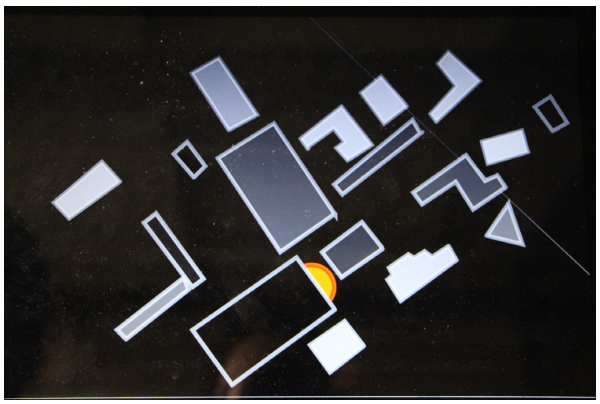
Technical overview

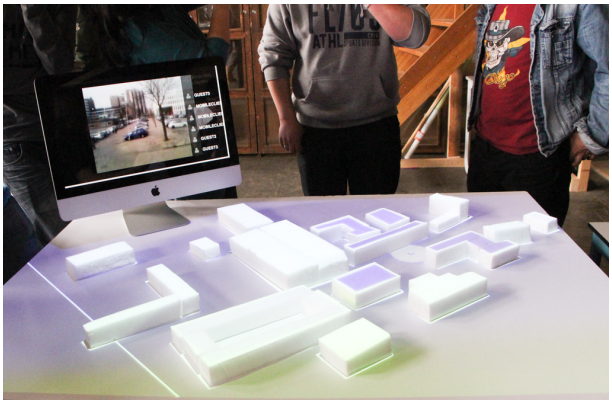
Co-location engine

Tracking different positions of devices on the street
Tracking will be done using GPS from Android devices. The devices will be running a native Android app with Pure Data integration for sound mapping and modulation. Interface will be developed in native code and Pure Data using socket communication with the servers through socket.io. There will be pre-installed devices available for runners. Other players can join (or be recruited to) the game by installing the application on their own device or possibly by interacting through social networks. Most of the interface will be conveyed through dynamic sound that will be controlled and generated by the Pure Data module. All players should be wearing headphones.

Detecting co-presence and handling game logic
Real-time distance measurements between all the points, game logic, events and communication is accomplished by a server running Nodejs.

