Multiplayer Design *Group 30*

The general idea

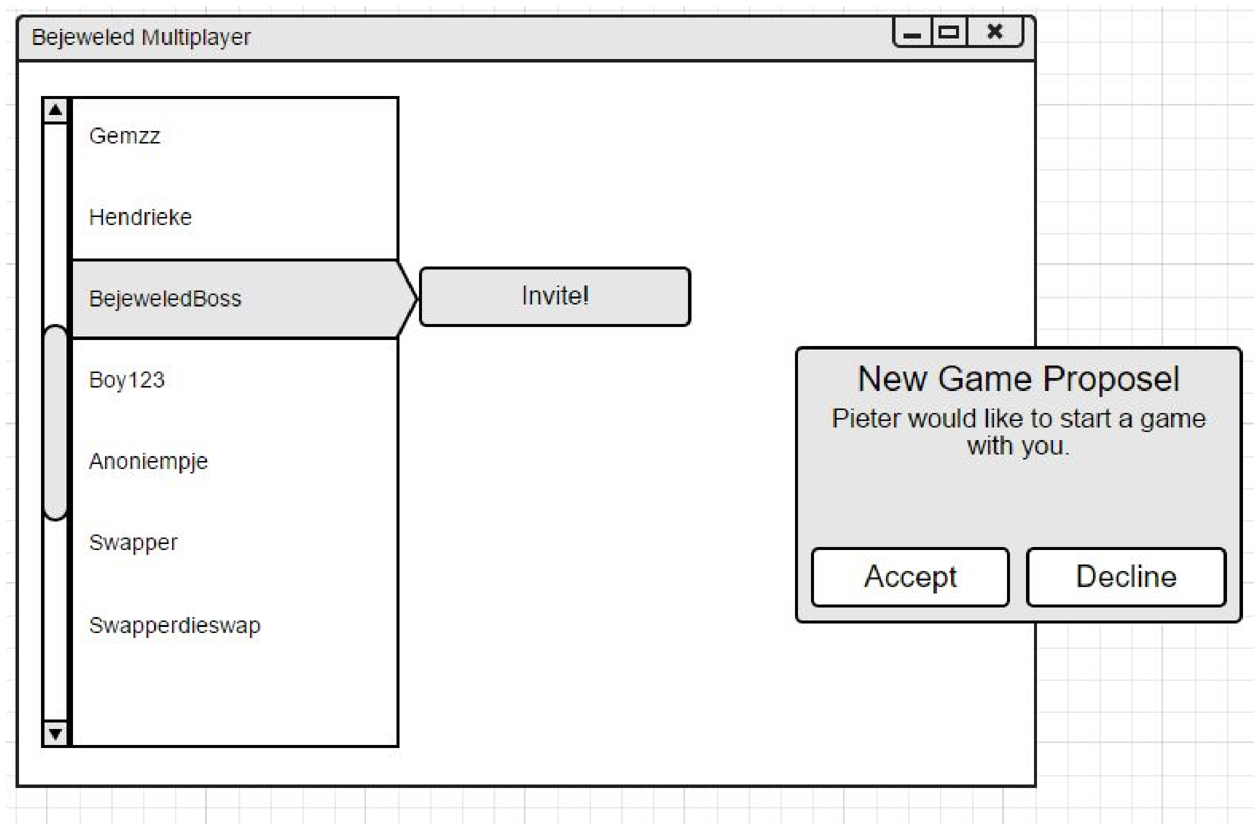
We'll use a central server to establish connections between players. So that there's a central lobby where players can find their opponents. When a game has started both players will run their own game and logic, but when a player decides to make a swap, the swap will be requested to the server, when the server approves the swap he'll send a message to both clients telling them which player made the swap and which gems got swapped. The players own game and logic will then handle the swap, removing combinations and putting new gems on the board, granting score to the player that requested the swap. This way we'll ensure that when two players make a swap at almost the same time, the server will decide fairly which request came first. When we would use a design where both players would communicate without a server, it would be hard to decide who made a swap first and there could be some unfair advantage.
To make sure that both games insert the same new gems, we'll use a seed for the random aspects of the game, the server will generate a seed at the start of a new game and send this to both players, that can then handle their own generation of the board. We'll use a fixed time for games and after a certain amount of time, or when there are no moves left, the final score of the players will decide who has won.
By letting both users do their own logic we can ensure that all the animations we have in the current version will still be performed smoothly, this way we can keep using most of our current code.

