Code examples

In the workspace of Brownie from the main branch, the BrowserExperiment can be used to create a session for a specified number of users for a specific duration.  
<https://bitbucket.org/kit-iism/experimenttool/wiki/Home>

public class BrowserExperimentInitialScreen extends Screen {

/\*\* duration of the screen shown in milliseconds \*/  
private static final int DURATION\_OF\_SCREEN = 120000;  
/\*\* The Constant serialVersionUID. \*/  
private static final long serialVersionUID = -5532903241165693886L;

/\*\*  
\* Instantiates a new browser experiment initial screen.  
\*  
\* @param gameId  
\* A String that represents the ID of the game.  
\* @param parameter  
\* A ParamObject which contains screen parameters.  
\* @param screenId  
\* A String which contains the ID of the screen.  
\* @param showUpTime  
\* A Long variable which indicates how long the screen is shown  
\* to the client.  
\* @throws RemoteException  
\*/

public BrowserExperimentInitialScreen(String gameId, ParamObject parameter, String screenId, Long showUpTime) {  
super(gameId, parameter, screenId, 30L);

/\* Create WebBrowser component with user input logging \*/  
// WebBrowser browser = new WebBrowser(true);  
/\* Create WebBrowser component without user input logging \*/  
WebBrowser browser = new WebBrowser(true);  
Rectangle bounds = MainFrame.getCurrentScreen().getBounds();  
browser.setBounds(bounds);

/\* Set screen layout to null => full screen web browser \*/  
setLayout(null);  
add(browser);

/\* Load initial web page \*/  
browser.loadURL("https://www.opitz-consulting.com/index.html");

/\*  
\* Send client respone to end the experiment after a certain amount of  
\* time  
\*/  
new java.util.Timer().schedule(new java.util.TimerTask() {  
@Override  
public void run() {  
 ClientGuiController.getInstance().sendClientResponse (parameter, gameId, screenId);  
}  
}, DURATION\_OF\_SCREEN);  
}

}

The above experiment runs the home page of Opitz Consulting on the users for 2 minutes, and generates a csv file of their mouse tracking coordinates.

In RStudio, the output of these can be analyzed and plotted, as shown below.

install.packages("devtools")  
install.packages("webshot")  
install\_phantomjs(version = "2.1.1",  
baseURL = "https://github.com/wch/webshot/releases/download/v0.3.1/")  
install\_github("Fiddleman/BrownieR")

library(devtools)  
library(ggplot2)

# Check java\_home path in system variables. check if jvm.dll is present on path  
# check if 64 bit versions of RStudio installed.  
Sys.setenv(JAVA\_HOME="C:\\Program Files\\Java\\jre1.8.0\_181\\bin\\server")  
Sys.setenv(PATH="C:\\Program Files\\R\\R-3.3.3\\bin\\x64;..........;C:\\Program Files\\Java\\jre1.8.0\_181\\bin\\server")

library (rJava)  
library (NeuroIS)  
library (webshot)

# import path where csv files generated are placed  
import (path="C:/Users/ahr/Documents/experimenttool/Exp\_Implementation", prefix="test",create\_physio\_object=T,physio\_mv\_dir=T)

# Output browsing durations on each url  
summary(data =test\_web, objectives = F)  
web\_summary <- summary(data = test\_web, objectives =F)  
urls <- web\_summary$URL

# take screenshots of URL  
take\_screenshot(urls = urls)

# iterate and generate plots  
for (k in 1:length(urls)) {  
 plot(test\_web, url = urls[k], type = "motion",  
 subject = 1,  
 alpha = 0.1, size = 3, color = "purple")  
}