­­

Concurrent Network Applications report

ANTHONY STURDY

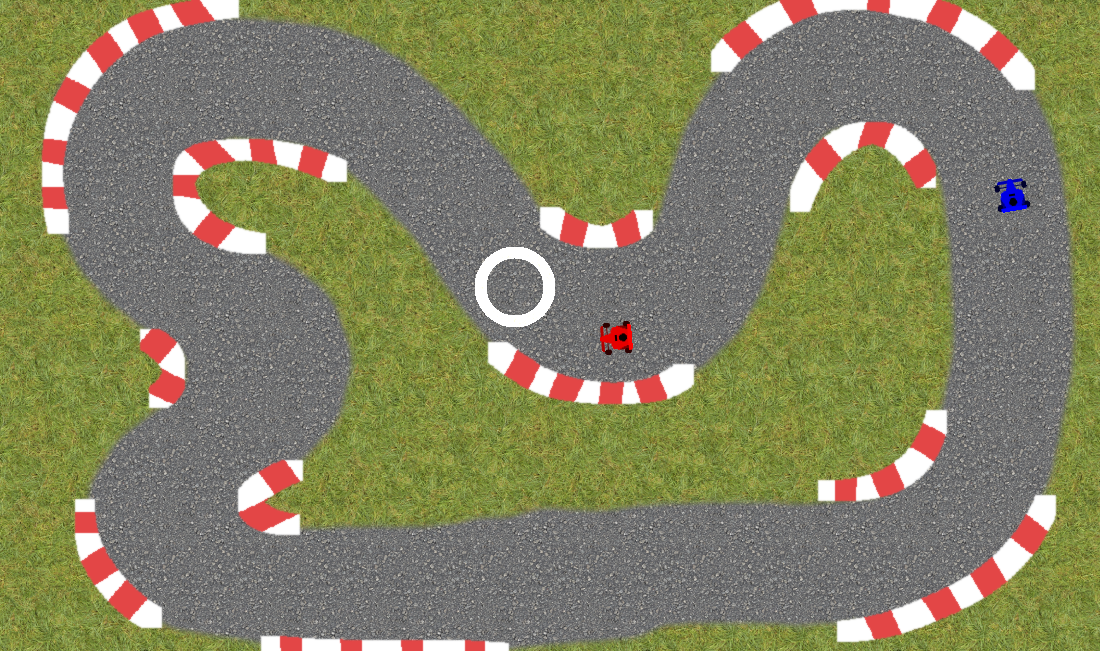


Table of Contents

[User Guide 2](#_Toc26969365)

[Design 2](#_Toc26969366)

[Functionality/Concepts Implemented 2](#_Toc26969367)

[Concurrency Network Diagram 2](#_Toc26969368)

[Class 2](#_Toc26969369)

[Sequence 2](#_Toc26969370)

[State 2](#_Toc26969371)

[Activity 2](#_Toc26969372)

[Explanation of Strategy 3](#_Toc26969373)

[Critical Evaluation 3](#_Toc26969374)

[What I have Learned 3](#_Toc26969375)

[What went well / what went wrong 3](#_Toc26969376)

[Challenges I have overcome 3](#_Toc26969377)

[What I would do differently 3](#_Toc26969378)

[References 3](#_Toc26969379)

# User Guide

**Change Profile Picture**

Before connecting to a server, click the Profile Picture icon to the left of the Username input box. Select an image, then click Open. When you connect to the server, the profile picture will be applied.

**Connect to a server**

Enter server details in IP Address and Port input fields, ensure you have entered a username (custom profile picture not required), then press the Connect button.

**Send message**

Type your message into the large input field at the bottom of the application, then press the Send button.

**Send image**

Press the button with a file icon on it, next to the message input field. Select an image you want to send, then select Open.

**Send direct message**

Double click a use you would like to direct message from the User list on the left of the application, the username will appear in the message input field (e.g. @Anthony), type your message after this.

**Help command**

Type ‘/help’ into the message input field to get help from within the application.

**Challenge user to a racing game**

In the message input field, type ‘/game’ followed by the username of the use you would like to challenge (e.g. ‘/game Anthony’)

**How to play racing game**

Use the arrow keys to control your car, drive through the circle checkpoints, the first person to complete three laps of the circuit will win which is then output to the chat.

