Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

WikiGame: Graph-Exploration, Fun included

Distributed Systems Laboratory

Filip Jaksic, Leonardo Salsi, Lorenzo Paleari, Patrice Delley

jaksicf@ethz.ch, salsil@ethz.ch, lpaleari@ethz.ch, pdelley@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Luca Lanzendörfer
Prof. Dr. Roger Wattenhofer

April 16, 2024

Abstract

This paper describes our work in which we translated the ruleset of Fibbage, a popular word-based game, such that it makes use of data stemming from Wikipedia but mirrored on ETH Zurich's infrastructure. Our aim is to gamify the graph-exploration of Wikipedia articles by introducing elements that make use of this infrastructure. Using a back-end implemented in Python and a front-end based on Next.js alongside with Tailwind, we created a small but fun experience.

Contents

Chapter 1

Introduction

Graph-Exploration can be exhausting, especially when dealing with a vast landscape of different data. Performing examinations on them can be tiring, and a break is deserved. With this in mind we present WikiGame, a project which makes use of the article data available from Wikipedia and takes inspiration from Jackbox Games' Fibbage[1].

Fibbage is a trivia game in which multiple parties aim to not only correctly provide an answer to a question, but also to deceive their opponents by doing so. One can imagine it as a task of filling in the gaps, where after everyone has submitted their proposal all parties can vote for an answer provided by other players. There are no rules by which this voting is conducted; one can simply choose an answer that seems to be most accurate or one that seems rather humorous.

Our project connects this idea with the readily available database of Wikipedia articles, which is one of the results of the efforts of Niklas Pohl[2], but changes the rules slightly. Where in Fibbage one should want to provide an answer to a more open-ended question, the main question in our game, from here on simply referred to as WikiGame, is "What article lies between article `A` and article `B`?". Now, to simply choose any article that could be lying between two articles `A` and `B` does not incite any sense of competition yet. To remedy this, the game rewards players that find an article `C` between articles `A` and `B` whilst minimizing the distance to them.

Visually speaking, if articles are considered to be nodes in a graph, then, naturally, the distance between two specific ones are the number of edges connecting them. The connection between articles is established by hyperlinks, e.g., if we were to look up the Wikipedia article for ETH Zurich, we should be able to find a hyperlink to the Wikipedia article for Zurich.

In WikiGame, similar to Fibbage, we also include a phase in which players can choose the articles selected by their opponents and vote for them (and no, a player cannot vote for themselves). The game then rewards all players based on their personal selection of their article, given articles `A` and `B`, and based on how many votes they were able to receive from their opponents.

WikiGame is played with a main screen every party has access to, which serves as the host of the game and where all important game-related information, such as scores or current articles for example, are displayed. Players join with their personal devices and choose their articles there.

Chapter 2

Development

In the following we will touch upon languages and technologies used to realize WikiGame. Many more approaches were taken during development, but this section focuses only on the technologies that made it in the final product. For a better overview, we distinguish between what is used on the back-end (server-side) and on the front-end (client-side)2.1.

2.1 Back-End

Python Our reasons for choosing *Python*[3] as the language for implementing the back-end is twofold. It allows for fast prototyping and also increases maintainability in case future students want to expand on our work, perhaps by adding a new game mode that makes use of the database. We implemented the logic for creating and joining lobbies, for tracking scores and examining the validity of the user's input in *Python*.

FastAPI *FastAPI*[4] is a library that is compatible with *Python* that helps developers create API endpoints for their applications. On one hand, it allows for clients to communicate with the back-end via these endpoints, but also enables the usage of websockets, which we required information distribution in real-time from and to the back-end.

Uvicorn In order to serve our backend as a web application we also use *Uvicorn*[5]. It serves as the binding elements that handles all connections from the clients and then passes the requests to *FastAPI* to handle them accordingly.

Neo4J *Neo4J*[6] is a graph database management system, allowing us to make graph-related queries to it. It is the backbone for managing the relationship between the Wikipedia articles and for efficiently retrieving relevant metrics, such as the aforementioned distance between two articles. For fast access, the entire *NEO4J* database is loaded into memory upon starting the back-end.

