# Inleiding(Eli)

Goeiemiddag allemaal

In verband met onze GIP maken wij een netwerkgame en 1 van de onderdelen van dit project is het maken van een werkend chatprogramma. Met dit chatprogramma zal je berichten kunnen versturen en ontvangen van anderen die ook verbonden zijn met onze chatserver. Eveneens zal u ook kunnen zien wie welk bericht heeft verstuurd aan de hand van een Nickname die je ingeeft bij het verbinden met de server.

Wij zullen onder andere uitleg geven over de belangrijke dingen die je zal nodig hebben bij het maken van dit chatprogramma. Met name TCP , multithreading , invoke en sockets.

# Communicatie over een netwerk(Eli)

Dus eerst het belangrijkste dat je nodig hebt voor het maken van een chatprogramma is Communicatie over een netwerk. Zonder dit kan je onmogelijk berichten verzenden en ontvangen. Dit onderdeel kan je regelen op 2 manieren. Sockets en TCP , met deze 2 manieren kan je hetzelfde bereiken. Maar het enige verschil is dat TCP werkt op Layer 4 van het OSI-model terwijl Sockets werkt met IP-adressen en poorten en dus op Layer 3. Je gaat dus iets meer stappen moeten ondergaan bij sockets voor hetzelfde resultaat. Wij gebruiken voor onze oefening TCP en hier gaat rob nu meer uitleg over geven.

# TCP(Rob)

TCP of ***Transmission******Control******Protocol*** is een connectie-georiënteerd protocol dat veel gebruikt wordt voor het maken van een netwerkverbinding. Dit is echter slechts 1 manier waarop dit mogelijk is. De andere manier is UDP of ***User******Datagram******Protocol***. UDP is veel sneller dan TCP maar gaat de connectie tussen ***sender*** en ***reciever***niet controleren waardoor er data verloren kan gaan. Dit wordt gebruikt bij livestreams of ***First******Person******Shooter*****games**.

De primaire gedachten bij UDP is dus gegevens snel krijgen en bij TCP de garantie dat alle gegevens aankomen. Daarom gebruiken wij dus het principe TCP omdat dit ons de garantie geeft dat onze gegevens aankomen bij de ***server*** of ***client****.*

Omdat TCP zich op de transportlaag bevindt van het netwerkmodel moet er eerst een connectie gemaakt worden tussen een zender en ontvanger. TCP heeft hiervoor de ***Three******way******handshake***

De ***Three way handshake***zorgt voor een verbinding tussen een TCP-client en een TCP-server. De TCP-client stuurt een Pakket met een SYN-vlag (***synchronize***) naar de server. Als de server de verbinding accepteert, dan stuurt de server een pakket terug met zowel een SYN-vlag als een ACK-vlag (***acknowledge***). Als de client dit accepteert dan stuurt die vervolgens ook een pakket met een ACK-vlag naar de server. Bij het sluiten van de verbinding stuurt de server een pakket naar de client met een FIN-vlag (***final***) waarna de client een ACK-vlag terugstuurt en omgekeerd. Hierdoor zijn zowel de server als de **client** is nu op de hoogte van het sluiten van de verbinding.

# Hoe gebruiken we TCP? (Seppe)

Nu we de uitleg hebben gegeven over TCP, moet je nu weten hoe je het gebruikt.

Hier is een belangrijk datatype die je zeker nodig hebt:

* TCPClient, zorgt voor een client connectie voor TCP netwerk diensten

Dan hebben we ook enkele methodes die je nodig zal hebben:

* Close() : zorgt ervoor dat de TCP verbindingen gesloten wordt , dit doe je meestal als een client zich wil verwijderen van het chatprogramma. Dan wordt de verbinding gesloten
* GetStream() : geeft de netwerkstroom terug die wordt gebruikt bij het versturen en ontvangen van data
* **Connect**() **:** maakt een verbinding met een server of host door het IP-adres en de poort in te geven.
* **StreamReader** : dit datatype leest karakters van een byte stream ( bv. Networkstream)

Het krijgt als parameter een datastroom waarvan hij de data kan lezen.

Hierbij zijn er ook enkele datatypes

* **Readline**() **:** leest de data van een stream en geeft deze data terug als string , hiermee kunnen we bv. Inkomende berichten onderscheppen en lezen.
* **StreamWriter :** dit datatype schrijft data weg op een stream. Het krijgt als parameter de stream waarna hij de data moet wegschrijven.
* **WriteLine**() : schrijft data weg naar de stream me daarachter een regel terminator(scheiding tussen verschillende regels) dit is om verschillende berichten te onderscheiden van elkaar.
* **Flush**() : spoelt alle data door van de *Buffer* naar de datastream, de data kan dan gelezen worden door *streamreaders.*

Zonder deze methode zal de ontvanger de data nooit kunnen ontvangen.