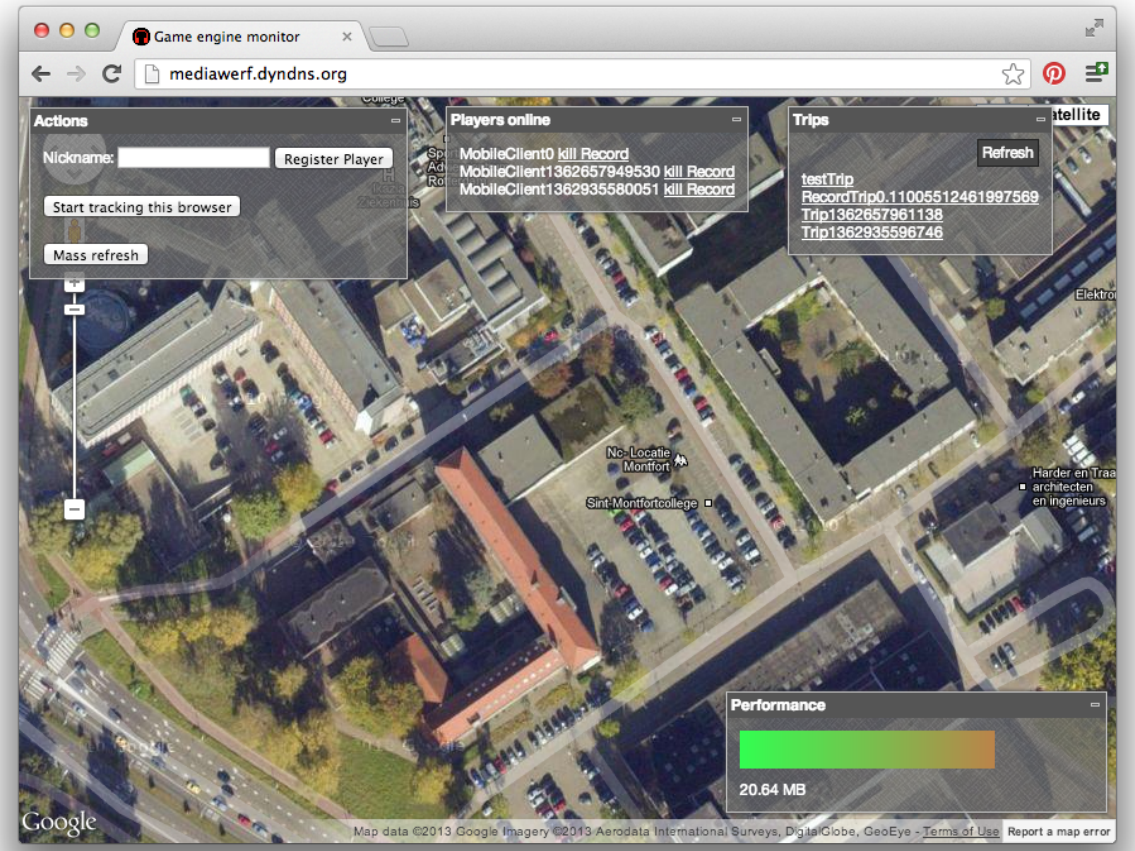
Storing / logging and displaying/replaying paths
MongoDB will be used to store player data and recorded paths. Recorded paths can be replayed and analysed.
The system will be able to display current players positions and actions through the web and on a custom interface at the Rotterdam library These interfaces will be developed using HTML5/CSS3/JS/Processing. Processing sketches will be run in a javascript environment using processing.js.
Library and online players will be able to play a role similar to the street players by acting as geolocated "ghosts". They will also have a sound intensive experience through headphones along with a visual representation of the game area and all the other players positions and actions.





Serving content
Static content such as files and client-side frameworks will be served separately from the game data by an Apache server.

Scaling / Monitoring
Automated monitor programs and a monitoring web page will be available to monitor live system data such as load on the game engine, errors, memory alerts, load balancing, current game information and server crash recovery. These will be written in python and javascript.
At the moment we have a basic monitoring system to see who is logged in the game, how much resources are been taken and perfom some tests adding / removing players and replaying previous trips.



The game engine is built with the possibility to add extra instances of Nodejs and keep all instances synchronized in terms of the game live data. All client applications can connect to a load balancer that forwards them to one of the available servers. These Nodejs servers can be deployed dynamically from the load balancer. This means, for example: There is only one instance of the server running, a lot of clients connect to it, the memory usage or response time

starts to increase so the load balancer decides to launch a new instance of the Nodejs server to cope with the high volume of traffic.

Resources

- 10 Alcatel smartphones with 3G SIM cards
- Hosted server
- 4(+?) Library terminals
- Custom made headphones
- Game area maquete for projection
- Beamer
- Gamepads or custom physical interfaces for the Library players

System architecture

Game engine

Server side:

Apache serves the client side framework (HTML / CSS / JS / Images / Sounds / videos / Android app) to the arriving clients.
The **Nodejs server** accepts and manages a group of socket.io connections comming from clients. Clients can register themselves as players on the game and start sending location updates. Location updates will be broadcast to all connected sockets.
The server will detect proximity and send "in range" events to all clients. Distance calculations are asynchronous and can be deferred to a parallel worker instance of Nodejs.
Mongodb is used to store persistent information such as saved trips, user data and logs. Any information that does not need to be persistent will not be sent nor stored on the database.

Client side:

The **client game engine** consists of a single-file javascript and a single Java class include (For native android clients) that encapsulates and abstracts all the necessary functionality / data model management and synchronization, communication protocols with the server and client position detection and update.

