ARCHITEKTURDOKUMENTATION

NEUROFEEDBACK-OPTIMIZED SELF-SCHEDULING PLATFORM

25. JUNI 2023

TECHNISCHE HOCHSCHULE ASCHAFFENBURG

Inhaltsverzeichnis

Inhaltsverzeichnis **1**
1. Einführung und Ziele **2**
1.1 Aufgabenstellung 2
1.2 Stakeholder 3
2. Randbedingungen **4**
2.1 Technische Randbedingungen 4
2.2 Organisatorische Randbedingungen 4
2.3 Coding-Konventionen 4
3. Kontextabgrenzung **5**
3.1 Fachlicher Kontext 5
3.2 Technischer Kontext 6
4. Grundsätzlicher Überblick über die Anwendung **7**
4.1 Teil-Anwendungen 7
4.2 Kommunikation zwischen Client und Server 7
4.3 Überblick der Zuständigkeiten von den Teil-Anwendungen 7
5. Bausteinsicht **9**
5.1 Serverseitige Websocket-Implementierung 9
5.2 Konfigurations-Auslese-Modul 10
5.3 Clientseitige Websocket-Implementierung 11
6. Laufzeitsicht **12**
6.1 Verbindung zum Server, Initialisierung des Clients und Starten eines Spiels 12
6.2 Task eines anderen Clients abnehmen 13
7. Querschnittliche Konzepte **14**
7.1 Benutzeroberfläche 14
7.2 Taskgraph 16
7.3 Spiele 17
7.4 Fehlerbehandlung 20
7.5 Logging 21
7.6 Implementierung des Entwurfsmusters Model View ViewModel 23
8. Glossar **25**

1. Einführung und Ziele

1.1 Aufgabenstellung

Was ist die Neurofeedback-Optimized Self-Scheduling Platform?

- NOSP ist eine Desktop-Anwendung zur kollaborativen Abarbeitung zeitkritischer Tasks.
- Die Anwendung befasst sich mit der optimierten Aufgabenzuteilung einer Arbeitsgruppe
- Hierbei ist jedes Gruppenmitglied selbst für die eigene Zuteilung von Aufgaben verantwortlich.
- Die Software soll einen Masteranden bei seiner Forschungsarbeit unterstützen.
- Ziel der Software ist es, ein Mittel zum Analysieren der Abarbeitung zeitkritischer Tasks mit Unterstützung von Neurofeedback bereitzustellen.

Wesentliche Features

- Mehrere Spieler können gleichzeitig Aufgaben abarbeiten.
- Diese Aufgaben werden durch Spiele / Minigames realisiert.
- Jeder Spieler kann gleichzeitig bis zu vier Spiele spielen.
- Die Anwendung stellt einen Taskgraphen in einer Mikro- und Makro-Ansicht dar.
- Spieler sehen eine Liste der anderen Teilnehmer inklusive der aktiven Spiele und des Workloads jeden Spielers.
- Spieler können anderen Spielern aktive Spiele wegnehmen und mit dem letzten Spielstand weiterspielen.
- Spiele können gewonnen und verloren werden, haben dabei aber auch eine Punktzahl, welche darstellt, wie “gut” eine Task abgearbeitet wurde und darüber hinaus gibt es für jedes Spiel zwei Schwierigkeitsstufen.

1.2 Stakeholder

Wer?	**Bezug**
Entwicklungsteam	Haben in der Regel oberflächliche Kenntnisse zum wissenschaftlichen Aspekt / Anwendungszweck der Software und ein tiefes Verständnis des Codes der Anwendung
Herr Prof. Dr. Biedermann	Benötigt die Architekturdokumentation zur weiteren Unterstützung von Herr Mende bei der Masterarbeit
Herr Mende	Muss für seine Masterarbeit die Anwendung erweitern und daher ein tiefes Verständnis der Applikation erlangen
Probanden	Benutzen die Anwendung, um Herr Mende Daten für seine Masterarbeit zu liefern.

2. Randbedingungen

2.1 Technische Randbedingungen

Randbedingung	**Erläuterung**
Betriebssystem	Die Software läuft auf Windows.
IT-Sicherheit	Die Anwendung wird nur in lokalen Netzwerken genutzt, daher ist die IT-Sicherheit zu vernachlässigen.
Bildschirmauflösung	Die Anwendung benötigt bei einer Auflösungs-Skalierung von 100 % eine Bildschirmauflösung von mindestens 1500 * 900 Pixel

2.2 Organisatorische Randbedingungen

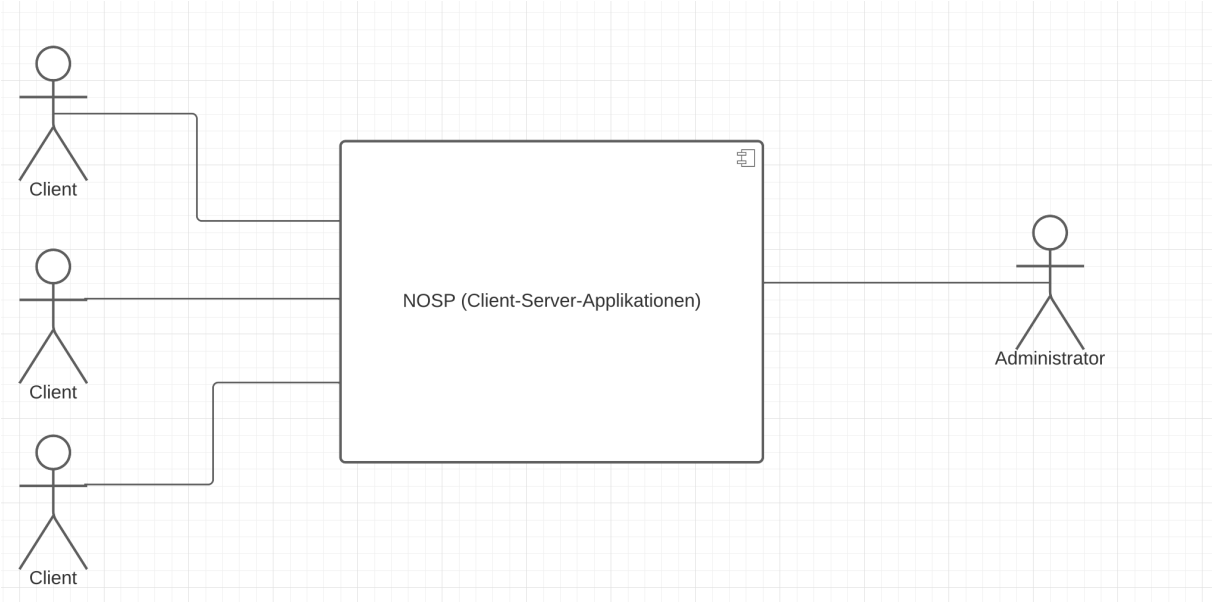
Randbedingung	**Erläuterung**
Team	Herr Mende und Herr Prof. Dr. Biedermann unterstützen das Entwicklungsteam fortlaufend durch regelmäßige Meetings, in denen hauptsächlich Anforderungen weitergegeben werden.
Vorgehensmodell	SCRUM (agil)
Entwicklungswerkzeuge	• Microsoft Visual Studio oder Jetbrains Rider als IDEs • Gitlab als Quellcode-Versionsverwaltungs-Tool • Jira als Ticketverwaltungs-Tool

2.3 Coding-Konventionen

- C# Coding Conventions von Microsoft
- Englischer Code
- Englische oder deutsche Kommentare im Code

3. Kontextabgrenzung

3.1 Fachlicher Kontext



Clients (Benutzer)

Es spielen mehrere Clients gleichzeitig miteinander. Jeder von Ihnen kann sich über einen Taskgraphen Tasks ziehen, die in Form von Minispielen abgearbeitet werden. Dabei spielt jeder Client seine Spiele alleine, kann aber bei stressigen Situationen anderen Clients helfen, indem der Client Tasks von anderen Clients abnimmt und fertig spielt.