What do we need to change

Gui

The user interface of the board can stay the same for the most part, only the score of the opponent should be visible so we should add that to the screen. Also we wanna remove the hint button since this would be silly in a multiplayer version.
We have to create a lobby screen where the user can see a list of players, there should be an invite button to invite another player for a game. Also a user should get a pop-up when he gets invited with the ability to accept or decline a game invite.
We'll remove the main menu and create a screen where a user can pick a name and enter the lobby. Also the submenu for displaying the highscores should be changed, since these scores are all local highscores.

We need to add a menu for setting up the right name to use in the game as a username. You have to be able to save this, so you don't have to fill in your name again next time. This name is visible in the other player's lobby.



A mockup of what the lobby could look like

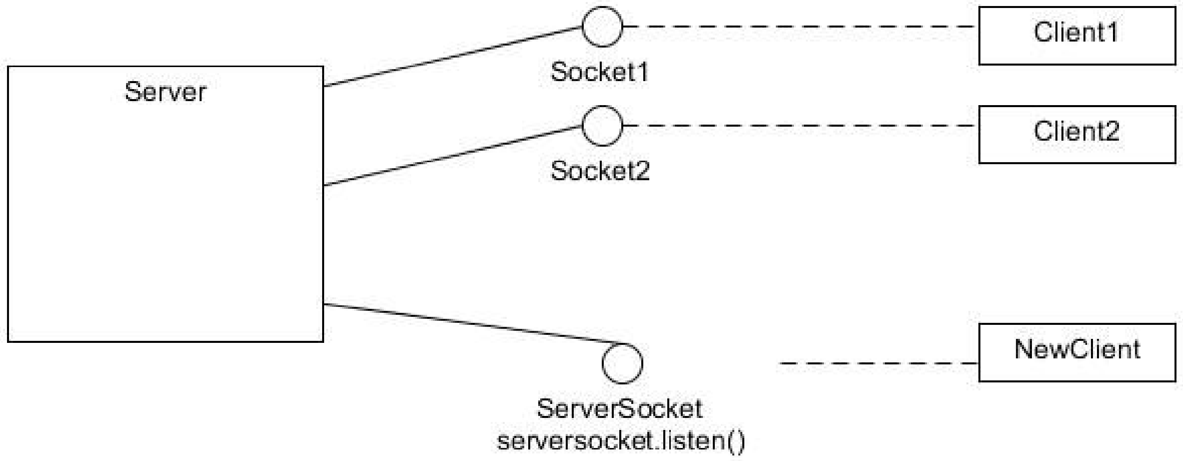
Logic

We'll have to change the generation of random gems to be able to handle a seed. This will be easy since all random generation is done in de GemType enum class, we only have to set the seed at the start of a new game after receiving it from the server.
We'll remove any time granted from swaps.
We'll need to keep an extra score of the other player and when we receive a new swap set a variable so we grant new score to the correct player.

Communication

We'll make use of socket communication over TCP, the Transmission Control Protocol ensures that messages get delivered. We're dealing here with swap messages, which are really important to get delivered, so TCP is our best option here.
An alternative would be to use RMI for the communication, but this seems to have a lot of overhead and since the message we're sending are fairly simple we decided to use sockets.

Our server will use a ServerSocket that listens for new connections. When a new connection is made, the server will start a new Thread and create a new Socket for communication with this particular client. It will remember the Socket together with the username of the user. By creating the new Socket for communication the server and client can send messages. Our clients will send a request for a socketconnection to the server when they start the lobby.

