# Minor Skilled Analysis

Pieter Schreurs 318775

Contents

[Minor Skilled Analysis 1](#_Toc62502110)

[Introduction 3](#_Toc62502111)

[Start Minor 4](#_Toc62502112)

[Expectations 4](#_Toc62502113)

[Learning Goals 4](#_Toc62502114)

[Setting up the project 4](#_Toc62502115)

[Phase 1 - Analysis 5](#_Toc62502116)

[Getting started 5](#_Toc62502117)

[Planning 5](#_Toc62502118)

[Progress 5](#_Toc62502119)

[Obstacles 6](#_Toc62502120)

[Feedback 6](#_Toc62502121)

[Phase 2 – Design/Prototype 7](#_Toc62502122)

[Planning 7](#_Toc62502123)

[Progress 7](#_Toc62502124)

[Obstacles 8](#_Toc62502125)

[Feedback 9](#_Toc62502126)

[Phase 3 – Production 10](#_Toc62502127)

[Planning 10](#_Toc62502128)

[Progress 10](#_Toc62502129)

[Explaining the set-up 12](#_Toc62502130)

[Obstacles 13](#_Toc62502131)

[Feedback 13](#_Toc62502132)

[Phase 4 – Production 14](#_Toc62502133)

[Planning 14](#_Toc62502134)

[Progress 14](#_Toc62502135)

[Obstacles 16](#_Toc62502136)

[Feedback 16](#_Toc62502137)

[The Product 16](#_Toc62502138)

[Self Reflection 17](#_Toc62502139)

[Improvements 17](#_Toc62502140)

[Future Perspectives 17](#_Toc62502141)

# Introduction

The minor skilled is about developing yourself to be a more knowledgeable about a certain topic.  
I took this time to increase my knowledge of multiplayer games and how they work, and eventually making a multiplayer game myself.

The reason why I chose this is because we’ve had a networking course for my study, and it appealed to me. The big thing you could’ve done with is, is to create a game in the Project that came after that course. Unfortunately, we as a group didn’t chose this approach. We made a single player game.

My biggest reason that I would have liked to make a multiplayer game is because that brings new challenges I’d like to have figured out. Plus, most of the games, practically all AAA, are played in multiplayer. It would not be a big overstatement to say that if you want to develop at a successful game company you would require the tools to make a multiplayer.

In this report I will describe my journey, my choices and following results which I had in my Minor.

# Start Minor

## Expectations

My expectations for this project were to make a functional multiplayer game.  
I chose to make a RTS game first of all because I really like that sort of games and played it a lot in my youth.  
My second motivation is because it looks like a big enough project that it can be a real challenge for me. I never made such a big project on my own and I’m curious whether or not I can make it work.  
It will require some good planning and thinking before programming which in my opinion we haven’t really had in most project groups. I’m looking forward to seeing what comes in my path during development.

## Learning Goals

I set up the following learning goals for myself:

Learning how to make a correctly functioning multiplayer game.

This because the multiplayer part is the functional part of the thing I’m making. Actually being to showcase it is quite important.

Learning how to organize a bigger well coded game on my own.

Having an organized base is important to a project, if you were to make some changes or re-use some code you can easily identify what parts you need or edit. Plus being able to document your code is a valuable skill that’s often overlooked.

## Setting up the project

I made my project in Visual Studio and Unity. Using GitHub for version control in combination with SourceTree. I have 3 directories for my code. One is the Client which is made for Unity. Another one is a C# Console application which I use as a server, and the last one is a Shared repository which is a code library which is used in both directories to set up communication protocols between the projects.

# Phase 1 - Analysis

## Getting started

The first few weeks I spent most time analyzing on what to use in my game.  
Is the goal to make a RTS game or is the goal to learn how to make a multiplayer system.  
I carefully thought out the learning goals of my Minor and came to the conclusion that I want to make a system I can use in other games as well. If I ever wanted to make a project a multiplayer game.

I didn’t want to be dependent on a built-in Unity helper or plugin such as Photon.  
I wanted to create a skill for myself so that I could use it multiple ways instead of just focusing for a game in the Unity engine.

In order to achieve this, I made a planning for myself.

