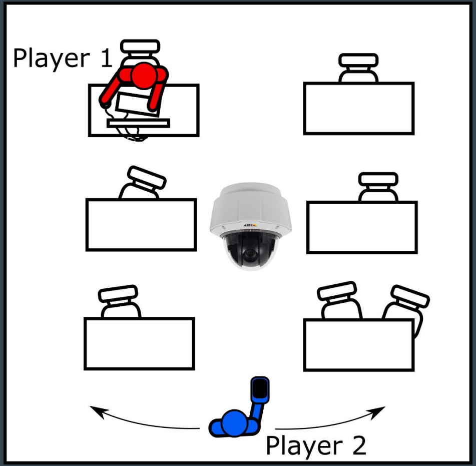
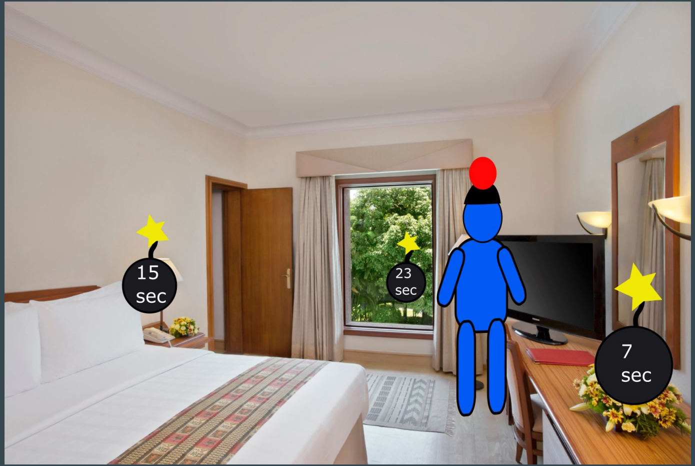
Alexander Baldwin
Simon Gullstrand
Björn Hansson
Johan Holmberg
Jonas Wahlfrid

Asymmetric Augmented Reality Game

What Did We Do?

- Two players must work together to defuse bombs before they explode.
- One player controls the pan of a Axis Q6045-E camera and can see bombs overlaid on the image in an augmented reality interface.
- Uses context awareness in the form of the second player's location to determine when a bomb can be defused.