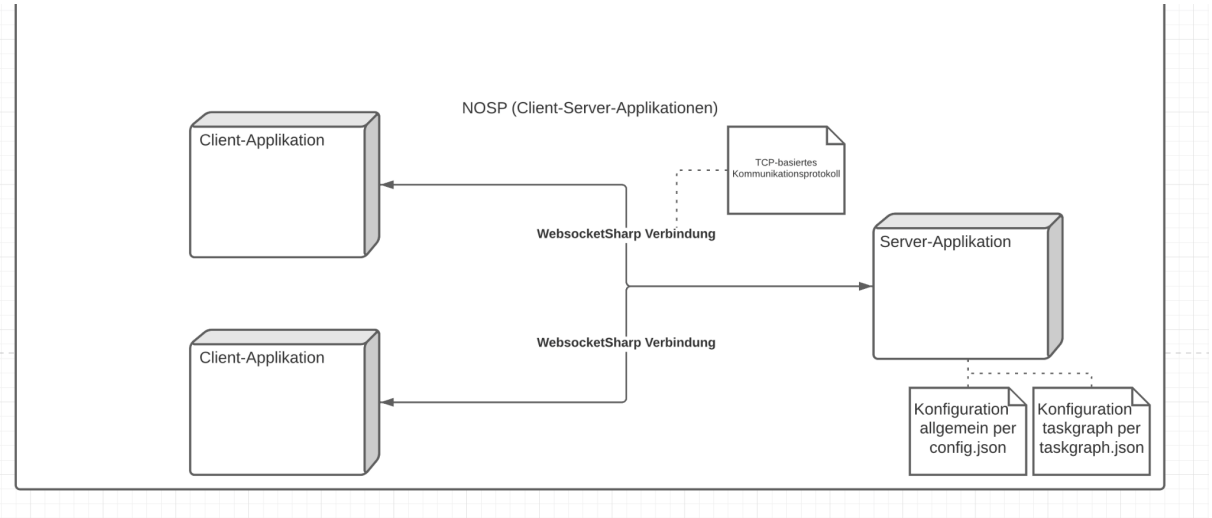
Administrator (Verwalter/Nutzer)

Der Administrator kümmert sich um die Vorbereitung eines Probandendurchlaufes. Er konfiguriert den Taskgraph, die einzelnen Spiele und die Mindestanzahl an Clients vorab über verschiedene Konfigurationsfiles. Zusätzlich erklärt er den Probanden, wie ein Durchlauf abläuft und welche Regeln zu beachten sind.

NOSP (Client-Server-Applikationen)

NOSP sind die verschiedenen Applikationen, welche für einen Probanden-Durchlauf benötigt werden. Unterschieden wird zwischen der Client Applikation, die jeder Client bei sich auf dem Rechner ausführt und so kollaborativ den Taskgraphen abarbeiten können. Sowie der Server Applikation die für die Kommunikation zwischen den Clients besteht und dem Administrator die Möglichkeit bietet, einen Probanden-Durchlauf vorab zu konfigurieren.

3.2 Technischer Kontext



Client-Applikation

Die “Anbindung” der Mitspieler erfolgt über eine grafische Oberfläche. Jede Client-Applikation ist mit dem Server verbunden und kann so mit anderen Client-Applikationen per TCP-Kommunikationsprotokoll interagieren.

Server-Applikation

Der Server steht als zentraler Kommunikationsknoten zur Verfügung und versendet den Taskgraphen sowie allgemeine Konfigurationen an jeden Client, wodurch bei jedem Client das Programm korrekt initialisiert werden kann.

4. Grundsätzlicher Überblick über die Anwendung

4.1 Teil-Anwendungen

NOSP wurde als C#-Anwendung mit Client-Server-Architektur realisiert. Dabei findet Microsoft .NET in Version 6 Einsatz. Es besteht aus folgenden Teil-Anwendungen:

- **Client-Anwendung**
  - Mit C# und dem Frontend-Framework WPF entwickelt
  - Desktopanwendung, mit der jeder Mitspieler Tasks abarbeitet und eine Übersicht über den Spielstand hat
  - Verbindet sich zur Serveranwendung
- **Server-Anwendung**
  - Als C#-Konsolenanwendung implementiert
  - Nimmt Verbindungen von allen Client-Anwendungen an
  - Versorgt die Client-Anwendungen mit allen nötigen Daten und koordiniert zwischen den Clients, um den Spielfluss zu koordinieren

Darüber hinaus gibt es zwei weitere C#-Projekte, welche nur für die Entwicklung nötig sind:

- **Shared-Projekt**
  - Enthält Code, den sich Client- und Server-Anwendung teilen
- **PlaceholderTextbox-Projekt**
  - Enthält Code für eine Textbox, die in der Client-Anwendung benötigt wird

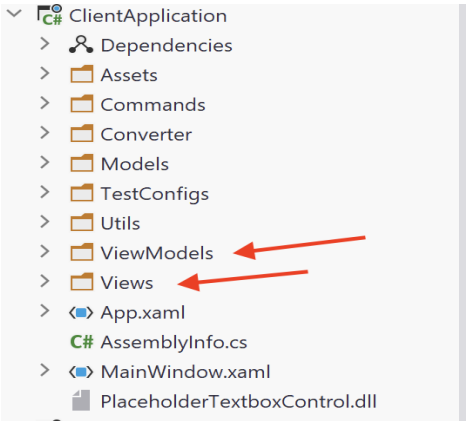
Alle vier Projekte sind in einer Solution zusammengefasst.

4.2 Kommunikation zwischen Client und Server

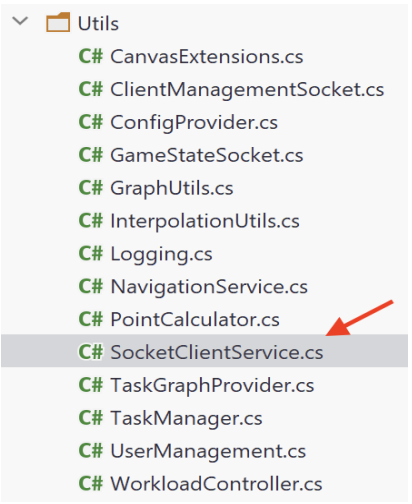
Der Server stellt einen Websocket-Server bereit, der eingehende Verbindungsanfragen annimmt. Jeder Client hat einen Websocket-Client implementiert und kann sich daher zum Websocket-Server verbinden.
Die Verbindung ist bidirektional und nach dem Verbindungsaufbau durchgehend offen.

4.3 Überblick der Zuständigkeiten von den Teil-Anwendungen

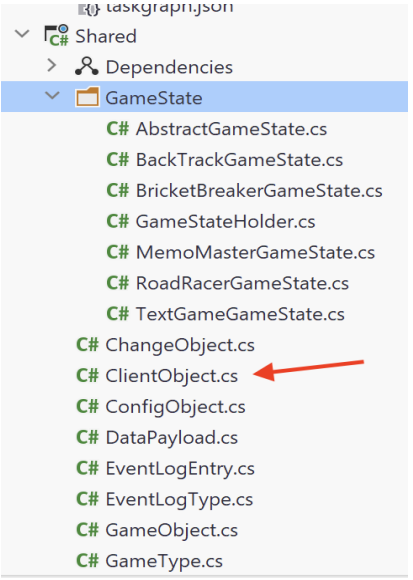
Client-Anwendung: In der Client-Anwendung können Änderung an der UI vorgenommen werden, die für den User sichtbaren Dateien sind in XAML geschrieben und befinden sich in “Views”. Die UI Logik und die Brücke zwischen View und Businesslogik befinden sich in den "ViewModels"-Folder. Die Zusammengehörigkeit einer View und eines ViewModels ist anhand des Namens gekennzeichnet, beispielsweise existiert die “ConnectToServerView” und das dazugehörige ViewModel heißt “ConnectToServerViewModel”.