# Sockets (Simon)

In onze oefening gebruiken we TCP-clients om ervoor te zorgen dat we kunnen communiceren met andere computerprocessen. Maar er zijn ook andere mogelijkheden om dit te doen die wij niet gebruiken, namelijk **sockets**. Een socket is een communicatie eindpunt in een netwerk dat op een internprotocol gebaseerd is. Elke socket wordt geïdentificeerd aan de hand van een combinatie van het IP-adres en de poortnummer. Deze combinatie wordt ook wel socketadres genoemd. Aan de serverzijde worden hiervoor voor gedefinieerde poortnummers gebruikt. De **client** kiest zelf een willekeurig poortnummer (boven 1023) dat nog niet in gebruik is.

Dan afbeelding met de uitleg erbij

* **Socket**() : maakt een **socket**
* **Bind**() : koppelt een socket aan een IP-adres en poort nummer
* **Listen**() : Luistert naar inkomende Connecties
* **Accept**() : accepteert een connectie , deze methode stopt tot er een connectie is gemaakt
* **Connect**() : Connecteert de **socket** van de **Client** met de **server**
* **Send**() : verstuurt data over een verbinding
* **Recv**() : ontvangt data van een verbinding
* **Close**() : Sluit de verbinding

# Multithreading (Eli)

Een andere belangrijk deel bij het maken van een netwerkgame en chatprogramma is multithreading. Multithreading zorgt er namelijk voor dat onze server gaat blijven luisteren naar binnenkomende verzoeken van clients en gelijktijdig de binnenkomende berichten van de al geconnecteerde clients gaat afhandelen.

Maar wat is multithreading nu eigenlijk

Bij multithreading wordt een taak verdeeld in kleinere delen die we threads noemen. Deze kunnen dan bijvoorbeeld tegelijkertijd uitgevoerd worden. Zoals bij ons voorbeeld met de server. Onze luister thread die luister thread die gaat luisteren naar inkomende berichten gaat pas stoppen wanneer de server stopt. Andere threads kunnen dan weer veel korter duren maar steeds opnieuw worden gebruikt.

De taken zoals luisteren , doorsturen van berichten enz. worden dan Threads genoemd.

# Waarvoor gebruiken we threads? (Seppe)

Wij gaan threads gebruiken in de server voor het luisteren naar inkomende berichten van de cliënts en het doorsturen van de berichten van de cliënt naar alle andere verbonden cliënts. Ook gebruiken we een thread op de cliënt om te kunnen luisteren naar de server.

# Hoe gebruiken we multithreading? (Seppe)

Eerst en vooral maken we een veld die je als thread declareert

Vervolgens maak je een nieuwe thread aan die ervoor zorgt dat een sub wordt uitgevoerd.

Hier zie je dat ConnectClient de sub is die wordt uitgevoerd

Daarna zorg je ervoor dat de thread start.

In de server zit er niet veel tijd tussen het opstarten en het luisteren naar de clients, als er een bericht zou toekomen tussen die periode, kan je code niet verder, om dit te vermijden gebruiken we thread.sleep, die zorgt ervoor dat de code een aantal milliseconden wacht voor hij verder gaat met de code.

# Invoke (Simon)

***Invoke*** is een functie die we gaan gebruiken wanneer we gebruik maken van meerdere ***threads***. ***Invoke*** kunnen we gebruiken als een thread niet de andere thread nodig heeft.

We gebruiken de functie **BeginInvoke**() in plaats van **invoke**() omdat ***invoke*** een ***deathlock*** kan veroorzaken. Ook moeten we deze gebruiken bij seriële poorten bij de **eventhandler** omdat de *thread* anders niet direct update.

# Slot (Rob)

Nu mogen jullie starten met de oefening. Als oefening gaan jullie een chatprogramma maken waarmee jullie naar elkaar berichten kunnen sturen. Jullie kunnen deze oefening aan de hand van de cursus maken of a.d.h.v. het filmpje dat in de klasmap. In deze klas map vinden jullie ook een project met daarin al het formulier zodat jullie alleen maar de code moete schrijven. Dan mogen jullie nu starten en als er nog vragen zijn kunnen jullie die altijd stellen.