Swank-Rats documentation

Thomas Gaida (thomas.gaida@students.fhv.at)
Stefan Lässer (stefan.laesser@students.fhv.at)
Johannes Schwendinger (johannes.schwendinger@students.fhv.at)
Johannes Wachter (johannes.wachter@students.fhv.at)
Michael Zangerle (michael.zangerle@students.fhv.at)

Abstract

Semester Project for course “S1 - Kopplung und Integration von heterogenen Systemen”.

Inhaltsverzeichnis

1 Introduction

1.1 Game Idea

Swank Rat is a rat fighter game. Two rats are trying to shoot each other with cheese. The rats are represented by robots which are controlled by two players. With a Camera over the Game-World can the software "see" where the rats are. In addition, the obstacles are detected over this camera. This obstacles are straight walls (e.g. wood slates with a red with a red mark). The Rats are able to throw pieces of cheese after the opponent. The walls serve as a limitation for the cheese-bullets.

To control the robots the live video of the world (overlaid with video of the cheese-bullets) will be displayed in a HTML UI in the browser. With buttons (and keyboard shortcuts) can the player control the real robot.

If a robot is hit (one or more) the game is over.

1.2 Architecture

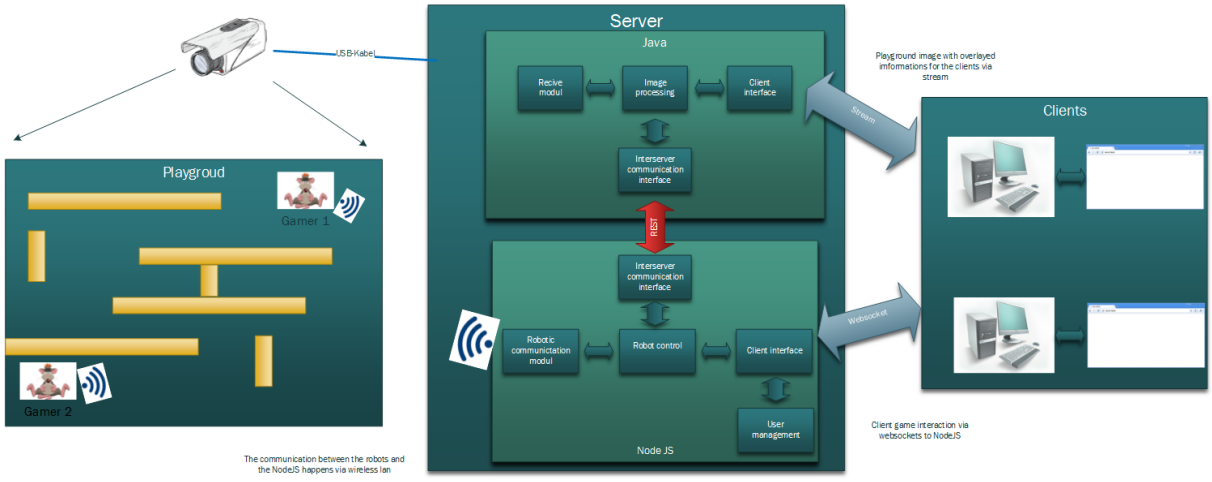

Abbildung 1.1: Architecture of Swank-Rats

1.2.1 Hardware

- 2 x "Rat-Robot" with WLAN Dongles to communicate with the server
- 1 x Webcamera (for the detection of position and world)
- 1 x Server (Notebook or PC for image processing and game logic)
- 2 x Clients (Notebooks with modern Browsers)

1.2.2 Server-Software

- Server Application (Java)
  - Image processing
  - Position detection
  - Overlay webcam video with cheese-bullets
  - Stream video for client
- NodeJS Server
  - Robot control
  - Server UI (HTML)
  - User management

1.2.3 Client

- Browser Application
  - HTML5
  - Presentation of game stream
  - Javascript with Websockets
  - Buttons to control robot
  - Login
  - ...

1.3 Communication

TODO

2 Image-Processing

2.1 Components

For the implementation of our image processing functionality we decided to use C++ in connection with OpenCV 2.4.9 (http://opencv.org/). It will help us to get the video stream of a web cam, to detect the position of the robots and to detect collisions (e.g. collision between robot and wall, but also collisions between a shot and a wall or robot).

For the communication between the NodeJS server and the image-processing server we decided to use WebSockets. To fulfil this purpose we use the library XXXXX (TODO!).

For compiling our source code we use the MinGW GCC C++ 4.8.1-4 compiler. This offers us the full support for C++11, including std:thread (this is missing in other Windows-versions of GCC). As IDE we use Eclipse Kepler SR 1.

2.2 Why WebSockets

2.3 Why OpenCV and C++

We did some research and searched for possible free image processing libraries. We decided to use OpenCV, because it offers the biggest amount of functionality compared to the other libraries which were available for free like SimpleCV, OpenCV for Java, OpenCV for .NET (Emgu CV) or JAI. We do not want to take the risk to use a library which offers less functionality and finally we may be faced with the problem, that a functionality that we need is missing.