## Planning

My planning for the first phase is quite simple:

* Decide on what network protocol I want to use.
* Look up some projects on GitHub/YouTube tutorials on how to make a multiplayer game.
* Flesh out my game idea more, think of all the requirements of a RTS.
* Make a class diagram in which is visible how data will be handled.
* Re-visit the Networking lessons we’ve had to freshen up my knowledge.

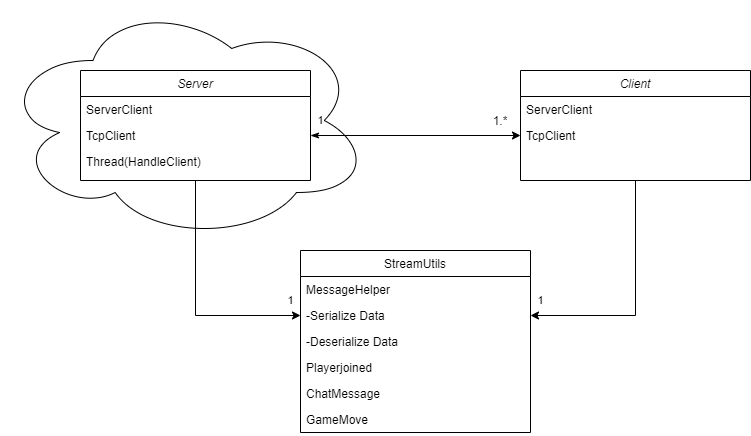
## Progress

The first subject in my planning was deciding on a network protocol.  
I chose to go for TCP, this because TCP is more reliable and makes sure to get the data over in a correct way.  
The game is not meant to be a massive player game in where it doesn’t matter whether or not data is correctly displayed. With designing it in mind I have a max of 4 players per game.

The second subject was looking up tutorials on GitHub/YouTube was somewhat fruitful. However most of the projects I noticed were using Unet/Photon instead of a custom networking code. They however still had some information on how the project should be structured and how the client should handle things.

I went through the networking courses again, re-did the lab assignments and the labs. This gave me enough insight on how the standard TCPclient can write and read data across the network.

Using this method I came up with a small visualization of how I thought the networking part is supposed to be handled, visible in this picture below.



There would be one server running which could receive multiple clients.  
Both the server and the client would have access to the StreamUtils class. This is a class which could “translate” the packets which are sent over the network. For example, a client would send a chat message, the StreamUtils class would translate that chatmessage to bytes which could then be read by the server and distributed to the rest of the clients.

## Obstacles

I didn’t have a lot of obstacles during this phase, mostly because it was all research based and repeating of already laid out assignments.

It was mostly thinking out ideas of how it should work instead of testing the practical side of it.

## Feedback

The big feedback I received was that my initial idea to make a RTS game is somewhat too ambitious.  
I made a big document on researching what components should be part of a RTS, did some market research on RTS. I should put less emphasis on the making of the game, but more of the multiplayer side of things. Take my learning goals and mold them into the game form you can use to support your learning goals instead of making a RTS.

I came up with a new idea:   
Make a lobby system in which people can join and play different games with each other.   
The emphasis is in the multiplayer system, not the creation of the game.

# Phase 2 – Design/Prototype

## Planning

The planning for this phase concluded of a few things:

* Set up User stories for the game I was going to make.
* Make a flowchart to show how data is handled.
* Make a prototype in which I showcase the way network protocols are handled.

## Progress

Using the feedback I received from the first phase, I came up with a small prototype server in which multiple people could connect and send messages.

I had to write this prototype quickly so that I quickly could find out what’s possible in terms of network code so that I could make my game.

The server is just a C# console application which runs and continuously listens for incoming connecting clients and puts them in a separate thread once connected.

Each message the client can send via the StreamUtil class.  
The way this StreamUtil class works is the following:

Each “command” be it a request or sending of a message is a separate class.  
Each class can be serialized or deserialized in which it details how to convert it to something which can be sent over the TCPnetwork, or be read when it’s incoming from the TCPnetwork.   
These classes all have their own variables and an identifier.

Each command is built up as the code shown on the next page.

You basically have a messagehelper which is called whenever a new message is received, it will read the first int so it can identify what kind of message it received, be it a Playerjoined, chatmessage or gamemove etc. Once it’s been identified what kind of message it is, it is then further deserialized. Since it can be parsed as a class instead of a row of bytes.

This method was enough to make a prototype working of a lobby system in which people can send chat messages to eachother.