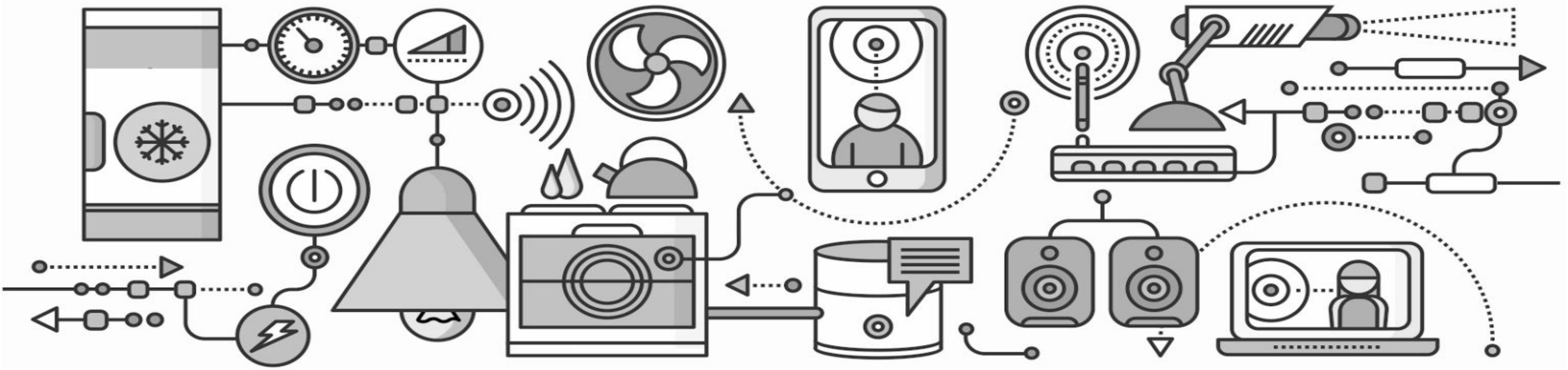


Demo

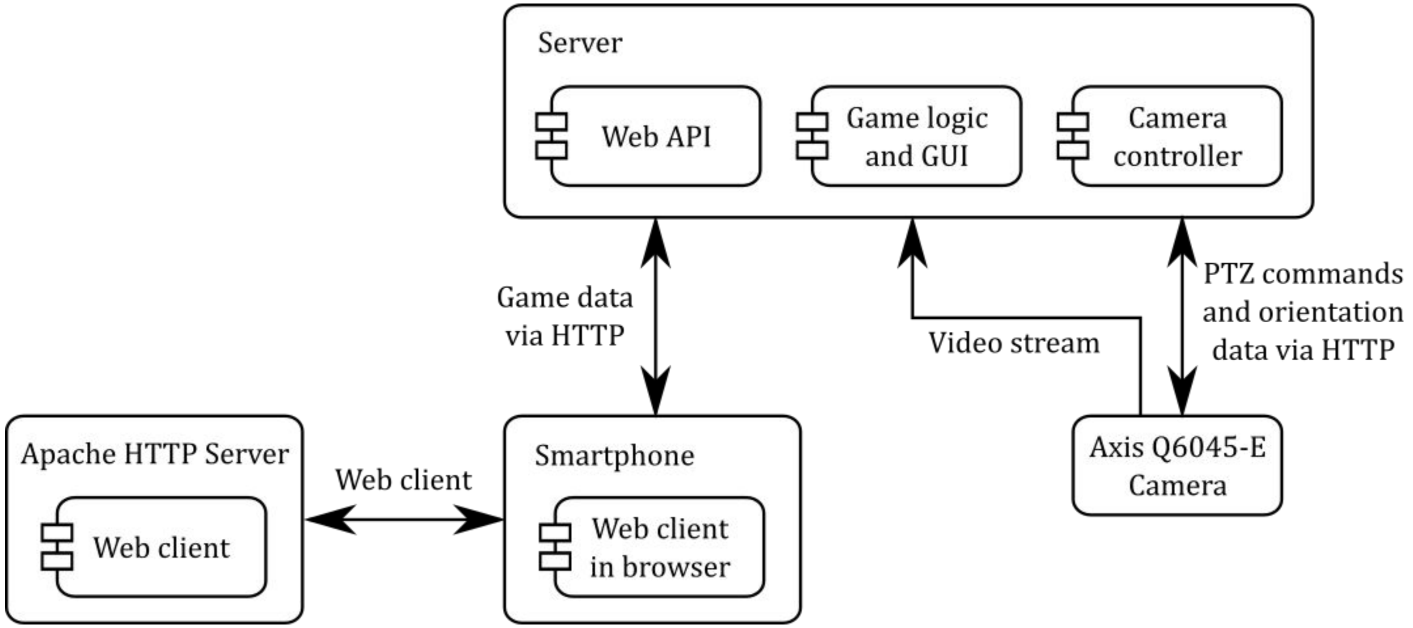
https://www.youtube.com/watch?v=H2Q0k6yuxPE

Why did we do it?

- Multiple communicating devices.
- Make use of the Axis camera's PTZ functionality.
- Web service-based architecture.
  - REST API.
- Both augmented reality and context awareness are related to IoT.



How did we do it?