First we thought about using Java together with the ported OpenCV version, but then we were a little bit afraid about possible performance issues, instability and the fact, that you have to use native method calls in your Java code to get access to the OpenCV functionality since it is written for C/C++. Also offers the wrapper for OpenCV under java less functionality then the original OpenCV for C++.

Another possibility would have been C#.NET together with EmguCV (http://www.emgu.com/wiki/inde
But EmguCV is also just a wrapper to the OpenCV library.

So we decided to use OpenCV in connection with C++. It is a little bit risky because we do not have much experience in using C++ and it is the first time we use it in a project. So we are looking forward to learn a lot of new things.

2.4 Requirements

2.4.1 Needed functionality

- Providing video stream for clients
- Collision detection
  - Robot/wall collision
  - Shot/robot collision
  - Robot/robot collision
  - Shot/wall collision
- Position detection of walls/robots
- Communication with NodeJS server
- Simulation of shoots in video stream

2.4.2 Communication with NodeJS server

It is necessary that the NodeJS server and the image-processing can talk with eachother. The communication is needed, because the NodeJS server has not enough knowledge to make all the game logic decision by its own.

The following messages can be sent: - Messages by image-processing server to the NodeJS server - If a collision was detected (see above which cases exist) - If a shot was made to notify if a robot was hit or not - Messages by NodeJS server to image-processing server - If a shot was make by a player - If game has stopped - If game has started - If game has paused (e.g. connection problems) - Info about which player has which robot

2.4.3 Video quality and resolution

For our project we use the webcam LifeCam HD-3000 from the manufacturer Microsoft. We decided to use 720p resolustion for the streaming. This allows us to provide the clients a gaming environment in a today acceptable resolution without too much traffic through the transmission. Therefore the system need following requirements - Intel Dual Core 3.0 GHz or higher - 2 GB of RAM - 1.5 GB - USB 2.0 required

The maximal resulustion for motion video is 1280 X 720 pixel for still image 1280 X 800. The webcam has a maximal image rate up to 30 frames per second and a 68.5 degree diagonal field of view. The other image features of the webcam are Digital pan, digital tilt, vertical tilt, swivel pan, and 4x digital zoom - Fixed focus from 0.3m to 1.5m - True Color - Automatic image adjustment with manual override - 16:9 widescreen - 24-bit color depth

2.5 Architecture

TODO Architecure diagram #Requirement analysis for client and server

We defined a platform independent implementation as a general requirement for the client and the game-server. Furthermore we wanted to keep the possibility to play our game also via a tablet or even a smartphone and to keep it also extensible. Last but not least we wanted to use new, state of the art web-technologies for the sake of the web and for our continuing education.

2.6 Specific requirements for the client:

- clean and simple user interface
- the user interface should be responsive by default
- a login / ranking page should be visible for logged in users
- a welcome / introduction / registration page should be visible for all users
- a permanent connection to the server for fast and efficient transfer of game related commands should exist
- the client should be able to display the game universe with it's players and their interactions via a stream

2.7 Specific requirements for the server:

- the server should be able to communicate in an easy/efficient/fast way with the robots, the image processing unit and also the clients (bidirectional communication)
- the server should be able to communicate with a database to persist game results and also player specific data e.g. the user-/player-name, password
- the server should provide a fast webserver which also supports SSL for basic security
- the server has to provide interfaces for the client to process registrations, logins, page-ranking requests and normal page requests as well as game-specific commands

- the server should be able to communicate with the image processing component (more details here
- FEATURE: game unrelated users with a smartphone should able to watch the current game on their smartphones. This means that these users should film the game from defined positions with their smartphones and another image will be used as overlay to display player interactions and more

3 Used technologies

The technologies we used for this project for the client and the game-server is the mean stack.

3.1 MEAN-Stack

This stack consists of four different software components which work very smooth with each other. These components are the MongoDB which is a database, the and Express the Angular.js Javascript frameworks and last but not least the Node.js environment.

3.1.1 MongoDB

MongoDB (from "humongous") is an open-source document database, and the leading NoSQL database. It's main key features are document-oriented storage, full index support, replication and high availability, auto sharding, map/reduce support, in place updates, and a lot more. You can find more information about the database on their website.

3.1.2 Express

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications without obscuring Node.js features that you know and love. You can find more information about the framework on their website

3.1.3 Angular.js

Angular is an open source framework from Google and helps to make and structure single-page web applications according to the MVC pattern. Angular.js operates entirely on the client side. You can find more information about the framework on their website

3.1.4 Node.js

Node.js is an open source, cross-platform runtime environment for Javascript applications. Node.js provides an event-driven, non-blocking I/O model that makes perfect for data-intensive real-time applications. Internally Node.js uses Google V8 Javascript engine which is also used in the Chrome Browsers to execute the applications. A lot of the environment is also written in Javascript and it provides modules for file, socket and HTTP communication which allows it to act as a web server. Popular companies which use Node.js are for example SAP, LinkedIn, Microsoft, Yahoo, Walmart and PayPal. You can find more information about the framework on their website