# Design

### Functionality/Concepts Implemented

* TCP Sending (Server and Client)
* UDP Sending (Server and Client)
* Nickname System (Stored on server)
* Customisable Server Details
* Profile Picture
* Clients List
* TCP Message Sending
* TCP Direct Messaging
* TCP Image Sending
* Command System (/help and /game)
* Two player 2D racing game using MonoGame

### Activity

**Server**

*(Figure 1).* This concurrency diagram of the server demonstrates how data flows through the application and how different threads are utilised to prevent the application from locking. Upon a client joining the server, a new thread is created which handles TCP packet reading for that client. The TCP read function uses a while loop which would lock up the whole server if it were on the main thread, having it on its own thread only locks that thread until another packet is received. The TCP thread is also responsible for creating another thread to handle UDP packet reading for the client.

**Client**

*(Figure 2).* This concurrency diagram of the client application demonstrates how data flows through and is processed by the client application. The application starts by first creating a windows form, which runs on its own thread. The windows form then calls a connect function upon button press, the connect function, similarly to the server application, starts a new thread to handle the TCP reading for the client. The function contains a while loop, which causes that thread to lock up when waiting for a packet but doesn’t lock up the entire application, due to it running its own thread. The TCP function is also responsible for creating the UDP reader thread upon receiving a UDP login packet. When a player receives a game start packet, a windows form is created which is run on a new thread. This is responsible for handling the MonoGame application and processing player information packets it receives, so each client knows where other clients are and what checkpoint they’re currently aiming for.

### Class

**Server**

*(Figure 3).* In this UML Class Diagram for the server application, you can there are 3 classes being used. The SimpleServer class is the main class, which then references both the Client and Game class. The Game class also references the SimpleServer class as it stores an instance of class to send packets. The Client class doesn’t reference either of the classes but is referenced by both, the client is responsible for handling the server’s connection to a client, for each client who joins the server, another Client is instantiated.

**Client**

*(Figure 4).* This UML Class Diagram for the client applications shows how the SimpleClient class references both the ClientForm and DrawTest class. The DrawTest class is what MonoGame uses to display the game. The ClientForm class is the form which is displayed for the chat room features. It includes the functionality for buttons and other GUI elements.

# Explanation of Strategy

# Critical Evaluation

TODO: REMOVE THE SECTIONS THIS SHOULD BE ONE BIG PARAGRAPH (600 WORDS APPROX)