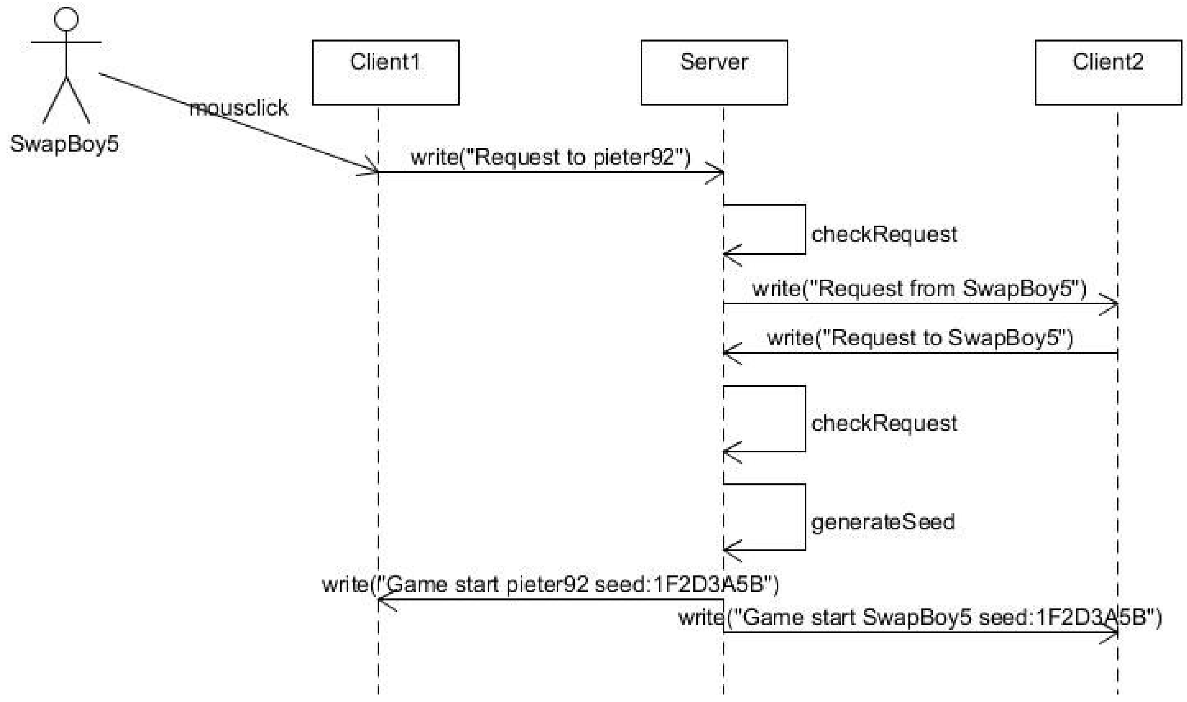


How the server handles multiple clients

Since we'll use sockets, we'll have to create a protocol for this. We'll need the following communication to be implemented:

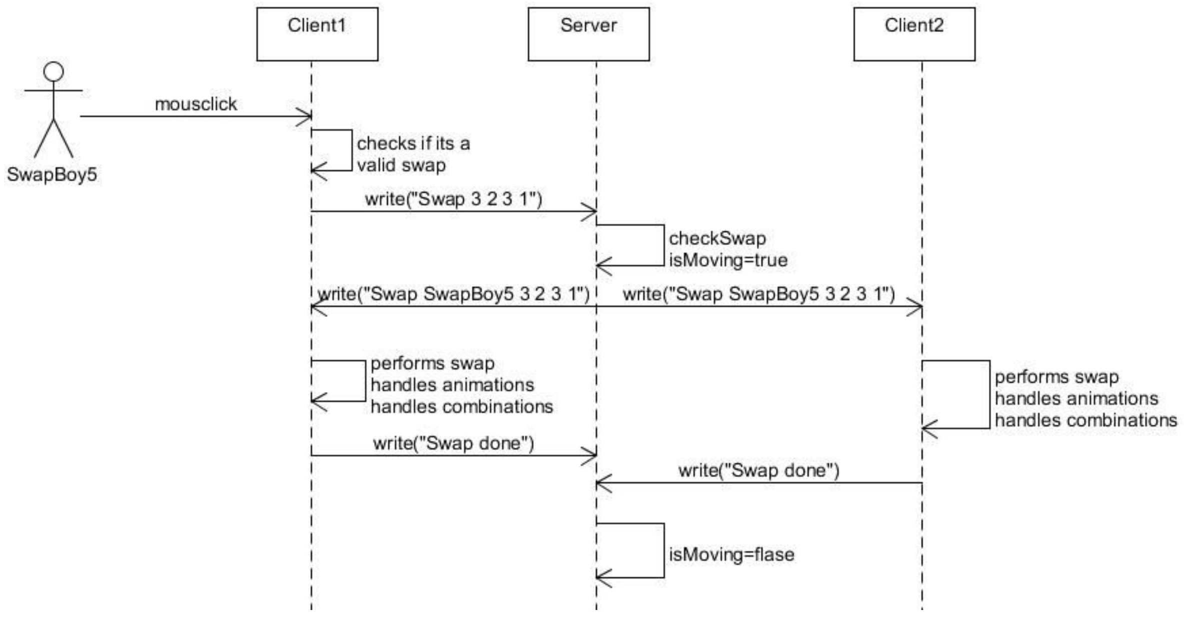
- Let a user connect to the server, the user should appear in the playerlist of other players and the other players should be able to see the user. Also when a user leaves the lobby he should be removed from the playerlist of other players.

- Let a user send a game request to another player, the other player should receive it and be able to respond. When two players sended a request a new game should be started.