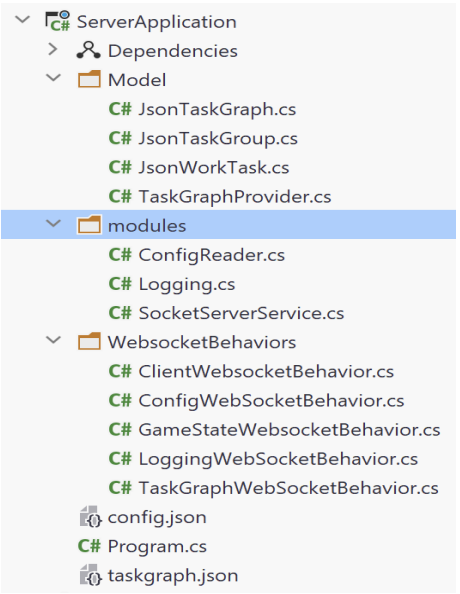
In dem “Utils”-Folder befinden sich die Klassen für die Kommunikation mit dem Server. Die Websocket Verbindung wird in der Klasse “SocketClientService” initalisiert und darin wird auch die Aufteilung in verschiedene “Pfade” verwaltet, mehr dazu in “5.1 Serverseitige Websocket-Implementierung”. Beispielsweise gibt es den “ClientManagementSocket”, welcher für die Verwaltung des aktuellen Clients zuständig ist. Er sendet ,nach Verbindungsaufbau, den Client als ClientObject zum Server und der Server sendet dann wichtige Client Informationen wieder zurück zum Client, zum Beispiel wie viele Clients mindestens angemeldet sein müssen, damit die Tasks starten können.



Shared-Projekt: Im letzten Beispiel wurde der “ClientManagementSocket” betrachtet, dabei wurde ein ClientObject vom Client zum Server gesendet und vom Server aus wird eine Liste von ClientObjects wieder zurück zum Client gesendet. Damit dieses ClientObejct versendet werden kann, muss es serialisierbar sein und konstant gehalten werden, deshalb gibt es das Shared-Projekt, worin alle Daten, die zwischen Client und Server ausgetauscht werden, konstant gehalten werden. Zusammenfassend ist das Shared-Projekt für alle Klassen da, die zwischen Client und Server gesendet und empfangen werden und die in beiden Projekten verwendet werden.



Server-Anwendung: Der Server besitzt keine UI, aber verschiedene “WebSocketBehavior”, die vergleichbar mit den Sockets in der Client-Anwendung sind. Sie definieren, was beim Verbindungsaufbau, eine eingehende Nachricht oder ein Verbindungsabbruch passieren soll. Die ganzen Sockets werden im “SocketServerService.cs” initalisiert.
Die Klasse “ConfigReader.cs” ist für das Einlesen der “config.json” und "taskgraph.json"-Datei zuständig.
Die Klasse “Logging.cs” ist für das schreiben der Log Dateien auf dem Server zuständig.



5. Bausteinsicht

Dieser Abschnitt beschreibt die Zerlegung unserer Client-Server-Architektur und die Kommunikation per WebsocketSharp in einzelnen Modulen.

5.1 Serverseitige Websocket-Implementierung

Der Server besitzt eine Klasse SocketServerService, welche von der Main-Methode aus initialisiert wird. Sie initialisiert den Websocket-Server und speichert alle verbundenen Clients, sowie das Config-Object. Der Server ist auf Port 8000 mit der IP-Adresse localhost erreichbar. Diese sollte für den Produktiv-Betrieb so umgestellt werden, dass Geräte im gleichen lokalen Netzwerk eine Verbindung herstellen können.

Konsequente Trennung der Zuständigkeiten innerhalb des Websockets

Da über den Websocket viele verschiedenartige Daten übertragen werden, darunter Logs, Mitspieler-Daten, den Taskgraphen und den Gamestate, ist der Websocket in mehrere “Pfade” aufgeteilt. Mit der Bibliothek WebSocketSharp ist es so, dass jeder Pfad, z.B. “ws://localhost:8000/logging” eine eigene Klasse zugewiesen bekommt, die das Interface WebSocketBehavior implementiert. Dieses Websocket-Behavior definiert durch Callback-Methoden zum Beispiel, wie eingehende Nachrichten behandelt werden und was bei einer neuen Verbindung passiert.

Websocket-Behavior	**Zuständigkeit**
/logging	Logs vom Client empfangen, um diese zentral serverseitig zu verarbeiten und zu persistieren
/users	Verarbeitet Verbindungsanfragen neuer Clients und sendet an diese initial die zuvor eingelesene Konfiguration.
/clients	Steuert die verschiedenen User-Interaktionen.
/taskgraph	Sendet bei neuer Verbindung den Taskgraphen an den Client und verarbeitet Anfragen zum Auswählen und Abschließen von Tasks, sowie das Pullen von Tasks.
/gamestate	Steuert die Übertragung von GameStates
/config	Sendet bei neuer Verbindung das ConfigObject an den Client.

5.2 Konfigurations-Auslese-Modul

Die Klasse ConfigReader liest zwei Dateien ein, welche beide im Produktiv-Betrieb im gleichen Verzeichnis wie die ausführbare Datei (ServerApplication.exe) liegen sollten. Im Debug-Betrieb ist es praktisch, die Dateien im Root-Verzeichnis der Server-Applikation liegen zu lassen. Daher versucht das Modul zuerst, die Dateien im Verzeichnis der ausführbaren Datei aufzufinden und einzulesen. Wenn sie dort nicht auffindbar sind, wird das voraussichtliche Root-Verzeichnis überprüft.
Der eingelesene Text wird schließlich zu Objekten geparst.
Die ausgelesenen Dateien sind **config.json** und **taskgraph.json**.

config.json

Im Feld Host wird der Name des Hosts angegeben.
Participants enthält, welche Spieler verbunden sein müssen, um eine Session zu starten.
Server-IP gibt die Adresse an, unter der der Server erreichbar sein wird.
WorkloadViewChangeMode gibt an, wie die UI auf einen hohen Workload reagieren soll.

```
{
  "Host": "Hansi",
  "Participants": [
    "horst",
    "heinz"
  ],
  "ServerIP":  "192.168.2.1",
  "WorkloadViewChangeMode": 0
}
```

Konstante	WorkloadViewChangeMode
0	Keine Veränderung
1	Bei hohem Workload soll alles, außer den Spielen, ausgeblendet werden.

taskgraph.json

TaskGroups enthält eine Liste an Tasks. Jede GroupId muss einzigartig sein. Für die TaskId gilt das gleiche.
GameType gibt an, welches Spiel in dieser Task ausgeführt werden muss und TaskDifficulty die Schwierigkeit.

ConnectionsFromTo ist ein Dictionary, welches zu jeder TaskId eine Liste an TaskIds enthält. Der Schlüssel steht hierbei für den Task, von dem die Verbindung (Pfeil) zu einem oder mehreren anderen Tasks ausgeht. Der Wert steht für die Liste an Tasks, zu denen die Verbindung zeigt.

```
{
    "TaskGroups": [
        {
            "GroupId": 1,
            "Tasks": [
                {
                    "TaskId": 1,
                    "GameType": 1,
                    "TaskDifficulty": 0
                }
            ]
        },
        {
            "GroupId": 2,
            "Tasks": [
                {
                    "TaskId": 2,
                    "GameType": 3,
                    "TaskDifficulty": 0
                },
                {
                    "TaskId": 3,
                    "GameType": 4,
                    "TaskDifficulty": 1
                }
            ]
        }
    ],
    "ConnectionsFromTo": {
        "0": [2, 3]
    }
}
```

