1. **Software Design Document**
2. **Introduction**

This document presents the architectural design for the Rehabilitation Driving Simulator project, a software suite which aims to help individuals re-learn how to drive after a traumatic brain injury by allowing users to experience a simulation of real-world driving.

* 1. System Objective

The goal of the project is to provide a user-friendly and realistic experience for people using the simulator; as a model of real-world driving, the simulator uses hardware and software in tight correspondence. The software will feature different courses to navigate to in order to train and test the user in different driving scenarios. The motion simulation software will create haptic feedback that it will send to the hardware in order to provide users a realistic sensation of driving a car.

* 1. Hardware, Software, and Human Interfaces
     1. Hardware Interfaces:
        1. Network:

The base system, a Windows 10 PC, is connected to the internet via a wired LAN connection.

* + - 1. Audio/Visual:

Three HD video monitors and two speakers are connected to the system via HDMI connections.

* + - 1. Motion Simulation:

Two single degree-of-freedom screw motors and a “rumble” device in the seat are connected to the system.

* + 1. Software Interfaces:

The system interacts with the motion simulation hardware through purpose-built motion simulation software (Sim Commander and SimTools).

The top-level user software is primarily based upon MOTS commercial games that run via the Steam distribution and digital rights management platform.

The core of our team’s software - telemetry, content, and scheduler - interacts with the aforementioned motion simulation and top-level user software via their respective APIs and command-line arguments.

* + 1. Human Interfaces:

Human users interact with the system in 3 primary ways:

* + - 1. Logitech wireless keyboard, touchpad, and mouse connected by USB unifying receiver.
      2. XBox-branded, Logitech-manufactured steering wheel connected by USB cable.
      3. Custom-built gas, brake, and clutch pedals connected by USB cable.

1. **CSCI Description**
   1. Concept of Execution

System execution will begin with the experiment scheduler application, which will call the top-level user software with the appropriate parameters for test or training scenarios. The telemetry/data gathering functionality is enabled by default.

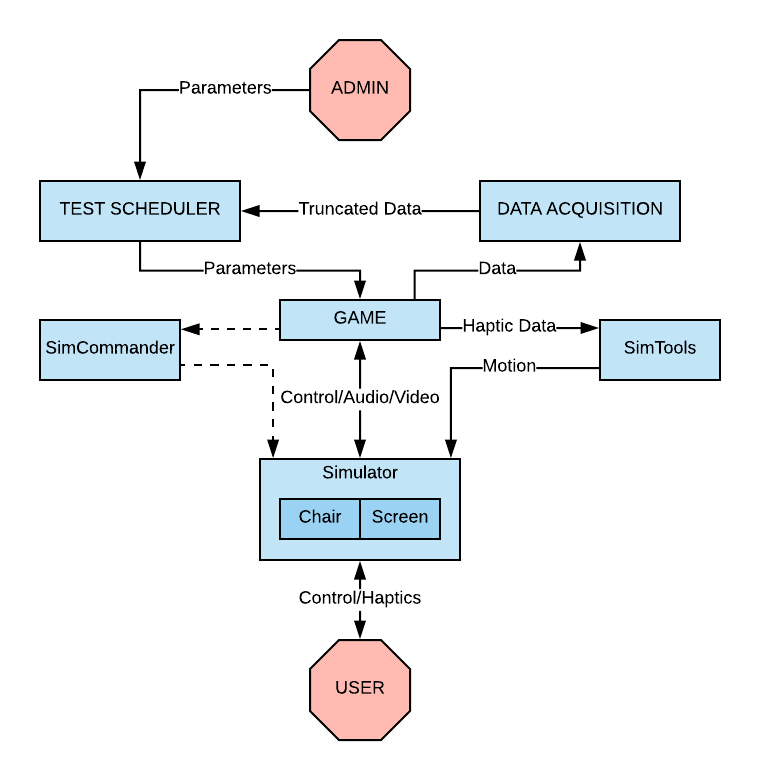
Regardless of whether the session constitutes a test or a training scenario, the user will be instructed to perform a sequence of driving maneuvers using the steering wheel and pedals as they would in a real car. Users will be expected to safely navigate a series of roads, adhering to realistic speed limits, stopping at stop signs, and negotiating traffic. Physics, logic, and graphics processing will be handled by the top-level user software, and data about user performance will accessed by the telemetry functionality through the top-level user software’s API.

The telemetry functionality will record information about the user’s performance to a file for analysis and feedback. For training scenarios, the data gathered will be evaluated and used to provide feedback on what the user should concentrate on improving. For testing scenarios, the data gathered will be used as a yardstick for the user’s ongoing progress.

* 1. Interface Design

The following sections provide visual diagrams detailing the how to the software will interface with the hardware.

* + 1. Interface Identification and Diagram



* + 1. Project Interactions:

The users will create individual accounts, or user profiles, within the experiment scheduler. These profiles will log their individual progress on an ongoing basis. The experiment scheduler will direct the functionality of the top-level user software by passing it command line parameters.

The maps the users will practice on will be generated from the top-level software itself. The top-level software has functionality that provides the user with feedback both in real time and after the trial. Real-time information is shown to users as they drive via functions included on the dashboard/HUD. A results screen with relevant data from the trial is displayed after the trial experiment.

Data from the user’s trial will be recorded by the telemetry/data gathering software interacting with the top-level software via its API. This data will be made available to the scheduler, which will log it in the user’s profile.

