‘t get informed.
Time-based operation on client-side	All time-based operations are happening on the client-side. After an operation the client sends an information to the server.
Use UDP as communication protocol	UDP is implemented. To differentiate the clients from each other I used their ports.
Packet size will be 512 bytes. Header contains message type, sequence number, player id and fec method.	The header looks slightly different. It contains sequence number, player id (port), message type and instead of the fec method it contains the world/map id the client is playing in.
10 different message types	There are 2 little changes. Instead of 3 different message for increase mining, producing soldiers and defence there is just one left. The Client always sends all information combined in one

	message to the server. The second change is that there is a new message type for registration. If a client wants to connect, they use that type of message. In total there a 9 different message types now.
Server and client consists of 4 main parts.	This is still the case. It is hard to observe because to keep the code readable there are multiple classes now. Still the logic is in control of all classes, the listener keeps listening, the sender gets used for every message and the data gets stored in a critical data area.
Special Settings	Via an extra random argument special settings can be activated. Nothing interesting gets distributed when activated, only a full 512byte message every second. The time can be changed.
Server only shares public(coordinates and owner) of forts. More information only with the owner of the fort.	The owner of the fort doesn‘t need private information. They know them already.
Clients send regurlarly alive messages and get disconnected if they don‘t	The server disconnects clients after a specific time, when they stop sending alive messages.