Jeopardy on a Raspberry Pi

Introduction

„Jeopardy! is a TV quiz show, in which the correct question for an given ansewr has to be found. In other words the typical scheme of a quiz is inverted: questions and answer are asked and given in reversed order. Usually there are five different categories, each with five different levels. For the easiest answer (for which the corresponding question has to be found) one can win 100 points, for the most difficult one can win 500 points.

Why do we created this?

„Jeopardy on a Raspberry Pi“ was created for a scientific project. Its goal is to make current research understandable and attractive to the public. The way Jeopardy makes this possible is, that it can show the results of many fields of research and the task is to find the correct question, that was solved in the corresponding scientific contribution. To present the scientific result, our software makes it possible to make use of different type of media e.g. simple text, images or moving images (aka videos). Since each quiz consists of five different categories one can combine very different types of research (e.g. psychology or physics), what supports a interdisciplinary mindset.

Common terms in this context are „edutainment“, „technotainment“ and „gamification“. „Edutainment“ mainly connects eduction and entertainment. Its goal is to connect the fun of playing games with learning. (Buckingham und Scanlon (2000)). The term „technotainment“ can be understood as the interaction between technology and entertainment, although the technology is only used for assistance and not to directly influence the participants (McKenzie (2000)). The third superordinate term „gamification“, implies that something is connected to the game, such that it earns a direct connection to the context of the game An example for „gamification“ is the level concept of the gaming platform steam. The more a person spend playing games on the plotform, the higher the level or rank. So the steam level becomes a part of the game, what binds the customer to the platform. (Michael Sailer, Jan Hense, Heinz Mandl und MarkusKlevers (2013))

Manijeh Mistryund Safeya Al Baharna (2011) described an experiment, in which they observed that games (e.g. Jeopardy) can be a suitable possibility to repeat specific topics and explain difficult ideas in a playfully way before an exam. Therefore it makes sense to apply the concepts of „edutainment“ or „technotainment“ and make use of Jeopardy to transfer knowledge to the public.

Hardware

STL files for the cases of the buzzers and the Raspberry Pi 3B case can be found here.

Material List

- Raspberry Pi 3B
- cinch cable (one per buzzer)

- buzzer
- cinch connectors

Software

Installation

- We used the Raspberry Pi OS
- You need to install Node.js. We strongly recommend the latest version (>14.0)
- Clone the repository

```
git clone https://github.com/danielgolf/raspberrypi-jeopardy
```

- Install dependencies, build and start frontend

```
cd raspberrypi-jeopardy/frontend
npm install
npm run build
```

- Install dependencies and start backend

```
cd ../backend
npm install
npm start
```

Design

We implemented a frontend and a backend. The frontend can be used by different devices (e.g. smartphone, raspberry, ...). Its main purpose is to visualize the current state of the game. There have to be exactly one gamemaster, who can create a game by selection categories (containing the task of the quiz) and submitting the team names. All the other clients can visualize the current game, initialized by the gamemaster. All the functionality and logic of the game is implemented by the backend. It maintains the points, categories and tasks and the current state of the game. The different categories and their tasks are stored in the `categories` directory as json files. The connection between backend and frontend makes use of WebSockets.

Frontend

For the frontend we used

- React for the basic logic
- tailwind for the styling
- ReactRouter for the routing between different sub-pages
- Typescript for the convenience and clean code

Initially the react app was created with the „create react app“ scripts and never detached from it. So

`npm run build` builds our project in two steps:

1. It compiles the css of tailwindcss
2. Afterwards the assets are bundled to the deployment build by `react-scripts`

Basically there are five toplevel react components under `frontend/components/`, between the `ReactRouterDOM` switches

- `Home.tsx` this is the welcome screen with the navigation bar
- `viewer/Viewer.tsx` the viewer screen for the participants
- `Gamemaster.tsx` this is to register a gamemaster at the backend and encapsulate frontend functionality
- `game/Game.tsx` this is the viewer of the gamemaster, if there is already a running game
- `game/Game2.tsx` this is the page to create and initialize a new Jeopardy

Backend

The backend contains of different classes and files, listed below.