## Obstacles

The big obstacle came once I completed my main goal of having a lobby system. The objects I could send to over the network were quite basic. For example the list of players I would need to send an array/list of objects, with the current implementation I couldn’t do this. There is a work around to use wildcard characters but it wouldn’t work if a user would use such a wildcard for his username for example.

I was stuck too long on how to make this work before I asked someone more experienced for some advice. I spent a long time on debugging, unfortunately debugging network code is a hard and tedious process. I should’ve asked earlier for help.

## Feedback

After the prototype was stuck on a point I couldn’t make it work. I had a talk with Hans Wichman. I knew that he was capable of helping me out, because he also gave the course for networking.

He said he also encountered the problem previously as he was also making a multiplayer system himself.  
He then referred me to a new and updated assignment from the next Networking course.

He understood what I went for in my initial idea’s and prototype, but said he had some ideas on how to fix the issues I had. They were implemented in a course assignment he gave me.

I will explain in the next phase what I did with that project which would eventually serve as the base as my final product.

# Phase 3 – Production

## Planning

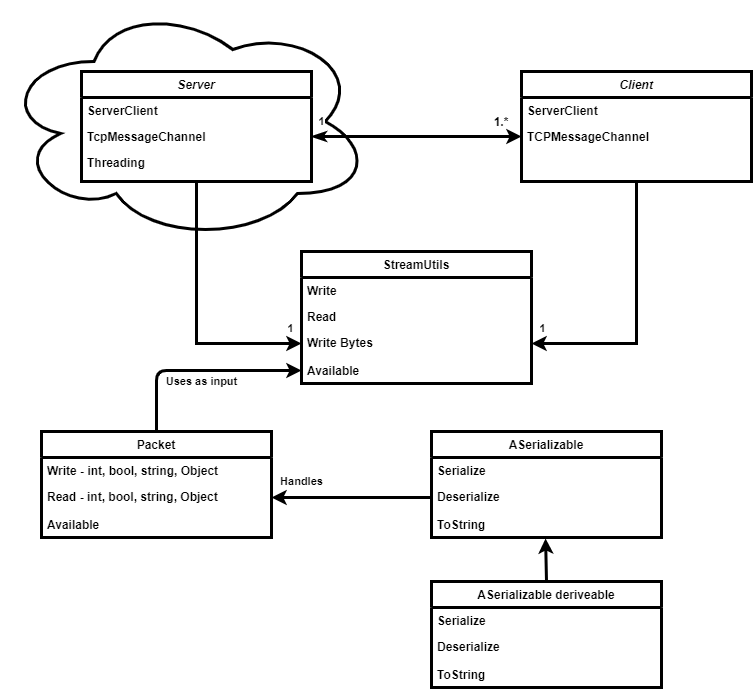
Now that I have a new project in which I can experiment I made a planning for myself.

* Make a turn based game working.
* Optimize and make the code working I received from Hans.

## Progress

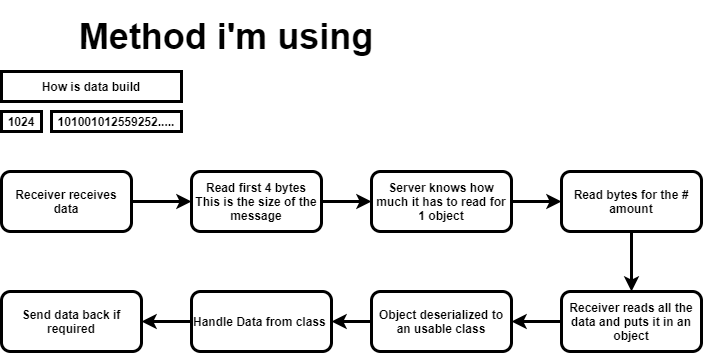
Having the code I received from Hans was a big help. It was actually the new Networking course final assignment with a lot of groundwork already laid out.

It was good to see there were a lot of identical ideas that he also implemented in the assignment, which I already had in my prototype. The basic concept worked the same as my prototype as seen in this class diagram



The biggest difference between my initial setup and the one I’m given is that it can send packets.  
Packets are made up out of classes which inherit from ASerializable. I can now send an entire class, I just have to put the parts which have to be serialized in the method Serialize. It gives a much more clear view of the actual data which has to be sent.