### What I have Learned

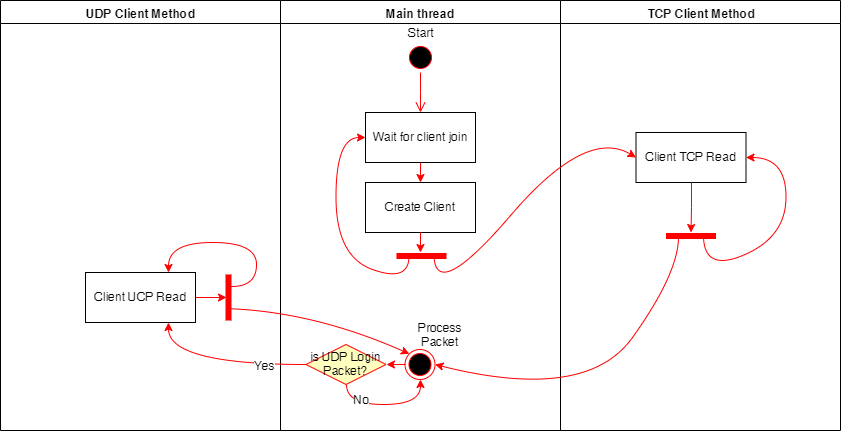
### What Went Well / What Went Wrong

### Challenges I Have Overcome

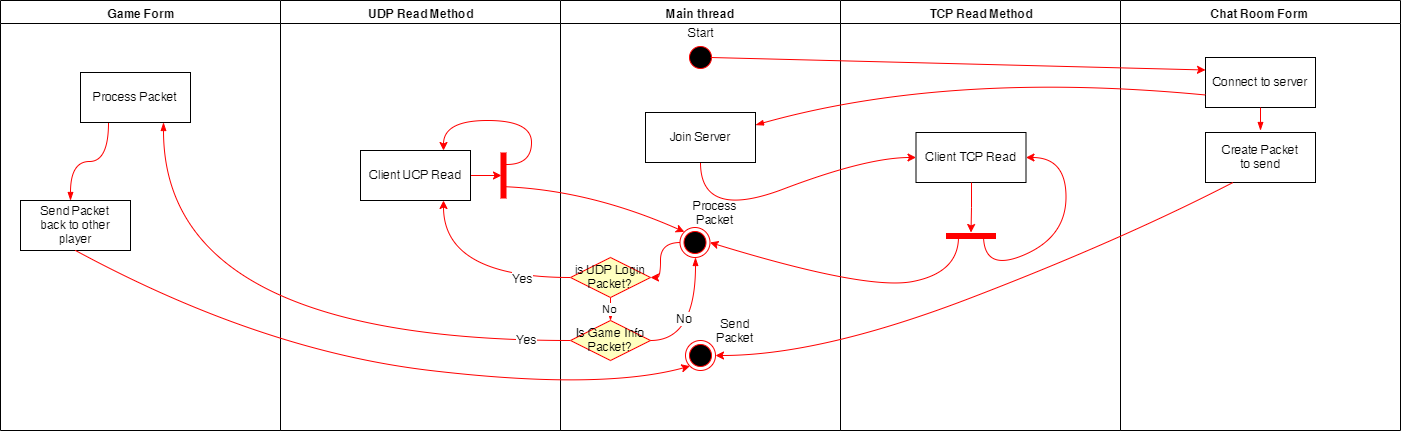
### What I Would Do Differently

# Appendix

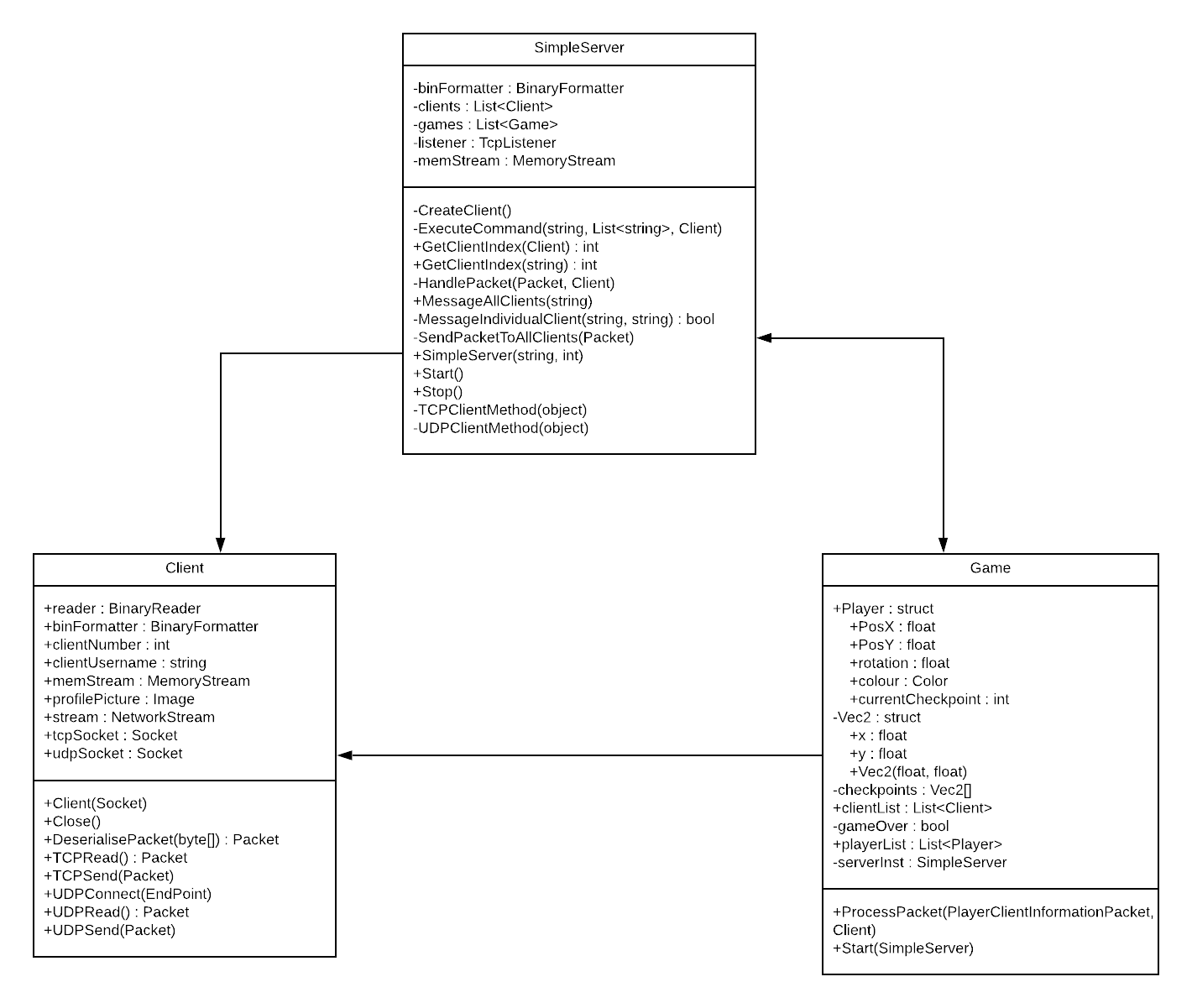
### Figure 1. Server Concurrency Diagram