1. **CSC and CSU Descriptions**

Our application will implement four CSC’s: Experiment Scheduler, Training Scenarios, Test Scenarios, Data Acquisition. Building this software will require use of multiple APIs, preexisting game modifications, and custom software.

* 1. Class Descriptions
     1. Telemetry subsystem

Telemetry data gathering in general is implemented as one or more DLL plugins to the top-level user software’s telemetry API. One such DLL (described below) has currently been implemented in C++, but is subject to change and refinement. The team reserves the right to add others.

* + - 1. Fields
         1. FILE \*log\_file // File handle for log
         2. bool output\_paused // Logging starts paused, and can pause under experiment proctor’s control
         3. *struct* telemetry\_state\_t telemetry

scs\_value\_dplacement\_t placement // Position and orientation of vehicle

scs\_value\_fplacement\_t offset // Offset of vehicle from default position

* + - 1. Methods
         1. bool init\_log() // Initialize the log file
         2. void finish\_log() // Close the log file after writing
         3. void log\_print(char\* text) // Utility function for writing arbitrary text to log
         4. void log\_line(char\* text) // Utility function for writing entire lines to log
         5. void telemetry\_frame\_end() // Handles telemetry events, calls log writing
         6. void telemetry\_pause(scs\_event\_t event) // Pauses telemetry
         7. SCS\_RESULT scs\_telemetry\_init(scs\_u32\_t version, scs\_telemetry\_init\_params\_t params) // Initializes the telemetry API
         8. void scs\_telemetry\_shutdown() // Deinitializes the telemetry API
    1. Experiment Scheduler

The experiment scheduler will be implemented as a web app. This software will use provided arguments and the command line to initiate sessions.

* + - 1. Experiment Scheduler front end will be simple, and will incorporate drop down menus to select parameters.
      2. Scheduler will initiate sessions via the command line
      3. Scheduler will correlate acquired data to experiment subject.
  1. Detailed Interface Descriptions
     + 1. Telemetry API

SCS Software, the vendor for one of the pieces of MOTS top-level user software used in experiments, exposes an API for telemetry data.

* + - * 1. Implementation

The API is implemented by DLLs that the software initializes and loads at startup.

* + - * 1. Function signatures

Functions that the API implements are written as C header files.

scssdk.h

scssdk\_telemetry.h

scssdk\_telemetry\_channel.h

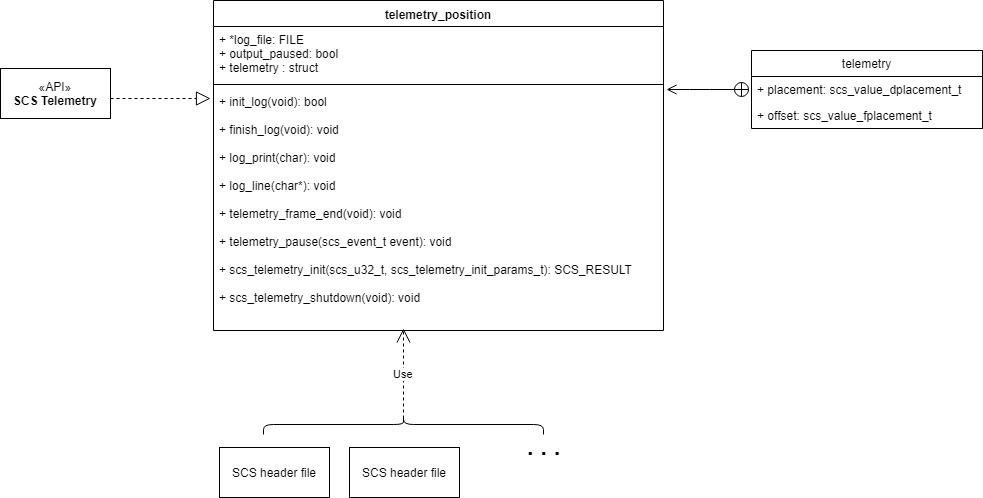
scssdk\_telemetry\_event.h

scssdk\_value.h

* + - 1. Experiment Scheduler/Top-level user software
         1. Software will send truncated data to two HDDs.
         2. Scheduler will take arguments that define the user session type and initiate sessions through the command line
         3. The command line will directly interface with the game through code initiated mouse moves and clicks
  1. Detailed Data Structure Descriptions

Data will be exported in a spreadsheet-like form. Each session will generate a TXT file of comma-separated value telemetry data, accompanied by a screen recording of the session. Data will be analyzed post-session by the customer. Training/Test Scenarios will be an object passed to the software via the command line.

* 1. Detailed Design Diagrams
     1. Telemetry System Class Diagram



1. **Database Design & Description**

Our customer has specified a desired format for research and training data, and videos of user trials. To that end, for the time being, he wants data to be sorted manually and stored in a directory structure on 2 large redundant external HDDs. There remains a strong possibility that further scale will necessitate implementation of a “real” DBMS, but the customer does not currently believe it to be necessary or important.

1. **Preliminary User Manual**

The user will be buckled into the simulator’s seat with no direct control over software functionality. The software will be started and controlled by the test administrator.

Training sessions and tests will be initiated from the Experiment Scheduler we build, implementing the supplied parameters. The user is required to supply throttle, brake, and steering inputs as required by the parameters of the particular session.

Details on Scheduler usage TBD—test administrator will have scenarios to choose from and parameters to set.