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

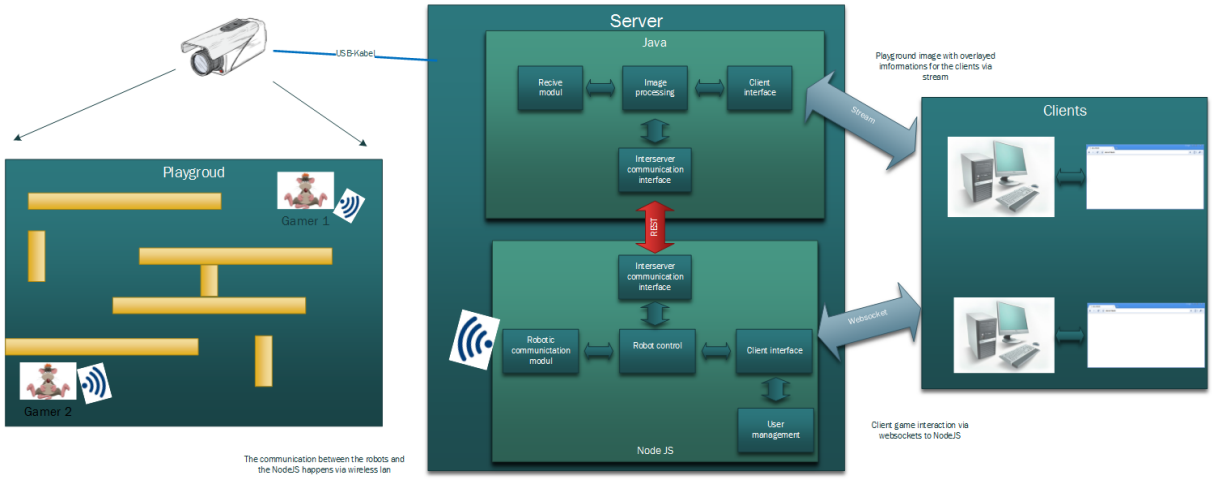

Abbildung 1.1: Architecture of Swank-Rats

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/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. 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++.

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. ## Requirements

2.3.1 Needed functionality

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

2.3.2 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

2.3.3 Video quality and resolution

2.4 Architecture

TODO Architecure diagram