Definition der Konstanten für GameType und TaskDifficulty in taskgraph.json

GameType	Spiel
0	TippTornado
1	BrickBraker
2	RoadRacer
3	MemoMaster
4	BackTrack

TaskDifficulty	Schwierigkeit
0	Leicht
1	Schwer

5.3 Clientseitige Websocket-Implementierung

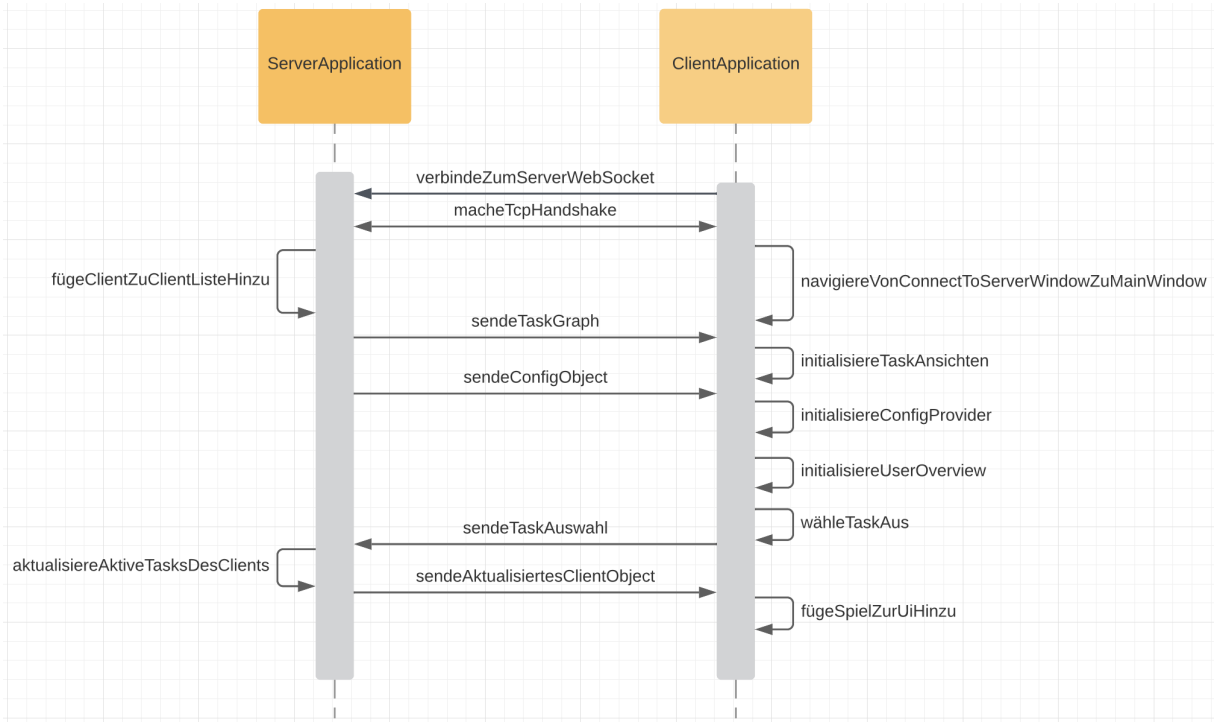
Der Client startet bei Ausführung des Programms verschiedene Verbindungen zu WebSockets, die zur Entgegennahme von veränderten und neuen Objekten implementiert sind und vom Server bereitgestellt werden. Mit Hilfe dieser werden Events getriggert, die bestimmte Verhaltensweisen mit sich bringen.

6. Laufzeitsicht

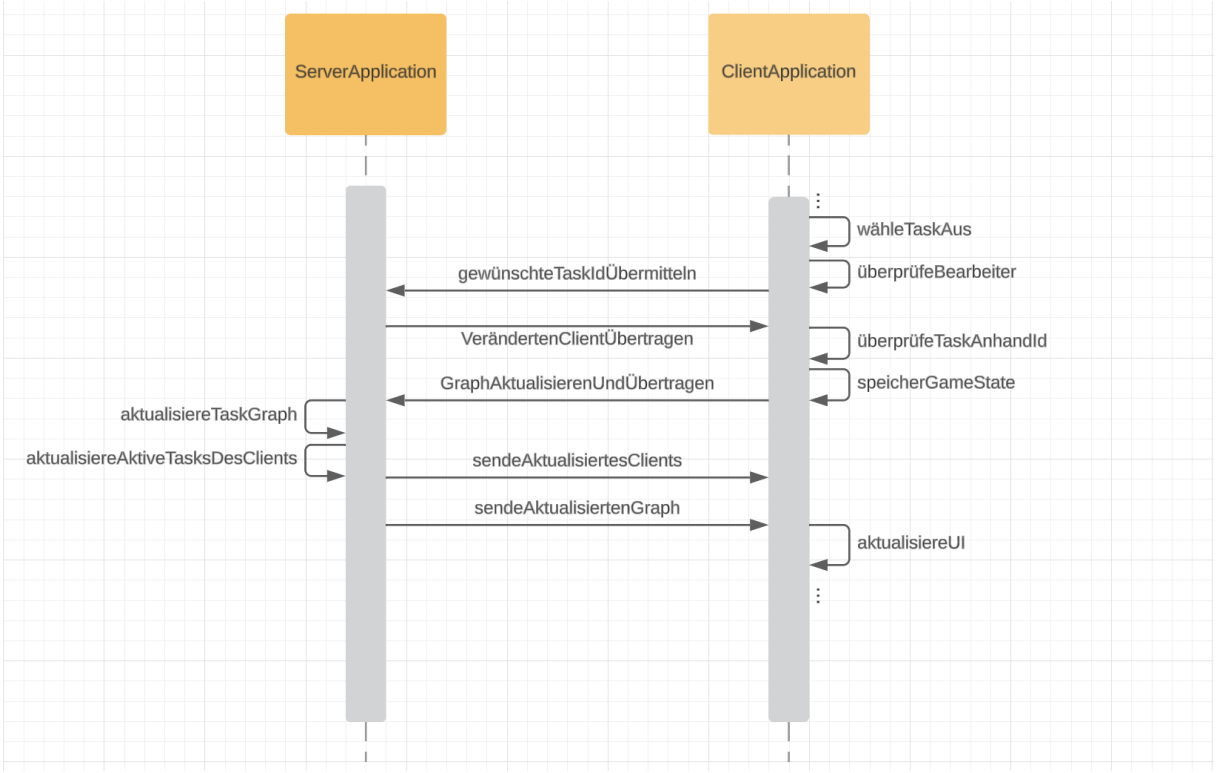
Diese Sicht visualisiert im Gegensatz zur statischen Bausteinsicht das dynamische Zusammenspiel und den Ablauf der Anwendung im Betrieb.

6.1 Verbindung zum Server, Initialisierung des Clients und Starten eines Spiels

Im folgenden Beispiel wird mithilfe eines Sequenzdiagramms exemplarisch dargestellt, welche Interaktionen in der Anwendung stattfinden, wenn ein Client sich zum Server verbindet, alle notwendigen Daten im Client daraufhin initialisiert werden und er schließlich das Spiel RoadRacer starten möchte.



6.2 Task eines anderen Clients abnehmen



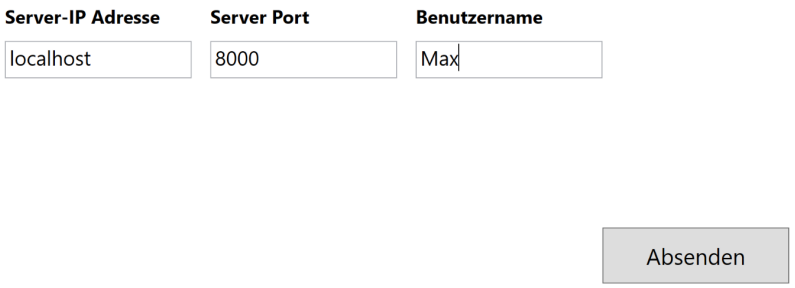
7. Querschnittliche Konzepte

7.1 Benutzeroberfläche

Die Benutzeroberfläche ist ein einfaches Windows-Fenster. Dieses kann in der Größe verstellt werden.