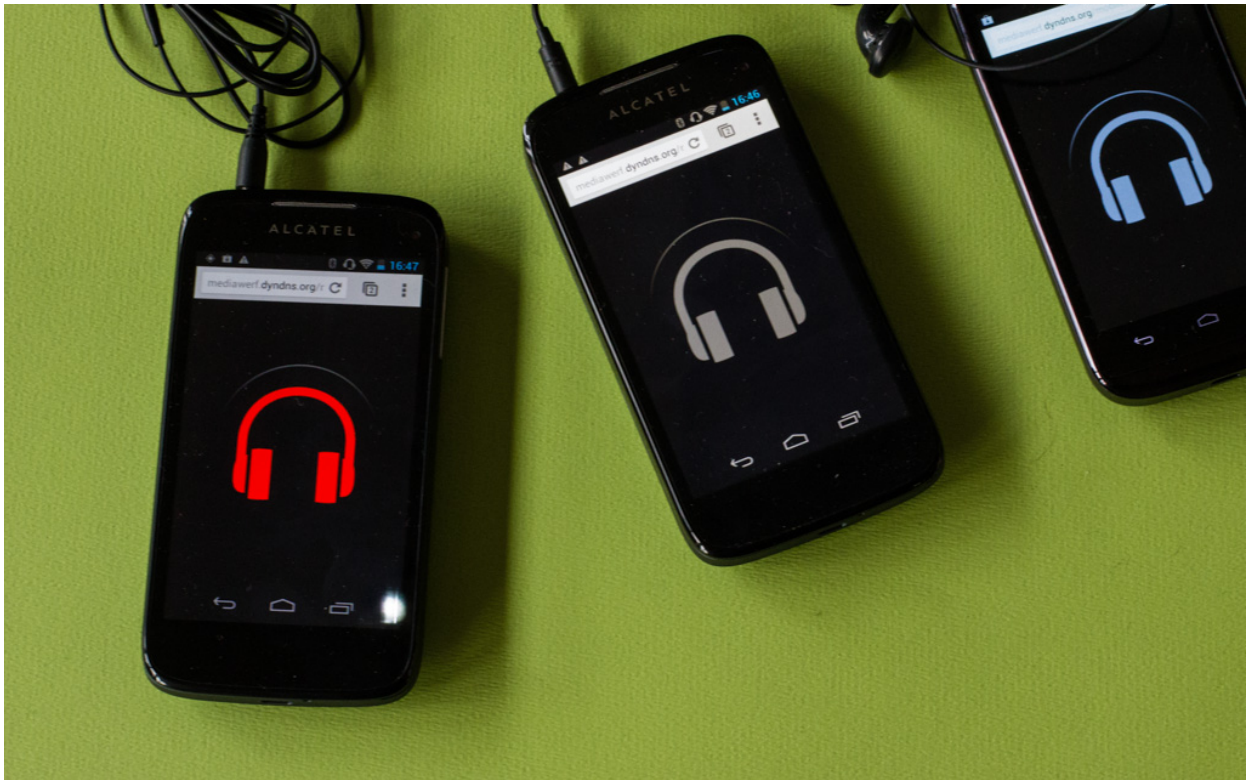
Aswering Nick’s questions

Exciting/interesting discoveries - either experiences or software!

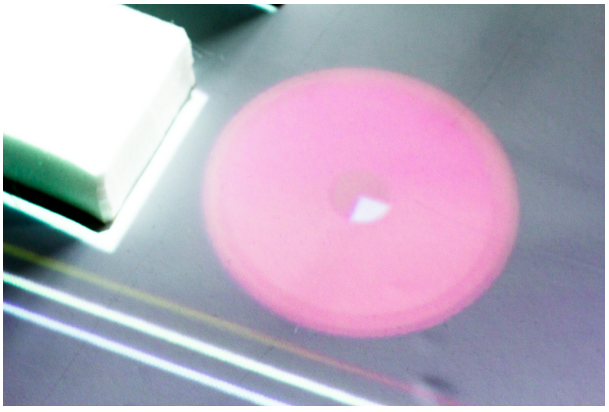
- The combination of NodeJS and socket.io seems to work pretty well for realtime data.
- HTML5 is really powerful and allows us to modularize the game pretty easily in the sense of having multiple screens / devices connected to the same system.