Another big improvement I had thought about which solved the issue of not being able to send lists/dictionaries/arrays is improving the way reading and writing messages happen.  
The writing of an object now happens in 2 stages; stage 1 is sending a 4 byte size int in which it displays how large the size is of the next incoming object. That way it reads until the full object is read and can be deserialized as an object on the connected client. As seen in this flowchart below.



The project I received had some very useful Debugging tools, it had an “in-game” debugging tool for Unity and a nicely working Console output printer for the server. This was particularly useful because debugging for Network messaging is quite hard to do. It also has simple State Machine for switching scenes in Unity and an example on how to send data. The rest I had to program myself, which I didn’t mind.

## Explaining the set-up

At first you have to somewhat “decipher” the code you are given so you actually know what’s going on and understand everything, after that I could go and implement my own fixes.

In the shared section of the projects data is set-up in the following way:  
**Model:**The models are classes which inherit from ASerializable, they contain data which needs to be sent over the network. For example: BoardData, this contains the board in data. This way we have an object we can serialize to clients and send them via a Protocol.

**Protocol:**The protocols are classes which also inherit from Aserializable. They however are small “commands” to the server with just one purpose. They are in turn received by the client/server and handled accordingly.

First I made sure the people who joined the lobby could send messages to the server and the server would then send that information to the clients. The chatting mechanic is done.  
Then I fixed the correctly displaying a chat message if a new user joined the channel.

After that I could work on the game mechanics and making sure people can actually play the game.

The server has a Data object which represents the board of the tic-tac-toe board, it’s basically an array of integers in which each index is a number to represent the value on the board.  
I implemented a check to see if there is a winner, with simple loops to check whether there are 3 indexes lining up. Once a player has won, the server will send a message to the players to inform them someone has won.

Then I made a simple indicator which showed which players turn it is, I made a new protocol for it and had some labels which should display text. The server then in turn held track whose turn it was and sent the right protocol accordingly.

After that I made a quit button so players could return to the lobby after they finished their game or even during.

## Obstacles

I had some difficulties making the quit back to lobby system working.  
The player can switch his view back to the lobby, but the server still thinks he’s in the game thus receiving and sending protocols can be hard.

It felt like I haven’t made enough progress during the weeks working on this phase. It was a lot harder than expected. Nothing really worked the first time and debugging was tedious.

## Feedback

This time I actually had a “guest presentation” with Paul Bonsma. This was a good idea since my peer group doesn’t really have programmers with whom I can go in-depth in to the stuff I’m working on.   
They seemed to understand my struggle of not feeling that you advance in a quick enough fashion.   
 Paul basically also had to work a long time to set up a simple networking project. That felt reassuring for me.

With a couple of others also having had experience with networking projects they gave me a good tip:   
See if you can open up external internet connections, for now I’ve only tested this setup on local network. Most people found some difficulties with this.

They also gave me idea’s on multiple games to showcase my API. Afterall it’s my goal to make a functional network system and the games are just for showcasing that it works.

# Phase 4 – Production

## Planning

Now that the Turn based game is done and documented it is time for the last step in production.  
This final phase my planning consists of a couple of things:

* Editing the code so I can have multiple game modes.
* Make an external connection work.
* Making a real time game mode.
* Document every function in the code properly.
* Bug fixing.

## Progress

I started with doing some research on how to allow external connections to be made.  
I had to log into my router and fiddle with some settings.  
First I had to open up the port forwarding to allow for external users to connect to my server I have running on my PC. After that I had to make sure my PC was running on a static IP so I could edit my code so that the code automatically pointed to the server.

I tested it with my mobile internet and a laptop, and it all worked out.   
There was also the option of hiring a domain name and let the server run there. That’s something I might want to do if the product is something more people want to use. It’s just impractical if you’re developing it right now.

After that I started with managing how to put more game modes in the program. I had to put a small dropdown box which the client fills with a request from the server for the available game modes.  
Once the game mode is chosen, the client can ready up and once there’s enough people available to play the game, the clients will get forwarded to the game itself.

The tools to make a turn-based game is already implemented, I received some tips on what kind of game I could also make with this, but instead I found it interesting to see whether a real time game would be possible with this method.

So I found a simple pong example on the internet. It’s a simple single player game so my goal was to transform it into a working multiplayer game.