Login-Fenster

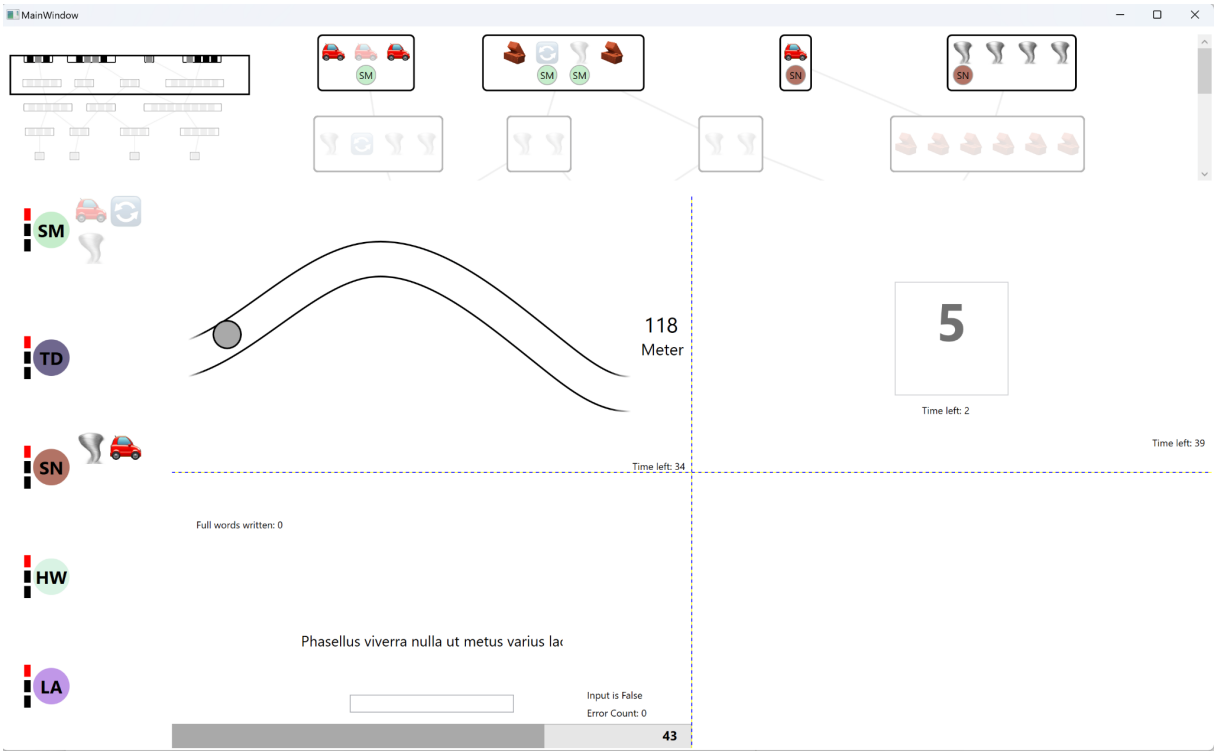
Das Login-Fenster besitzt drei Eingabefelder für die Adresse des Servers, seinen Port und den Benutzernamen. Diesen kann man frei wählen. Um die Verbindung herzustellen, muss auf “Absenden” gedrückt werden.
Wenn die Verbindung nicht hergestellt werden konnte, erscheint eine Fehlermeldung in der Benutzeroberfläche.



MainView

MainView ist die Hauptansicht der Anwendung. Sie ist unterteilt in drei Teile.

1. Die Spiel-Übersicht mit vier gleichgroßen Feldern, in welchen Spiele erscheinen, die der Spieler auswählt.
2. Die User-Übersicht, in der man selbst ganz oben steht und weiter unten eine Liste aller anderen Mitspieler. Der Benutzername des Nutzers wird d Durch einen Klick auf die Game-Icons kann man eine Task abnehmen, wenn diese nicht von einem selbst ist oder das Spiel nicht bereits läuft. Auch zu beachten ist, dass hierbei ein Natural Mapping angewendet wird, indem die Icons an der gleichen Gitter-Position sind wie die Spiele in der Spiel-Übersicht.
3. Die Taskgraph-Übersicht, welche in zwei Teile unterteilt ist:
    a. Die Mikroansicht des Taskgraphen rechts, in der der Taskgraph groß und vollständig mit allen Tasks, Verbindungen und Taskgruppen dargestellt wird. Auswählbare Tasks werden in voller Farbintensität dargestellt. Der Kreis unter dem Spiel-Icon zeigt, welcher Benutzer das Spiel spielt. Durch einen Mausklick auf das Spiel-Icon kann man die Task auswählen und durch einen Klick auf den Benutzer-Kreis kann die Task abgenommen werden.
    b. Die Makroansicht links, welche den gesamten Taskgraph und die aktuelle Scroll-Position mithilfe eines rechteckigen Rahmens darstellt. Auch dort sind noch nicht freigeschaltete oder nicht auswählbare Tasks grau.



7.2 Taskgraph

Der Task Graph setzt sich aus verschiedenen Task Gruppen zusammen, die miteinander verbunden sind. Jede Task Gruppe beinhaltet dabei mindestens eine Task, die von einem speziellen Spieltyp ist und individuell angepasst werden kann. (Schwierigkeit, Spielart, usw.) Nachdem der Graph beim Start des Servers eingelesen wurde (siehe Konfigurations-Auslese-Modul), wird dieser an die Clients übertragen und kann dort über das User-Interface repräsentiert werden.
Um eine Übersicht des gesamten Graphen und der aktuellen Position in der Detailansicht zu erhalten, wird in der linken oberen Ecke der Makro-Graph angezeigt.

Makro-Graph:

- die Farbgebung der jeweiligen Task (vollfarbiges Quadrat) verdeutlicht, wie der aktuelle Status ist:
  grau ⟶ Task wird aktuell ausgeführt; schwarz ⟶ Task ist ausführbar
- Rechteck: repräsentiert die Position der Scroll Ansicht im gesamten Graphen. (bewegt sich dynamisch mit, wenn die Hauptansicht des Graphen verändert wird.)
- Reduzierung der Farbintensität von Linien, Task Gruppen und Tasks verdeutlicht, dass diese aufgrund fehlender Abhängigkeiten nicht gespielt werden können.

Haupt-Graph:

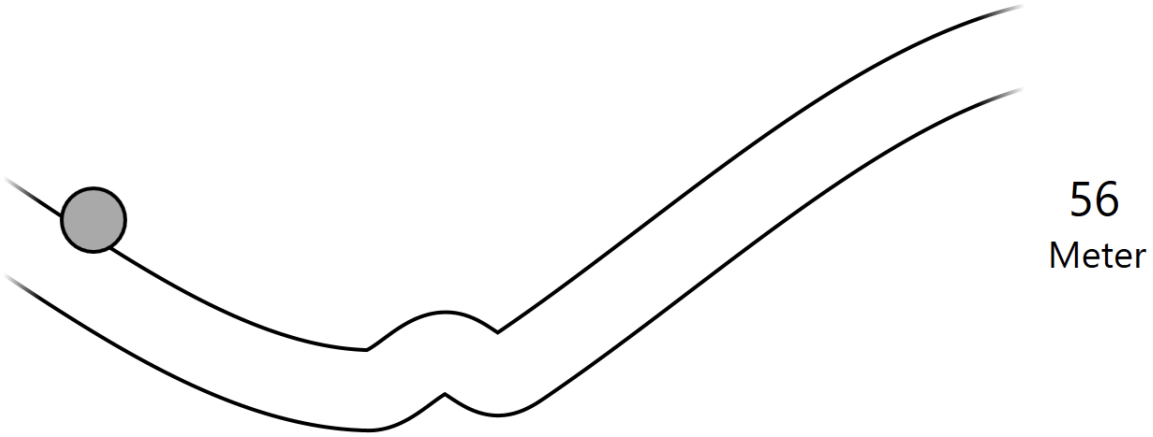
- Symbol zeigt die Art des Spiels. Der User, der das Spiel spielt, sieht das Symbol in dezenter Farbe, da diese von diesem nicht weiter angeklickt werden kann.
- Userkreis zeigt, welcher User die Task aktuell bearbeitet. Der Kreis hat die gleiche Farbe und Aufschrift wie derjenige des jeweiligen Users in der vertikalen Übersicht links.
- Reduzierung der Farbintensität von Linien, Task Gruppen und Tasks verdeutlicht, dass diese aufgrund fehlender Abhängigkeiten nicht gespielt werden können.
- Abnehmen von Task: Durch Klick auf Symbol, Kreis unter Symbol oder Symbol in der Benutzerübersicht kann man Task abnehmen (wenn Bedingungen erfüllt)

7.3 User Overview

Über die Übersicht wird dargestellt, welche Nutzer aktuell mit dem Server verbunden sind. Neben dem jeweiligen Benutzer auf der rechten Seite lässt sich erkennen, in welchem Spielfeld welches Spiel bei diesem ausgeführt wird. Das Wegnehmen einer Task kann dabei auch hierüber geschehen. Die eigenen Spiele werden auch hier ausgegraut dargestellt. Dies soll das Auswählen einer Task, die von dem User selbst aktuell gespielt wird, symbolisch blockieren.