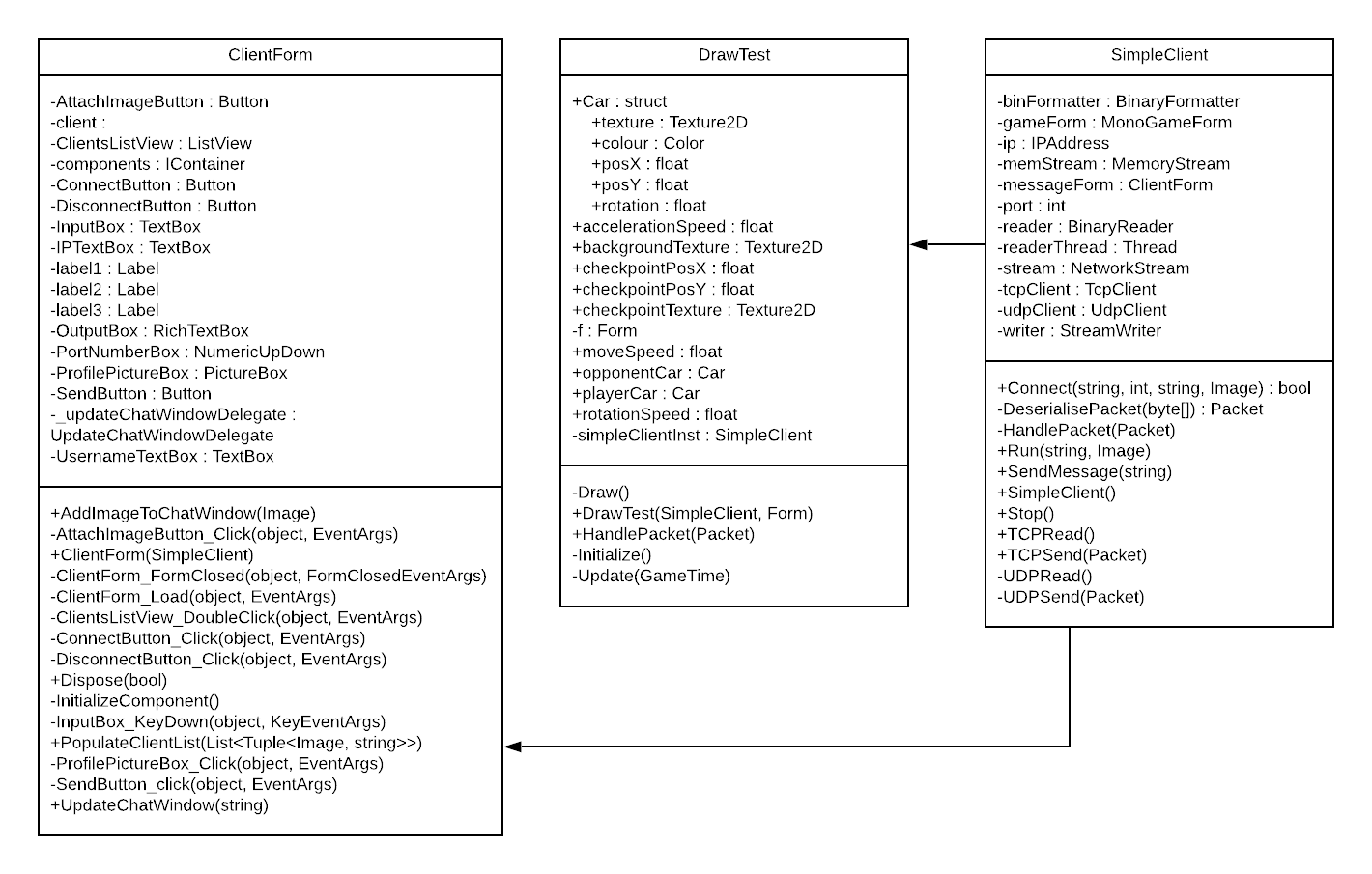
### Figure 2. Client Concurrency Diagram



### Figure 3. Server UML Class Diagram

****

### Figure 4. Client UML Class Diagram



# References