SwapBoy5
mousclick
Client1
Server
Client2
write("Request to pieter92")
checkRequest
write("Request from SwapBoy5")
write("Request to SwapBoy5")
checkRequest
generateSeed
write("Game start pieter92 seed:1F2D3A5B")
write("Game start SwapBoy5 seed:1F2D3A5B")

The start of a new game

- During a game, the users should be able to send swaps, the server will send accepted swaps to both players.

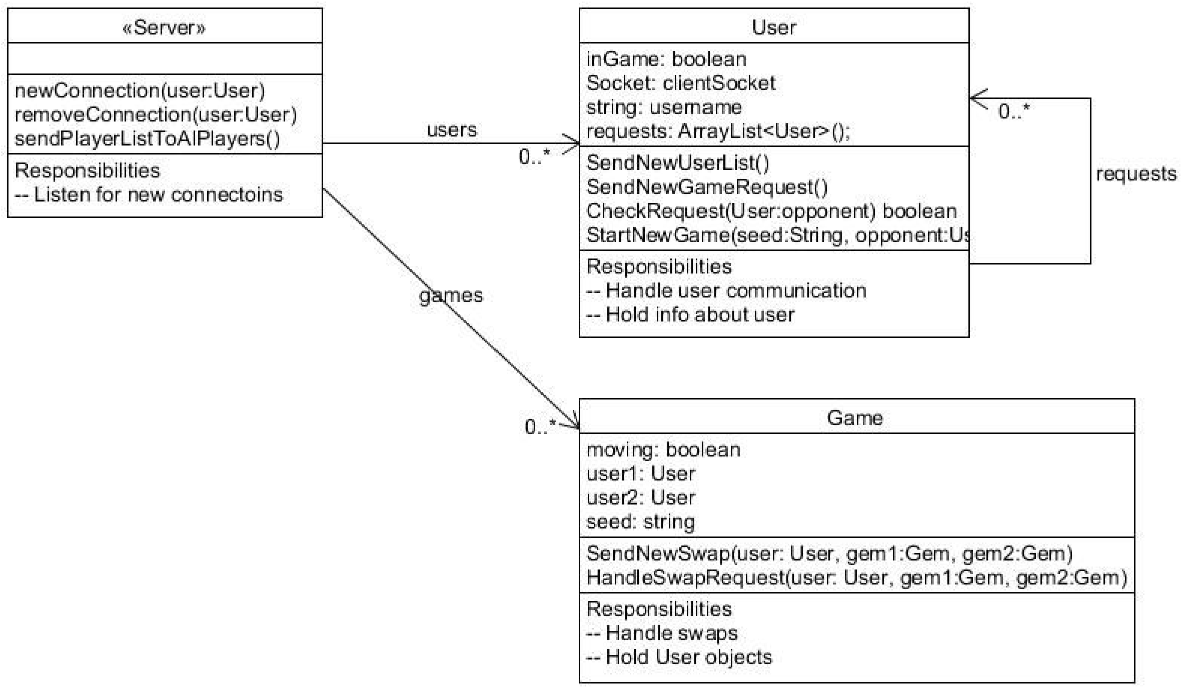


- A user should be able to send an end of game message to the server when time ends, when no moves are possible, when the user gives up and closes the game or when the connection is disrupted.

The server

The **Server** needs to hold some data. A playerlist, when a new users enters the lobby it will add the user to the list and resend the playerlist to all players. When a player leaves the lobby he will be deleted from the playerlist and the list will be resend to all players. The list will remember the player, the current connection and if the player is playing or not.
A gamelist, which will hold all current games. When the server gets a swap message from a user it will find the game the user is in in this list, check the current state to see if no swaps are in progress, and if correct send the swap to both players. When a game ends it will remove the game from the list.

For the server, of course, we'll need to create a separate application. Below is our class diagram for this application.



The class diagram for the server

The **User** class will remember if a user is in game. When a user is ingame he'll not appear in the playerlist that gets sent by the server. The User class will also remember requests, when a user requests another user for a game that user will get added the list. When a user accepts a request he'll just send another request, the request will check if the user is already in the request list. So when both users have each other in the request list a new game will start And the request lists will be emptied. The User class will also remember the Socket where to send messages.

The **Game** class will have a seed to make sure both randoms are equal, this seed will be sent with the user.startNewGame() which will send a message with the seed and opponent to a user. It will keep track if the gems are moving. So when a new swap request is received, it can simply check if another move is in progress, if not, it will sendNewSwap to both users. When users finish the move they'll send a empty swap request and the game will set moving to false.