- `index.js` is the entry point, which maintains the WebSockets, the hardware buzzer and the game
- `CategoryReader.js` Reads the categories from the `categories` directory to offer them to the game and the frontend
- `category.js` Maintains a category
- `ws_server.js` Is the server for the WebSockets - Responsible for the communication between the different frontend clients (gamemaster and viewers)
- `jeopardy.js` Implementation of the logic of the game
- `gpio.js` Communication with the hardware buzzer
- `player.js` Was used to maintain a participant - currently obsolet

Usage

Hint: in the following you will see some development screenshots to visualize the concept. The layout has probably already been improved ;)

The home screen welcomes you to Jeopardy! You can either create a new game, continue a game as gamemaster or just view a running game (if you are a participant).

In the menu under „Start Game“ you can setup a new game. You are then automatically the gamemaster and nobody else can create another game. To start with the new game you have to choose the teamnames and select the categories to be included in the game. Currently for each level in each category one question is randomly selected.

After the game was started, the frontend switch to the overview of the current game. Here you have to distinguish between the view of the gamemaster and the view of the participants (viewers). The gamemaster is able to click a level of a category, while the view of the viewers is static and nothing

can be clicked. When the gamemaster have chosen a level of a category both, the gamemaster and the viewer view, switch to the visualization of the corresponding answer. Now the participants can use their buzzers and try to find the corresponding question.

Gamemaster: 

Viewer: 

The visualization of the answer contains the answer and the current points of the teams. The gamemaster view also contains a set of buttons to accept or reject an answer of a team after they used their buzzer. Furthermore he can discard a answer if nobody found the corresponding question.

Gamemaster: 

Viewer: 

Maintain Categories

The categories contains the levels and the question answer pairs. In the frontend view of the game, a category is visualized as a column and the selected question answer pairs of each level are the entries of the column. Categories are stored in the `categories` directory as json file `<CategoryName>.<Version>.json`. The `<CategoryName>` can be arbitrarily chosen and the `<Version>` has to be a positive integer value. The game always choose the newest version, that is the category file with the largest `<Version>` integer.

A category file looks as follows

```
{
  "name": "When took the first flight",
  "demo": true,
  "levels": {
    "100": [ ... ],
    "200": [ ... ],
    "300": [ ... ],
    "400": [ ... ],
    "500": [ ... ]
  }
}
```

It contains a name (string), a demo flag (boolean, if true this category can not be used in a game) and an object containing the different levels from 100 points to 500 points. Each level contains and a array of question answer pairs which looks as follows

```
"100": [
  {
    "type": "text",
    "text": "12 April 1961",
    "question": "When took the first flight in space with a human person
place?"
  },
```

```
  {
    "type": "image",
    "uri": "https://upload.wikimedia.org/wikipedia/commons/3/35/Tux.svg",
    "question": "Which company was founded by Linus Torvalds?"
  },
  ...
]
```

Each question answer pair contains a type (currently only „text“ or „image“), the answer (either a text or an URI to the image) and a question. The type tells the frontend if it has to just display a text or either to visualize an image, which can be found under an URI. For the gamemaster it is important to include the correct question, since the gamemaster has to decide wether an question of a participant is suitable to the given answer or not.

Roadmap

There are many possible features to be implemented. Below we listed a few, which came into our mind.

- [x] Basic implementation
- [x] Documentation
- [ ] Nice frontend to maintain the categories (backend functionality partially implemented)
- [ ] Include other types of media for answers (e.g. audio files)
- [ ] Usage of other digital devices (e.g. smartphones) as buzzers
- [ ] Further Quiz customization: not only choose categories, but also tasks per level of points