Redis *Redis*[7] is an in-memory database which was needed as an intermediate means for retrieving data. This was used to store very minimal information

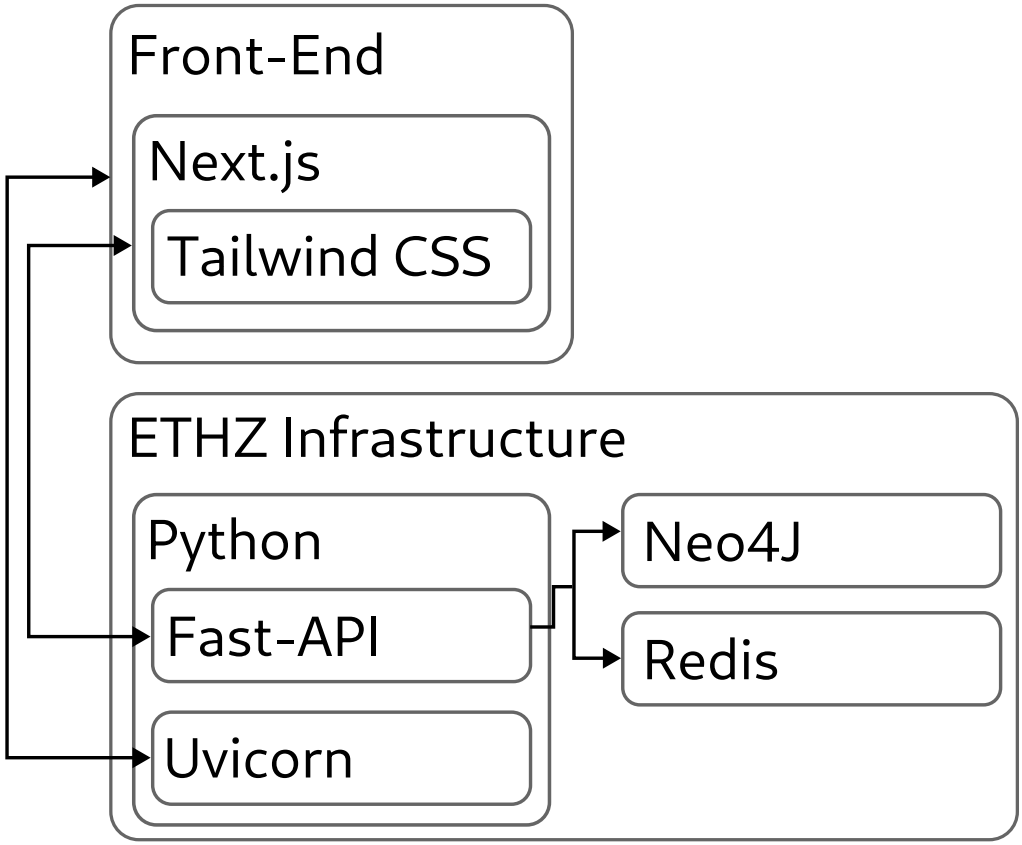

Figure 2.1: Small overview of all notable technologies used for the project, alongside their relation.

on the articles, mainly an identifier of how *Neo4J* knows an article by and the article name itself. Therefore, whenever we need to query for just any article, we make use uf *Redis* for faster access, and the poll the rest of the data from *Neo4J*.

2.2 Front-End

Next.js *Next.js*[6] is a React framework that we use to generate static websites for our web application that serve as the front-end of WikiGame. The choice for this specific framework arose from the fact that it is pretty straight forward to set up simple pages for early testing, but is versatile enough for us to create visually appealing pages for the game.

Tailwind CSS *Tailwind CSS*[8] is a CSS framework that offers a highly customizable and efficient way to stylize web applications. Unlike traditional CSS, *Tailwind CSS* provides utility classes that directly apply styling to HTML elements, which allows us to rapidly develop and easily maintain the visual components required for the game.

Chapter 3

Method

In this section we want to give insight on what is happening during the various phases of not only the game, but the entire application. We denote the host screen as the screen, all involved parties have access to at all times during the game, and the client screen the screen on a personal device, where only the corresponding player has access to.

3.1 Lobby Management

When first starting the game, one can either choose to create a lobby and thus assume the role of a host, or join an existing lobby. For a player to join, a lobby has to exist beforehand.

