```java
public class Server extends UnicastRemoteObject implements BattleshipServer
{

    private Playground p1 = new Playground();
    private Playground p2 = new Playground();
    private boolean hostTurn = true;

    public Server() throws RemoteException {
        super();
    }
    public static void main(String[] args) throws RemoteException,
NotBoundException {
        BattleshipServer server = new Server();
        Registry registry = LocateRegistry.createRegistry(1099);
        registry.rebind("BattleshipServer", server);
        System.out.println("Server ready");
        server.game(true);
    }
    @Override
    public boolean getHostTurn(){
        return hostTurn;
    }
    @Override
    public void changeHostTurn(){
        hostTurn = !hostTurn;
    }
    @Override
    public String game(boolean host) throws RemoteException {
        Game game = new Game(this);
        game.game(host);
        return "Yes";
    }
    @Override
    public Playground getPlayground(int p) throws RemoteException {
        if(p==1) return p1;
        if(p==2) return p2;
        return null;
    }
    @Override
    public void sendPlayground(Playground playground, int p){
        if(p==1) p1 = playground;
        if(p==2) p2 = playground;
    }
}
```

```java
else if (turn == -1) {
        if (host && server.getHostTurn()) {
            server.sendPlayground(enemyPlayground, 1);
            server.changeHostTurn();
            turn--;
            System.out.println("Server playground kopiert");
        } else if(!(host || server.getHostTurn())) {
            server.sendPlayground(enemyPlayground, 2);
            server.changeHostTurn();
            turn--;
            System.out.println("Client playground kopiert");
        } else if(!host && server.getHostTurn()) System.out.println("Bitte
auf Server warten");
    } else if (turn == -2) {
        if (host && server.getHostTurn()) {
            try {
                playground =
playground.copyPlayground(server.getPlayground(2), false);
                server.changeHostTurn();
                System.out.println("Client playground auf Server kopiert");
            } catch (RemoteException ex) {
                ex.printStackTrace();
            }
            turn--;
        } else if(host && !server.getHostTurn()) System.out.println("Bitte
auf Client warten");
        else if (!(host || server.getHostTurn())) {
            try {
                playground =
playground.copyPlayground(server.getPlayground(1), false);
                server.changeHostTurn();
                System.out.println("Server playground auf Client kopiert");
            } catch (RemoteException ex) {
                ex.printStackTrace();
            }
            turn--;
        } else if(!host && server.getHostTurn()) System.out.println("Bitte
auf Server warten");
        playground.enabled(true);
    } else if (turn == -3) {
        if (host && server.getHostTurn()) {
            server.sendPlayground(playground, 2);
            try {
                enemyPlayground =
enemyPlayground.copyPlayground(server.getPlayground(1), true);
            } catch (RemoteException ex) {
                ex.printStackTrace();
            }
            server.changeHostTurn();
            playground.getPlayground()[finalI][finalJ].setEnabled(false);
            hit(finalI, finalJ, playground);
        } else if(host && !server.getHostTurn()) System.out.println("Bitte
auf Client warten");
        else if(!(host || server.getHostTurn())) {
            server.sendPlayground(playground, 1);
            try {
                enemyPlayground =
enemyPlayground.copyPlayground(server.getPlayground(2), true);
            } catch (RemoteException ex) {
                ex.printStackTrace();
            }
            server.changeHostTurn();
            playground.getPlayground()[finalI][finalJ].setEnabled(false);
            hit(finalI, finalJ, playground);
        } else if(!host && server.getHostTurn()) System.out.println("Bitte
auf Server warten");

        enemyPlayground.enabled(false);
        playground.enabled((true));

        if (playground.getShipList().isEmpty()) {
            turn = -1000;
            if(host) System.out.println("HOST GEWONNEN");
            if(!host) System.out.println("CLIENT GEWONNEN");
        }
    }
}
```

```java
public void clear(){
    for (JButton[] jButtons : playground) {
        for (JButton jButton : jButtons) {
            jButton.setBackground(waterColor);
            jButton.setEnabled(true);
        }
    }
    for(int i=this.shipList.size()-1;i>=0;i--){
        shipList.remove(i);
    }
}
public Playground copyPlayground(Playground playground, boolean duplicate)
{
    this.clear();
    for(int i=0;i<this.playground.length;i++) {
        for(int j=0;j<this.playground[i].length;j++) {
            if(!duplicate) {

if(playground.getPlayground()[i][j].getBackground().equals(shipColor)) {

this.getPlayground()[i][j].setBackground(playground.getPlayground()[i][j].g
etBackground());
                }
            } else
this.getPlayground()[i][j].setBackground(playground.getPlayground()[i][j].g
etBackground());
        }
    }
    for (int i=0;i<playground.getShipList().size();i++) {
        if(playground.getShipList().size()>0) {
            this.getShipList().add(playground.getShipList().get(i));
        }
    }
    return this;
}
public void enabled(boolean enable){
    for (JButton[] jButtons : this.playground) {
        for (JButton jButton : jButtons) {
            jButton.setEnabled(enable);
        }
    }
}
```