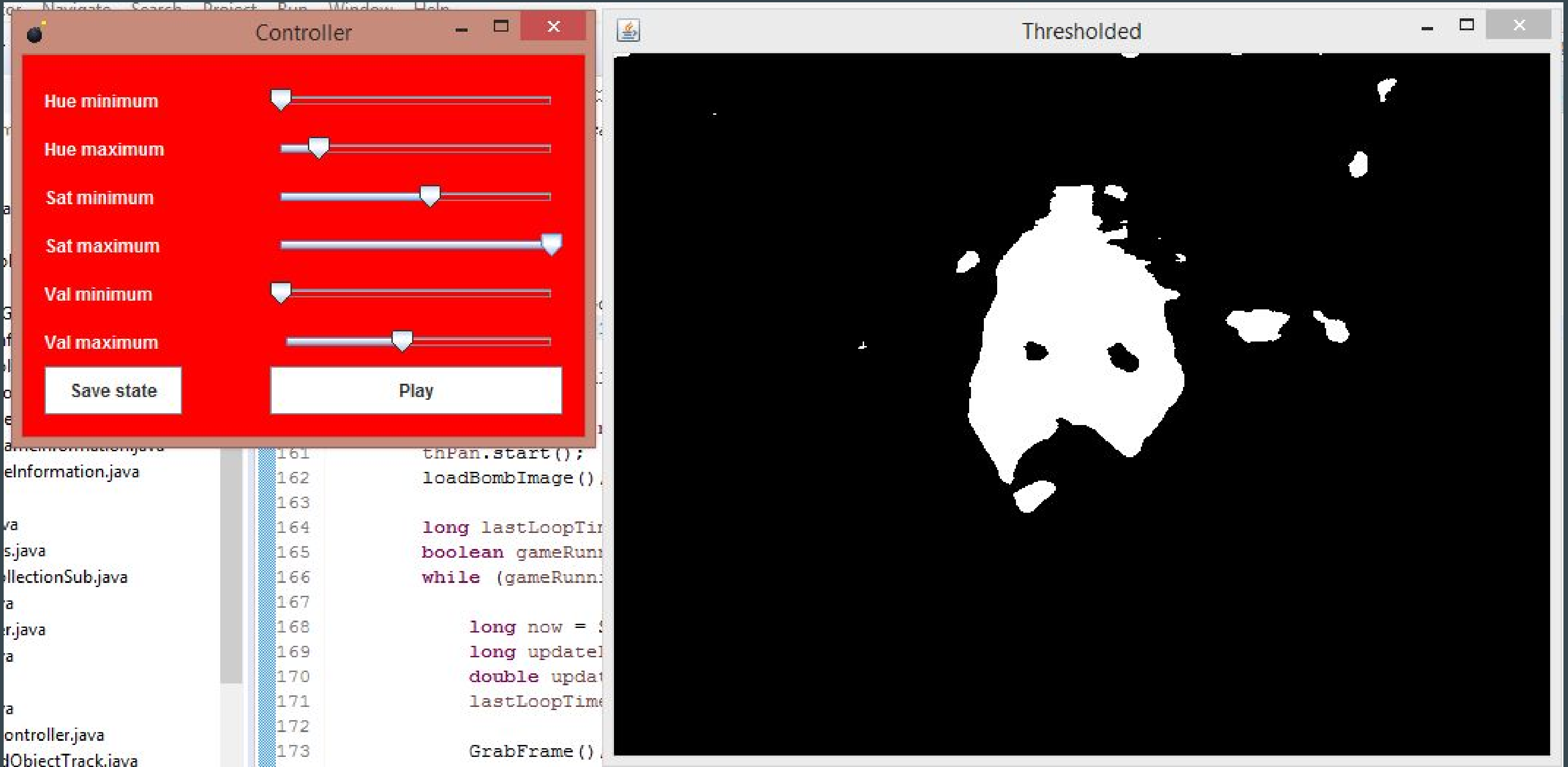
REST Web Service

- A RESTful, self-describing, web service
  - All functionality mapped onto resources, manipulated through the use of HTTP verbs
  - Uses JSON as data exchange format
- Simplifies the concurrent development of components in a distributed system
  - Easy to use and understand
  - Promotes heterogeneous environment setups
- Implemented in Java using Spark, documented using Apiary

Position Tracking

- Challenge: figure out where player two is in the video image so we know if he/she is close enough to defuse a bomb.
- Solution? JavaCV (wrapper for OpenCV) and brightly coloured clothes!
- How else could we have done this?
  - More advanced techniques exist that track a person’s skeleton (see Kinect, LeapMotion etc.).
  - Axis provides the Digital Autotracking API, but unfortunately this is for automatically moving the camera: not what we needed!