Creating a Lobby Before creating a lobby, the host needs to decide on how many rounds should be played, and whether the articles, between which the players need to find the shortest path to, are selected randomly or selected by the players.

When creating a lobby, a `lobbyID` and a QR-code3.1 are generated and displayed for all players to see on the host screen. This identifier is required for joining the newly created lobby. The host then waits for players to connect and choose a username and an avatar. At least three people need to be present in a lobby for a game to start, but the number of players cannot be over eight.

When all players are ready, i.e. all players have decided on a username and avatar, the host is allowed to start the game. When starting, a countdown is displayed the host is allowed to interrupt. If the countdown reaches zero, the game is started.

The host screen is not involved as a player, but as a means for displaying information to all players at all times.

Joining a Lobby If someone wants to participate in the game, they join an existing lobby by putting in the `lobbyID`. If the `lobbyID` is wrong, or the lobby does not exist, the player is notified. Alternatively they can scan the displayed

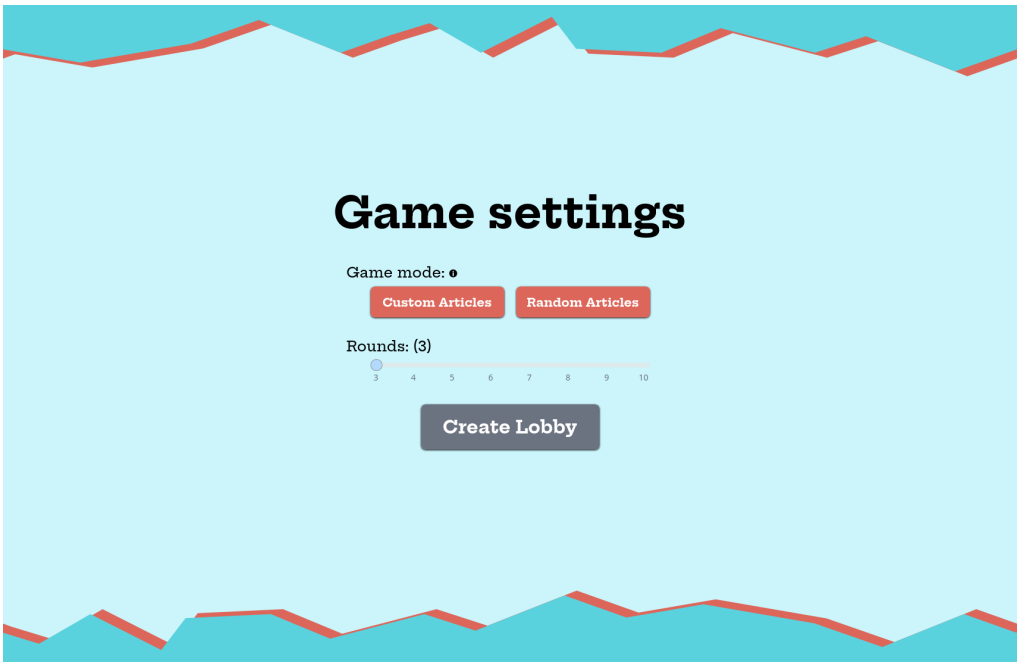

Figure 3.1: When creating a lobby, the host is free to decide on the settings such as the number of rounds and the method of retrieving the left and the right article before players are able to join.