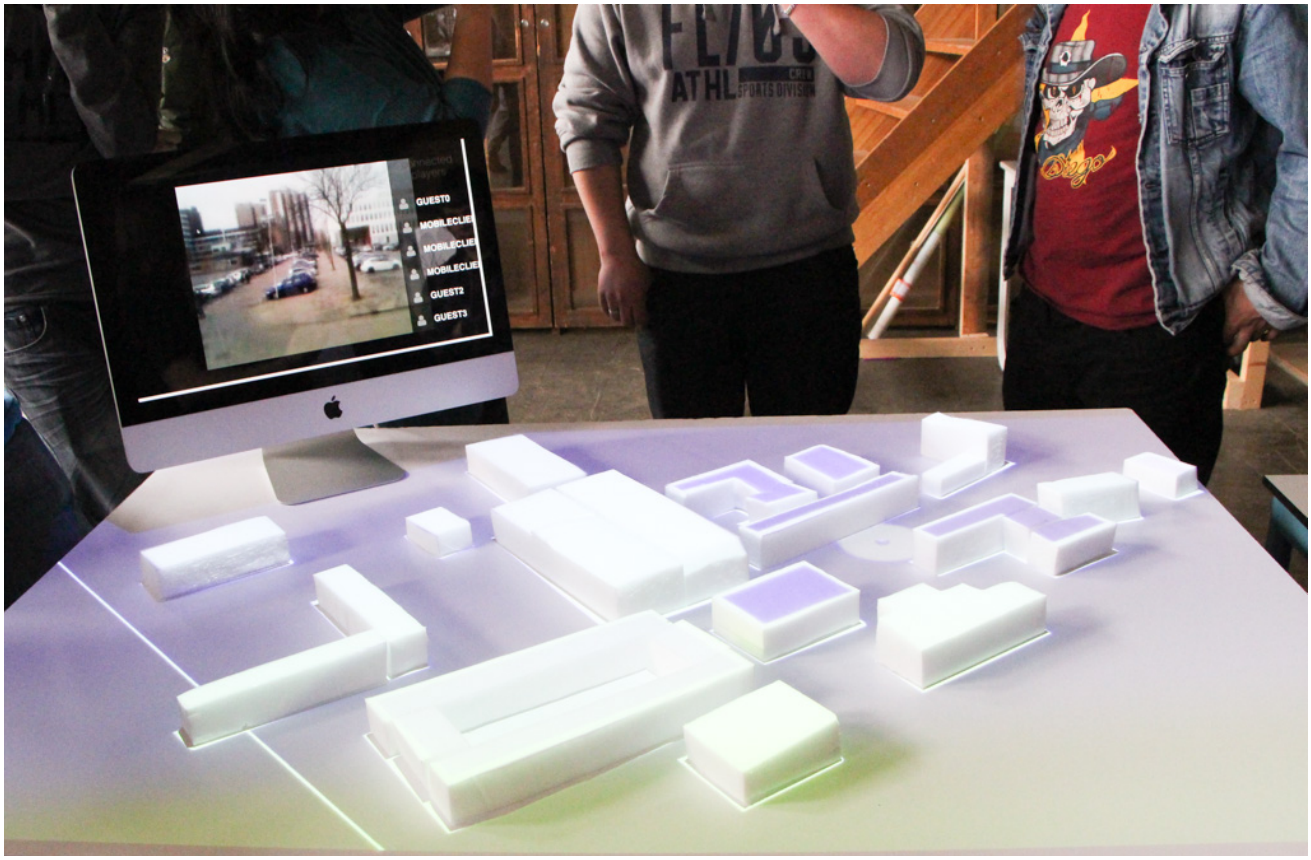
- We can connect any type of devices (game controllers, arduinos, etc) to the HTML5 part using OSC protocol to a server with NodeJS and then from the server to the web using web sockets.

Screengrabs and photos to see what we're each doing




INFORMATIE
ONTWIKKELING
INFO
VERHUUR





Current challenges and problems

- The advance sound engine using Pure Data is not yet integrated. We have some experience using Pure Data in Android but we might have some little problems with Android since the audio latency is pretty big.

- Still we have to find a way for not having to rely on accurate positions with the GPS. We are using halos / nimbuses to minimize the error. Still is a challenge to solve.

Practical resources, information or learning: eg. 3rd party platforms we're using, limitations we've found.

stackoverflow.com is your friend :)