Position Tracking



Hue minimum

Hue maximum

Sat minimum

Sat maximum

Val minimum

Val maximum

Save state

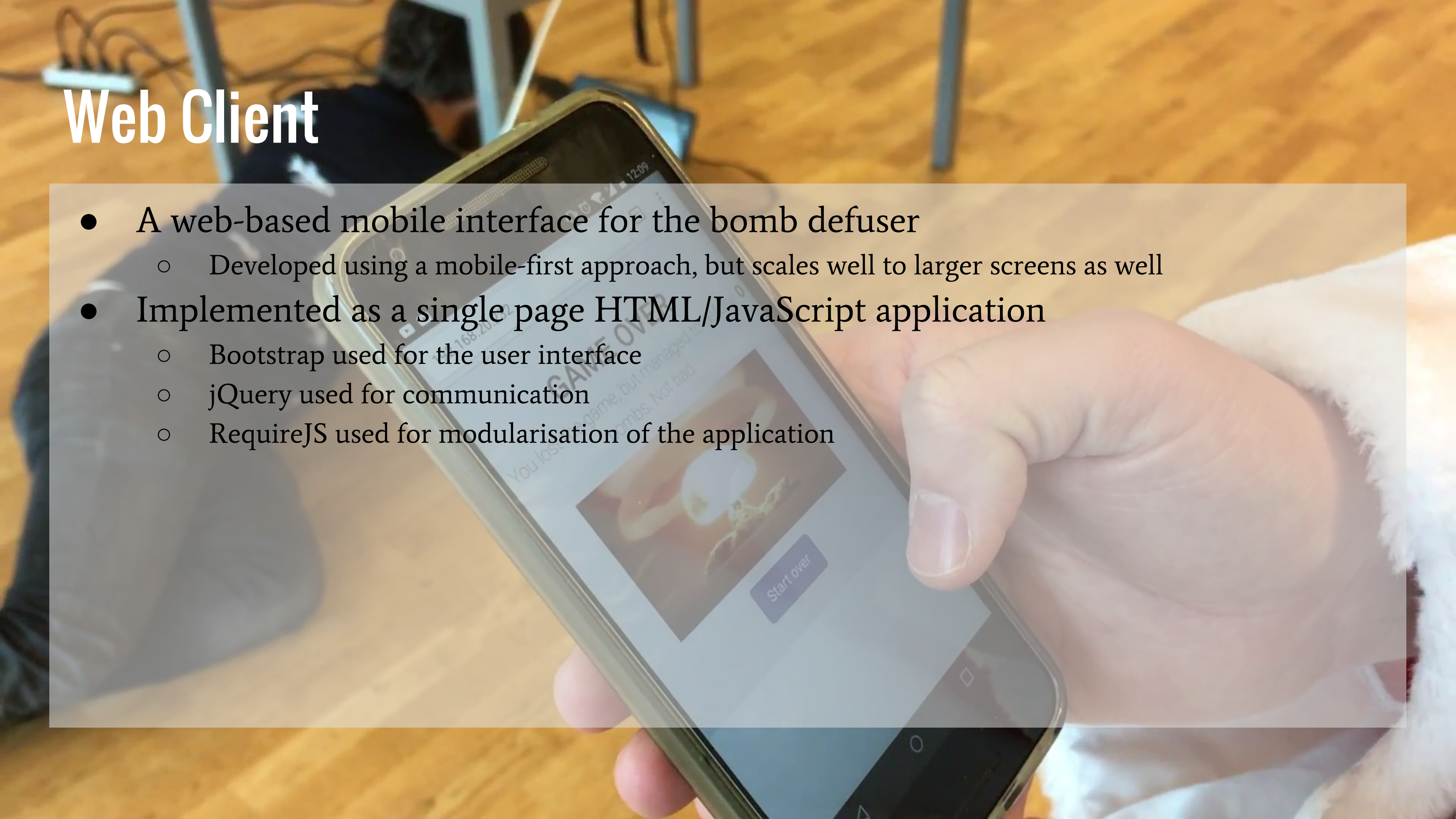
Play

```
elnformation.java
va
s.java
llectionSub.java
a
r.java
a
a
ontroller.java
dObjectTrack.java
```

```
        thPan.start();
        loadBombImage().

        long lastLoopTi
        boolean gameRunn
        while (gameRunn

            long now = 
            long updatel
            double updat
            lastLoopTime

            GrabFrame().
```

Camera Controls

- The camera starts to pan when the left and right arrow keys are pressed, by means of HTTP requests. Key repeat are filtered out. Similar requests stop the camera movement when the arrow keys are released.
- The controller continually requests the camera’s orientation in order to keep it updated in the application so we know where the bombs should be positioned in relation to the video image.
- Communication with the camera was realised using the Unirest library for sending HTTP requests.



Web Client

- A web-based mobile interface for the bomb defuser
  - Developed using a mobile-first approach, but scales well to larger screens as well
- Implemented as a single page HTML/JavaScript application
  - Bootstrap used for the user interface
  - jQuery used for communication
  - RequireJS used for modularisation of the application

Challenges

- Synchronising the camera’s orientation and video stream for placing bombs.
- Getting JavaCV to cooperate!
- Smooth camera movement (pan vs continuous pan).

Thanks!