To make it work, I had to do a few steps.

First I had to update the paddle positions to simulate the game between two players.

The way I envisioned first was just a simple protocol based in which the player gives an input in the form of a PlayerInput protocol. The server then in turn sends to everyone which player gave the input and what input it is in the form of a PlayerInputResult protocol. The PlayerInputResult converts an input command to a velocity vector and in turn sends it. Each client will then apply that velocity to the according paddle.

I tested this setup with a few play testers and it worked, however there was some de-sync issues. This happened because there was a small delay in between people sending and receiving the velocities.

I had to create a databoard which represents the pong board as data.   
This has the paddle positions and ball position. This is then serialized into sendable data which the client can read out and apply to their local game.

The way a player interacts with the server is to send a protocol called PlayerInput, this has a value of which input is sent. The server then in turn handles the input received from the client and updates the position of the paddles.

This was once again tested and worked out much better. It actually started to look like a game now.  
However there was still a small delay in between sending and receiving data.

I solved this by having the data board continuously broadcasting it’s state to the connected client, and reducing a Thread.Sleep function so it runs now on 60 ticks per second. It plays a lot smoother.

As my code I was nearly done with my goals from my planning I had a few things left to do.  
One was properly documenting my code, so I added more comments to the important bits of the assignment.

## Obstacles

In this phase I had a little amount of obstacles.

However one big obstacle I had is trying to simulate the physics engine in unity without using unity.  
It is needed in order to get the ball moving on the board data, updating it’s position and velocity, seeing whether or not it has collided with something.   
There might also be a solution in where the “host” (the player with playerID 0) is responsible for detecting ball positions and colliding with paddles/walls.   
This method however is more prone to being not implemented correctly since a client can determine whether scoring has happened which is not ideal in a multiplayer game.

Normally you’d have all the logic code in the server without the player being able to access it. This of course is very exploit sensitive.

## Feedback

The feedback I received from my play testers is that my project okay, it has a few bugs and didn’t work optimally on some points. But the project did what I had to do.

From my peer group I received feedback that it all looked cool, and during the update presentations I received some compliments that I could explain my progress well to non-programmers.

I was advised on how to hold my final presentation by Bram. He advised me to use Doxygen, this is a program which creates a HTML document from your code in which it shows all your methods with comments. This is especially useful if you have the learning goal “How to properly document your project” like I have.

# The Product

As my final product I have a working lobby system in which people can chat and start games which are loaded from the server against each other. The games I have implemented are Tic-Tac-Toe and Pong.  
But most importantly I have made a system work in which I can easily add game modes and add protocols to make these game modes playable.

Unfortunately, there are still some bugs but there I haven’t had the time to fix these yet.

Although I feel like the games aren’t the flashiest examples of what this can do, it definitely portrays the thought and effort that went into this Minor. It serves as a good base of a mini game lobby I can play with friends during this Corona Period.

# Self Reflection

## Improvements

There are a lot of improvements I can make, not only to the project but also on my work ethic.

My working times were irregular, there were some days I worked a lower amount of hours and some days where I had to catch up on that lost progression. Some more systematic order would be better.   
I also need to create a document (instead of using SourceTree) to really document my workflow, things I’ve done that day and what kind of issues I ran into. I mostly focused on coding instead of these peripheral looking matters. In hindsight these are not peripheral but are quite essential.

I noticed as the weeks progressed my motivation also increased and I got more work done during the same amount of hours.

Unfortunately I haven’t had a good method of tracking the exact amount of time I invested in the project. However my working times were 9:30 till 17:30. Most tasks I planned in for myself took a lot longer than I had initially planned.

I really liked the weekly update we had with a peer group, it sometimes pushed me the extra way to showcase something new for the presentation. I really felt that having such a group helped with the Minor.

## Future Perspectives

There are a couple of games I’d like to personally make which I enjoy myself.  
The minor product I made gives a good infrastructure to make new game modes.

One I’m actually requested to make by some friends is The Resistance Avalon board game.  
Once I’ve completed that we can play this together.

One game I actually still want to make is a bomberman game, it would look like a fun challenge to implement this.

On the organizing part of big projects I learned that I have to document a lot more.  
I’ve used GitHub for my logbook but unfortunately I haven’t updated it daily with the progress I’ve made that day.

# 