7.3 Spiele

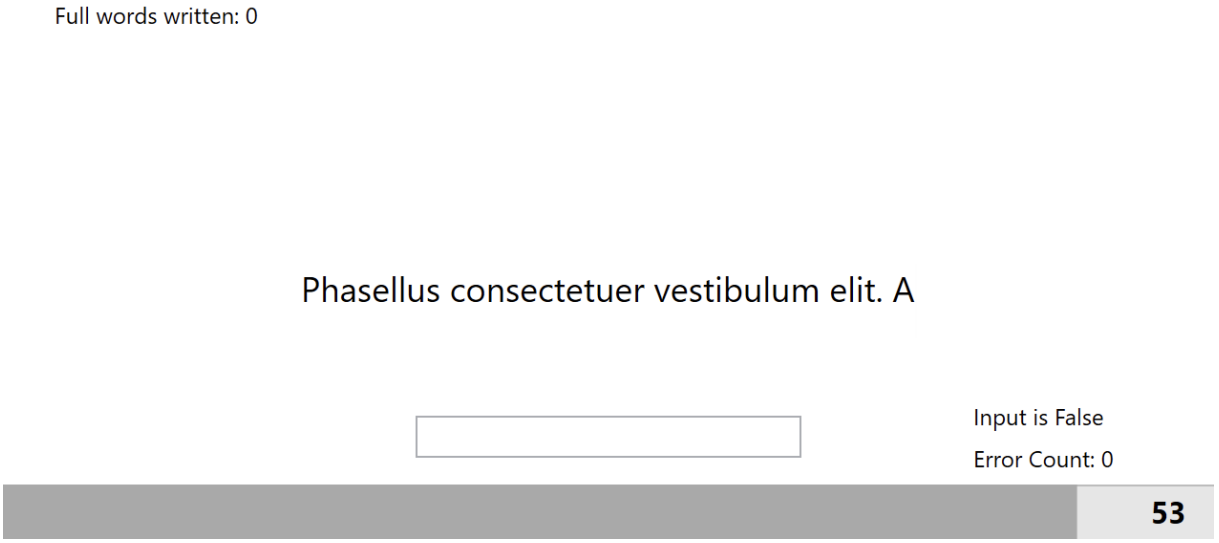
Road Racer



Time left: 48

Die aktuelle Version des Roadracers beinhaltet einen Kreis, der zwischen der Linie geführt werden muss, um innerhalb von 60 Sekunden die Mindestdistanz zu erreichen. Rechts wird angezeigt, wie viel Strecke bereits innerhalb der Linie zurückgelegt wurde, und wie viel Zeit noch übrig ist. Durch die Pfeiltasten UP und DOWN kann man den Kreis steuern.

Tipp Tornado

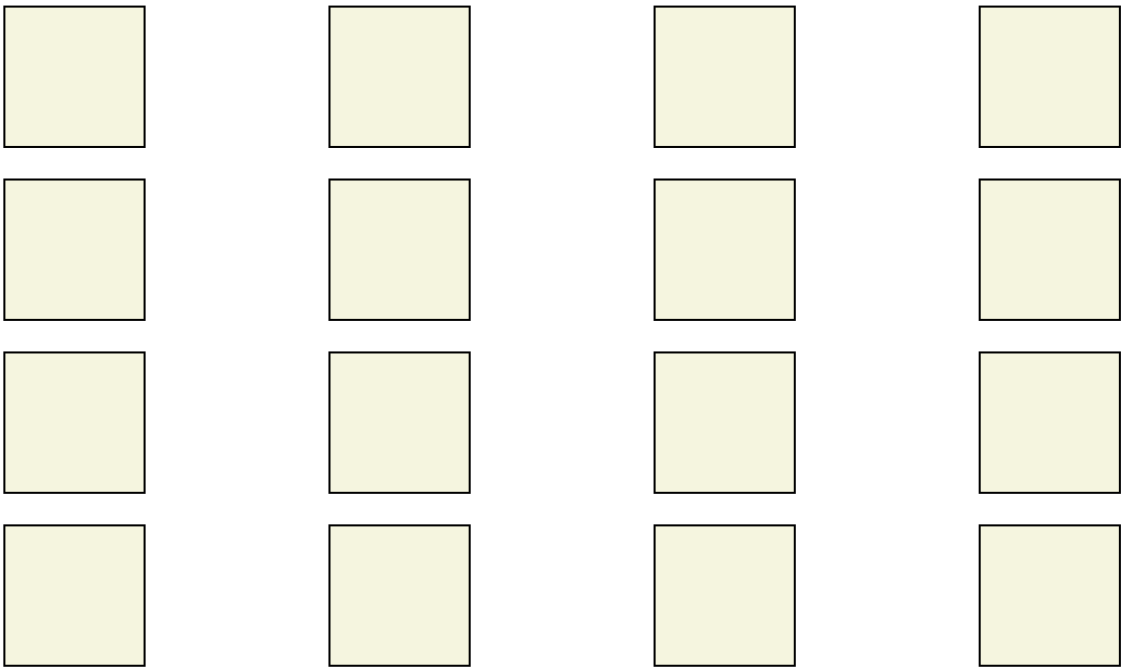


Die aktuelle Version des Tip Tornados zeigt einen Text, der nachgeschrieben werden muss, ohne die Maximalanzahl an Fehlern zu überschreiten. Rechts unten lässt sich erkennen, wie viele Tippfehler bereits eingebaut wurden (Error Count). Zudem kann man auch die verbleibende Zeit ablesen, die in Form eines Balken dargestellt wird.

Backtrack

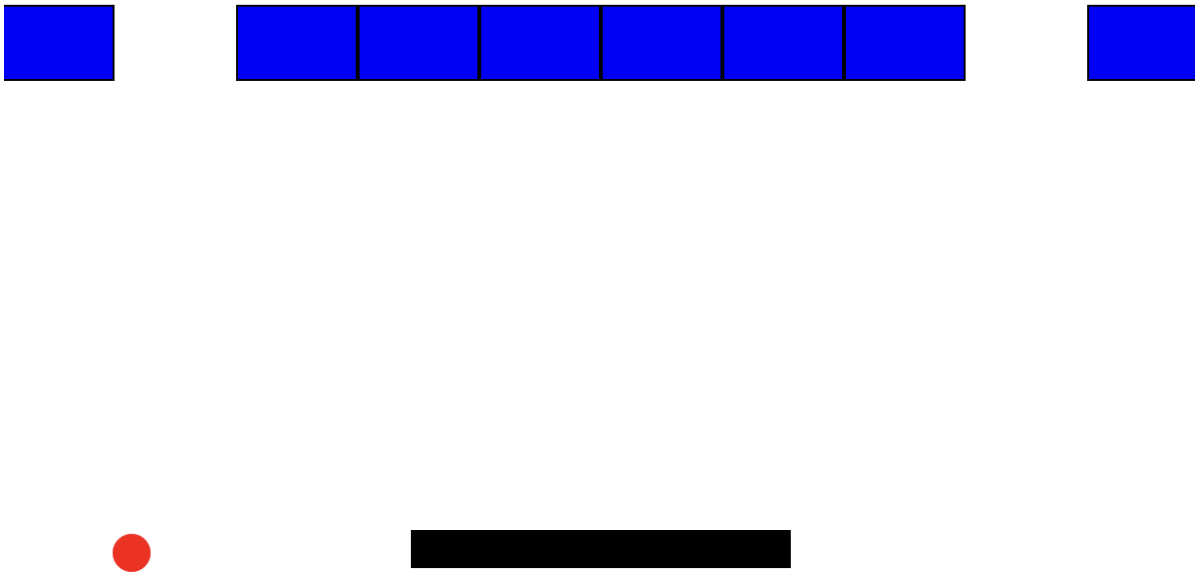
Die aktuelle Version von Backtrack zeigt anfangs 2 Zahlen nacheinander. Anschließend erscheint ein Eingabefeld, in das man diejenige Zahl eingeben muss, die 2 Anzeigen zurückliegt. Unter dem Eingabefeld lässt sich erkennen, wie viel Zeit noch übrig ist, um die Eingabe zu machen, was auch bei der Anzeige neuer Tasks der Fall ist. Die verbleibende Gesamtzeit des Spiels steht, wie bei den anderen Spielen auch, rechts unten. Die N-Backs sind auf 2 gestellt.

Memomaster



Die aktuelle Version von Memomaster implementiert noch keine Logik. Dadurch lässt sich dieses Spiel noch nicht spielen!

