Group 304 - Sarah Hierstätter, Maja Pedersen, Bjørn Winther, Patrick Andersen, Mikkel Krarup, Anna-Lien Juelsbak

# SCRUM

## Scrum team:

Owner: Group 304
Development: All group members
Scrum master: Maja (still included in the development team)

## Product backlog:

- Supports at least 3 clients using 1 server (sprint 1)
- Hosted on GitHub with two repositories (one for the client and one for the server) (sprint 1)
- Written in Java (sprint 1)
- Create UML diagrams prior (sprint 1)

- The server uses multithreading to connect to several clients (sprint 2)
- Receives input from the client (sprint 2)
- Create Data input and Data output stream to connect client and server and send information between them (sprint 2)
- Use Sockets to connect the client to server using the same port (sprint 2)
- Separate workspaces for client and server (sprint 2)

- Gives output/ feedback to the client (sprint 3)
- Will assign a player number to the client (sprint 3)
- Separate interface to store all constants (sprint 2&3)
- Use the built-in Scanner class to get user input in the console (sprint 3)

- Finalise UML diagram post creating the game (sprint 4)
- Will save clients' scores continuously (sprint 4)
- Possibility to play the game again (sprint 4)
- Create a README document (sprint 4)
- Will have a lobby (sprint 4)
- Allow the game to be played from different computers whilst connected to the same server (sprint 4)

## Daily scrum:

We will use 15 minutes at the beginning of the day to discuss the plan for the day and delegate tasks. This 15 minutes will also be used to reflect on the progress towards the Sprint goal.

## Sprint 1: Setting up the project
## 06/09/19 - 12/09/19

**Sprint goal:**

- Decide on topic
- Github setup
- Research about client and server
- Preliminary UML diagrams of the system

**Sprint backlog:**
In this sprint, the goal is to decide on the topic of the mini-project by researching different opportunities and look at the given options/examples from the lecture. After the topic decision, we want to set up GitHub, create the two repositories for the server and client and maybe start implementing the first classes. Before we can start with the implementation, we need to research exactly what a client and a server are and how they work together. As soon as we find the information of what we need for a client-server application and what our game will be, we will make a first attempt of a class, use case and sequence diagrams.

**Sprint review:**
Done:

- Topic choice: Rock, paper, scissors game with multiple players
- Github set up with two repositories
- Research
    - A socket is needed to create a connection between the client and the server
    - The server needs to use multithreading to connect to multiple clients at a time
    - Use localhost instead of IP address if all the clients are on the same computer
    - Server and clients need to run on the same port
- Class diagram, use case, sequence

## Sprint 2: Client and server connection
## 12/09/19 - 03/10/19

**Sprint goal:**

- Create server and client classes in different workspaces
- Create a connection between them using a Socket
- Set up data streams to send information
- Create a thread in the server
- Create a handler class
- Create an interface containing all the constants

**Sprint backlog:**
We will use the second sprint to create different server and client classes in different workspaces using our gained knowledge from sprint 1. In order to create a connection between the clients with the server, we need to create a Socket. Furthermore, input and output Datastreams will be created to send information back and forth from the clients to the server. The server will create a new Thread that invokes a new Handler. The handler class will contain all the calculations and functions that will make the game work. A separate

interface will contain all the constants used in the game, in which the values will never change.

**Sprint review:**
Done:
- The different workspaces for the server and client worked and the program runs
- We created two Sockets for each player to both be able to connect to the server
- We created two data streams, one for the output and one for the input
- A multithread in the server class allows several clients to connect to the same server at the same time
- The server creates a new Thread that invokes a new Handler that takes two players.
    - Every time there are two new players joining the game, a new thread will be created
- The separate handler class will contain all the game data and send them to the server which will send them to the client

Not done:
- Implementation of the constant class is only halfway done as we are not entirely sure what else will be needed as a constant

## Sprint 3: Game
## 03/10/19 - 24/10/19

**Sprint goal:**
- Finalising constants
- Create the game's functionality
    - Calculate which players win, lose or when it is a tie
    - Give feedback to the user about where they are in the system
    - Assign player numbers to the different clients
    - Use the built-in Scanner class to get user input in the console
    - The server needs to be able to read the users' input
    - Possibility to play the game again

**Sprint backlog:**
Sprint 3 is about creating the game's functionality, which includes calculating when a player wins, loses or when it is a tie (=same user input). The handler/server should always send feedback to the player so that they know which state they are in all the time. It is also very important that the players know their player numbers to avoid any confusion. There will be certain words or letters that the user should input into the console that will allow them to carry out certain actions. At the end of a round, the players should have the option to play again if they want.

**Sprint review:**
Done:
- Finalise the constants in a separate interface
- The game functionality works and the game can be played by two players (1v1) with several games running simultaneously
    - The player will be notified whether they are 'Player 1' or 'Player 2'

    - The player can write their desired choice of move in the console with “r” for rock, “p” for paper and “s” for scissors.
    - The player gets feedback on whether they have won, lost or if it is a tie
    - When the game is over, the players have the option to play the game again or terminate the application
    - The player always knows what state they are in

## Sprint 4: Finalising tasks
## 24/10/19 - 03/11/19

**Sprint goal:**

- Create a Lobby
- Enable playing the game online (using different computers)
- Finalise UML diagrams
- Create README-file
- Upload on GitHub

**Sprint backlog:**

In the last sprint, we have to make some final changes in the code like adding a lobby where there will be a welcome message and perhaps the rules presented to the player. We also want the players to have the option to play the game online by using the same network and IP-address. Once the code is completely done, we need to finalise the initial UML diagrams by adding the missing functions to them. In addition, a README-file containing all the necessary information for users on how to install, download and play the game will be written and added to the GitHub repository. The final step will then be to upload the project to GitHub.

**Sprint review:**

Done:

- A lobby containing user feedback like the rules, the current state and a repeat-function has been created
    - Once players run the game, they have the option to see the rules (press 1), start a game immediately (press 2) or terminate the application (press 3)
- The game can be played online, on several computers by all using the same IP-address instead of keyword “localhost”
- New, updated UML diagrams have been created and uploaded on GitHub.
- We wrote a Readme-file containing a step-by-step guide on the usage of the game
- The final step of the whole project is to upload the project to GitHub