Brick Breaker



Die aktuelle Version von Brick Breaker implementiert noch keine erweiterte Spiellogik. Dadurch kann der Ball nur mit Hilfe des Balkens in verschiedene Richtungen abgelenkt und eine Reihe an Blocks zerstört werden. Bewegt wird hierbei per Key-Eingabe mit Left und Right.

7.4 Fehlerbehandlung

Um eventuelle Probleme bei der Weiterentwicklung der Anwendung zu erkennen, hat man mehrere Möglichkeiten.

- Server- und Client-Anwendungen erzeugen Anwendungs-Logs, welche im gleichen Verzeichnis wie die ausführbaren Dateien abgespeichert werden.
- Da die Server-Applikation eine Konsolenanwendung ist, werden Informationen und Fehler auch in der Konsole ausgegeben.
- Bei der Client-Applikation muss diese im Debug-Modus gestartet werden. Dort werden Fehler dann im Debug-Output ausgegeben.

7.5 Logging

Anwendungs-Logs

Anwendungs-Logs auf Client- und Server-Seite sind nach dem Schema “log_[DatumUndUhrzeit].txt” benannt.
Jeder Eintrag ist besitzt folgende Informationen:

- Datum und Uhrzeit des Eintrags
- Log-Level
- Message
- Gegebenenfalls die Exception

```
2023-05-30 19:36:45.763 +02:00 [INF] Server Application started.
2023-05-30 19:38:03.105 +02:00 [INF] New Client connected Uid:d8c5fc2d-45db-4e32-81af-ade2c5839f96 Games active:0
2023-05-30 19:38:11.375 +02:00 [INF] New Client connected Uid:d8c5fc2d-45db-4e32-81af-ade2c5839f96 Games active:1
2023-05-30 19:38:25.322 +02:00 [INF] New Client connected Uid:a7ec53cd-a94c-40dd-8531-95004aebd197 Games active:0
2023-05-30 19:38:41.786 +02:00 [INF] Client disconnected Count: 2
2023-05-30 19:38:47.445 +02:00 [INF] Client disconnected Count: 1
2023-05-30 19:49:46.198 +02:00 [INF] New Client connected Uid:b58f7eb1-86d8-434e-a36e-50a6c4031162 Games active:0
2023-05-30 19:49:49.093 +02:00 [INF] New Client connected Uid:b58f7eb1-86d8-434e-a36e-50a6c4031162 Games active:1
2023-05-30 19:49:53.630 +02:00 [INF] Client disconnected Count: 1
```

Event-Logs

Event-Logs sind für die spätere Analyse des Spielverlaufs nötig und werden immer zentral am Server gespeichert. Für eine vereinfachte Analyse werden sie als .csv-Dateien (Semikolon-separiert) angelegt. Die Benennung ist “event_log_[DatumUndUhrzeit].csv”.
Die erste Zeile enthält stets die Spaltenbezeichnungen.
Jeder Eintrag besitzt zum aktuellen Stand folgende Informationen:

- Datum und Uhrzeit, welche beim Eingang der Log-Nachricht immer serverseitig ermittelt wird, um möglicherweise verschiedene System-Uhrzeiten bei den Clients zu ignorieren, da diese für die spätere Analyse problematisch wären.
- Der Benutzername des Users
- Ein Event-Type, welches angibt, um welche Art von Log es sich handelt. Es gibt neben Game-Events (Ereignis in Bezug auf ein Spiel / eine Task) auch Key-Events (für Tastendrücke), Neuro-Events (für Neurofeedback) und User-Interaction (Interaktion eines Users)
- Message, welche das Ereignis beschreibt.

	A	B	C	D
1	timestamp	user	event_type	message
2	05/31/2023 10:14:00.382	xcjlt	GAME_EVENT	Text game time left: 60
3	05/31/2023 10:14:03.518	xcjlt	GAME_EVENT	xcjlt added PathPilot to active games
4	05/31/2023 10:14:03.526	xcjlt	GAME_EVENT	PathPilot started
5	05/31/2023 10:14:03.545	xcjlt	GAME_EVENT	Text game time left: 60
6	05/31/2023 10:14:03.570	xcjlt	USER_INTERACTION	User input text: []
7	05/31/2023 10:14:03.571	xcjlt	GAME_EVENT	Input text is empty
8	05/31/2023 10:14:04.517	xcjlt	GAME_EVENT	PathPilot time left: 19, currentMeters: 0
9	05/31/2023 10:14:05.516	xcjlt	GAME_EVENT	PathPilot time left: 18, currentMeters: 5,675378100000001
10	05/31/2023 10:14:06.517	xcjlt	GAME_EVENT	PathPilot time left: 17, currentMeters: 11,553016999999999
11	05/31/2023 10:14:07.537	xcjlt	GAME_EVENT	PathPilot time left: 16, currentMeters: 18,6988025
12	05/31/2023 10:14:08.540	xcjlt	GAME_EVENT	PathPilot time left: 15, currentMeters: 25,1607188
13	05/31/2023 10:14:09.530	xcjlt	GAME_EVENT	PathPilot time left: 14, currentMeters: 32,086723899999996
14	05/31/2023 10:14:10.527	xcjlt	GAME_EVENT	PathPilot time left: 13, currentMeters: 39,060658
15	05/31/2023 10:14:11.535	xcjlt	GAME_EVENT	PathPilot time left: 12, currentMeters: 45,46953809999999
16	05/31/2023 10:14:12.531	xcjlt	GAME_EVENT	PathPilot time left: 11, currentMeters: 46,342195199999985
17	05/31/2023 10:14:13.526	xcjlt	GAME_EVENT	PathPilot time left: 10, currentMeters: 48,34159889999998
18	05/31/2023 10:14:13.940	xcjlt	GAME_EVENT	Text game time left: 60
19	05/31/2023 10:14:15.779	xcjlt	GAME_EVENT	PathPilot win

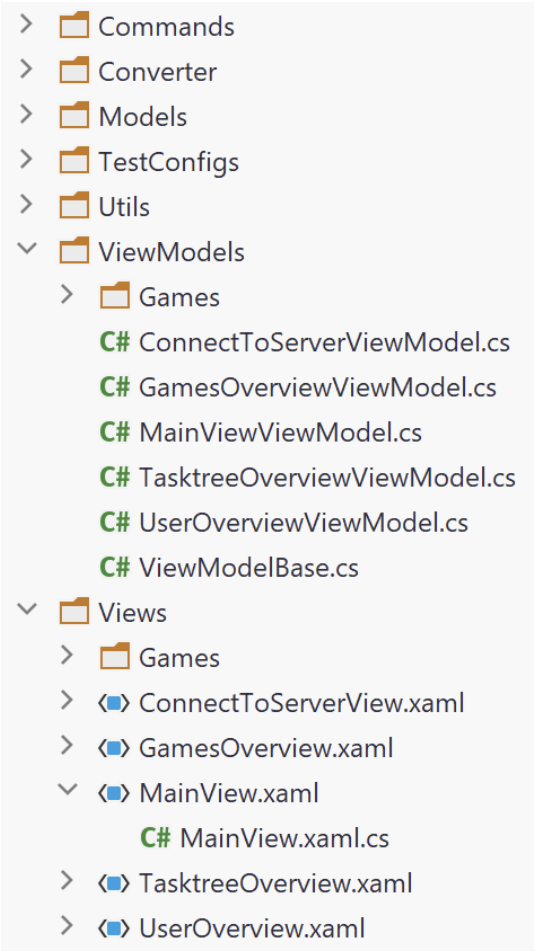
Zu erwähnen ist auch, dass “message” zukünftig folgende Spalten aufgeteilt werden soll:

- Mental_Workload
- Keyboard_Input
- BT_Time
- BT_Points
- BT_Mistakes
- BB_Time
- BB_Points
- BB_Mistakes
- MM_Time
- MM_Points
- MM_Mistakes
- TT_Time
- TT_Points
- TT_Mistakes
- RR_Time
- RR_Points

7.6 Implementierung des Entwurfsmusters Model View ViewModel

Die Client-Applikation ist nach dem Entwurfsmuster Model View ViewModel (kurz: MVVM) aufgebaut. Dieses erlaubt eine konsequente Trennung von Benutzeroberfläche und Businesslogik.

“View” enthält die Benutzeroberfläche, darunter zählen hier zum einen die Layout-Dateien (Dateien mit der Endung “.xaml”), sowie die Code-Behind-Dateien (Endung “.xaml.cs”). Diese sind stets an eine Layout-Datei geknüpft und enthalten einen Teil der UI-Logik, zum Beispiel, was bei einem Knopfdruck passieren soll. Hier im Beispiel ist MainView.xaml eine Layout-Datei und MainView.xaml.cs die dazugehörige Code-Behind-Datei.



“ViewModel” enthält die tiefere UI-Logik und hat greift auf das “Model” zu, welches z.B. Funktionen für das Versenden oder Empfangen von Informationen über die Websocket-Verbindung bereitstellt. Im ViewModel läuft zum Beispiel auch die Steuerung, wann ein Spiel gestartet wird, sowie der Spiel-Timer.

Die View registriert Observer, um Properties im ViewModel abzuhören und kann auf Methoden des ViewModels zugreifen, während das ViewModel keinen direkten Zugriff auf Methoden der View hat, wodurch eine vollständige Trennung der UI von der Business-Logik erreicht werden kann.

Veranschaulichung der MVVM-Architektur am Spiel RoadRacer

Im Code-Ausschnitt der Datei “RoadRacerView.xaml” sieht man, dass hier in einer xml-artigen Struktur Elemente der Benutzeroberfläche, wie zum Beispiel dem Kreis (in Form einer Ellipse), angelegt werden. Das Aussehen der UI-Elemente kann über zahlreiche Eigenschaften, wie z.B. “Color” oder “Height” festgelegt werden.

```
        Height="340" Width="640"
        Background="White" Loaded="UserControl_Loaded">
<Grid>
    <Canvas x:Name="Canvas" Width="630" Height="290" ClipToBounds="True">
        <Polyline x:Name="PathPolylineBackground" Points="0,0" Stroke="Black" StrokeThickness="45" Fill="Transparer
        <Polyline x:Name="PathPolylineForeground" Points="0,0" Stroke="White" StrokeThickness="41" Fill="Transparer
        <Ellipse x:Name="Circle" Width="36" Height="36" Stroke="Black" StrokeThickness="2" Fill="DarkGray" Canvas.
    </Canvas>
    <Rectangle Width="40" Height="300" HorizontalAlignment="Left">
        <Rectangle.Fill>
            <LinearGradientBrush StartPoint="0,0" EndPoint="1,0">
                <GradientStop Color="White" Offset="0.5"></GradientStop>
                <GradientStop Color="Transparent" Offset="1"></GradientStop>
            </LinearGradientBrush>
        </Rectangle.Fill>
    </Rectangle>
    <Rectangle Width="100" Height="300" HorizontalAlignment="Right">
        <Rectangle.Fill>
            <LinearGradientBrush StartPoint="1,0" EndPoint="0,0">
                <GradientStop Color="White" Offset=".75"></GradientStop>
                <GradientStop Color="Transparent" Offset="1"></GradientStop>
            </LinearGradientBrush>
        </Rectangle.Fill>
    </Rectangle>
    <TextBlock HorizontalAlignment="Right" Margin="0,56,0,0" TextWrapping="Wrap" Text="{Binding Path=
    <TextBlock TextAlignment="Center" Margin="0,0,10,0" HorizontalAlignment="Right" TextWrapping="Wrap" Ve
```

Dieser Code-Ausschnitt zeigt den Konstruktur in der Datei “RoadRacerView.xaml.cs”, in dem das ViewModel mittels der “DataContext”-Eigenschaft an die View gebunden wird.

```
public RoadRacerView()
{
    InitializeComponent();
    _viewModel = new RoadRacerViewModel(NavigationService.GetInstance());
    DataContext = _viewModel;
}
```

Diese Methode greift zum einen auf das Layout zu und fragt hier die Position des Kreises ab. Zum Anderen greift sie auch auf die Methode “UpdateCurrentMeters” des ViewModels zu. Somit wird das ViewModel mit der Information, ob die Meter-Anzahl erhöht werden kann, versorgt, ohne dass es selbst auf Teile der UI zugreifen muss.

```
private void UpdateIsCircleOnPath(PointCollection pathPoints)
{
    var circleTop :double = Canvas.GetTop(Circle);
    var circleBottom :double = circleTop + Circle.ActualHeight;

    var circleHorizontalMiddle :double = Canvas.GetLeft(Circle) + Circle.ActualWidth / 2;
    var rangeToTake :int = (int)Circle.ActualHeight / 2;
    var relevantPathPointRange :List<Point> = pathPoints // PointCollection
        .Skip((int)circleHorizontalMiddle - rangeToTake / 2)
        .Take(rangeToTake) // IEnumerable<Point>
        .ToList();

    var isCircleOnPath = false;

    foreach (var point in relevantPathPointRange)
    {
        if (point.Y >= circleTop && point.Y <= circleBottom)
        {
            isCircleOnPath = true;
            break;
        }
    }

    var snapshot = new CircleAndPathSnapshot(_currentPathPointCollectionOffset, isCircleOnPath);
    _viewModel.UpdateCurrentMeters(snapshot);
}
```

Das ViewModel enthält Beispielsweise die Methode “CheckIfAlreadyWinOrLose”, welche im Beispiel bei einem verlorenen oder gewonnenen Spiel die aktive Task des Spielers entfernt und im Hintergrund daraufhin der Taskgraph aktualisiert wird.

```
private void CheckIfAlreadyWinOrLose()
{
    bool? win = null;
    if (TimeLeft == 0 && CurrentMeters < _requiredMetersToNotLose)
    {
        win = false;
    }
    else if (TimeLeft == 0 && CurrentMeters >= _requiredMetersToNotLose)
    {
        win = true;
    }
    else if (CurrentMeters >= RequiredMetersToWinInstantly)
    {
        win = true;
    }

    if (win != null)
    {
        RemoveActiveTask();
        MessageBox.Show((bool)win ? "Congratulations, you win!" : "Sorry, you lose.");
        Logging.LogGameEvent(msg: $"RoadRacer {((bool)win ? "win" : "lose")}");
    }
}
```

8. Glossar

NOSP: Neurofeedback-Optimized Self-Scheduling Platform

Task: Ein Arbeitsschritt oder eine Aufgabe.

Neurofeedback: Technik zur Selbstregulation der Gehirnfunktion durch Darstellung von EEG-Informationen in Echtzeit.

Minigame: Ein kurzes, einfaches Spiel.

Taskgraph: Eine visuelle Darstellung von miteinander verbundenen Aufgaben.

Mikroansicht: Detaillierte Ansicht einer spezifischen Komponente.

Makroansicht: Übersichtliche Analyse der gesamten Komponente.

Workload: Die Gesamtmenge an Arbeit oder Anzahl an Aufgaben, die zu bewältigen sind.

SCRUM: Ein agiles Framework für Projektmanagement und Produktentwicklung.

IDE: Software-Anwendung, die Tools für die Programmierung bereitstellt.

Frontend-Framework: Sammlung von Tools und Bibliotheken für die Erstellung von Benutzeroberflächen.

Server: System, das Dienste oder Ressourcen an andere Computer (Clients) über ein Netzwerk bereitstellt.

Client: Computer oder Anwendung, die auf Dienste oder Ressourcen eines Servers zugreift.

Solution: Eine Gruppe von Projekten oder Komponenten, die zusammenarbeiten, um ein Ziel zu erreichen.

Callback-Methode: Funktion, die zu einem späteren Zeitpunkt als Reaktion auf ein Ereignis aufgerufen wird.

Root-Verzeichnis: Das oberste Verzeichnis in einer Verzeichnishierarchie eines Dateisystems.

UI: Komponente eines Systems oder einer Software, die Benutzerinteraktion ermöglicht.

Parsen: Prozess der Strukturanalyse einer Symbolsequenz.

Dictionary: Datenstruktur, die Schlüssel-Wert-Paare speichert.

Natural Mapping: Designprinzip, bei dem die Beziehung zwischen Steuerelementen und Aktionen intuitiv ist.

csv-Datei: Dateiformat, das tabellarische Daten speichert, wobei Felder durch Kommas oder andere Trennzeichen getrennt sind.

Code-Behind-Datei: Separate .cs-Datei, die mit einer XAML-Datei verknüpft ist und deren interaktive Funktionalitäten durch Implementierung von Ereignishandlern und Businesslogik definiert.