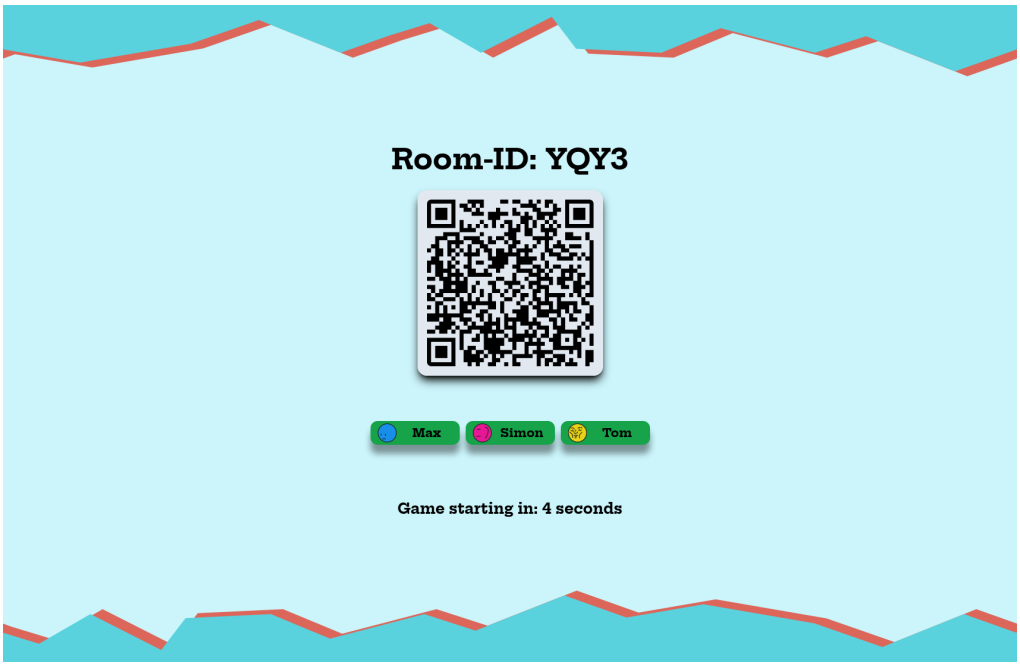

Figure 3.2: Players that joined the lobby are being displayed in real time on the host screen.

QR code to automatically fill in the field for the `lobbyID`.

Before joining the lobby, the player must decide on a username and select one of the nine preset avatars. If either one of these conditions is not fulfilled, the player cannot join the lobby due to the button necessary for it being disabled.

Depending on whether the articles are chosen randomly or not, the player needs to select a number of articles before readying themselves. Players that joined the lobby are being displayed on the host screen3.2.

3.2 Game Management

A round of WikiGame can be split up into four distinct phases. These are

1. Selecting an Article
2. Voting for an Article
3. Presenting Paths
4. Showing Scores

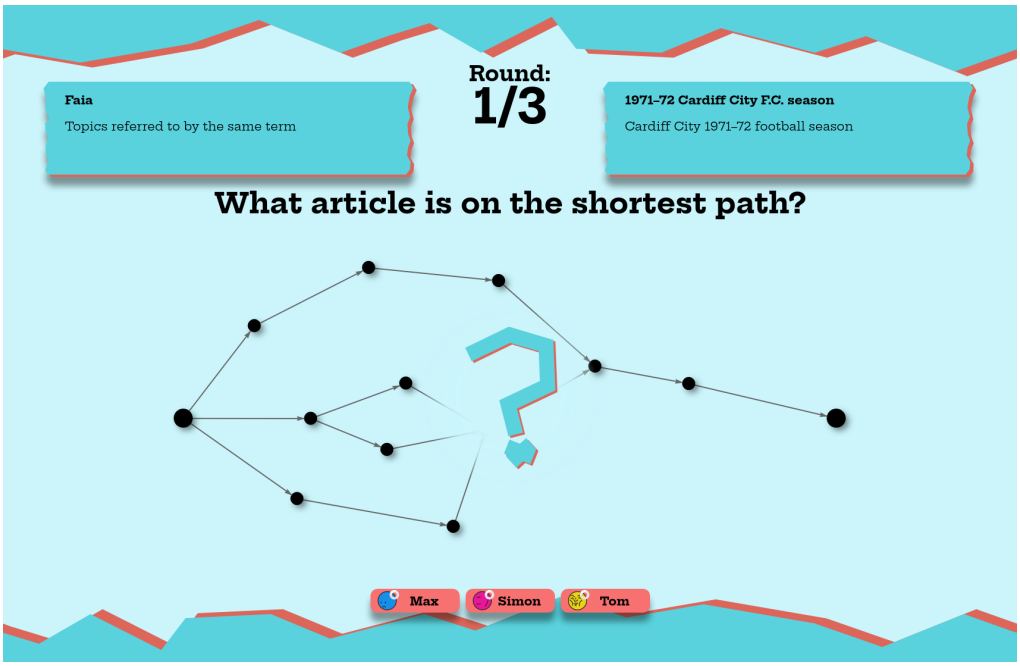

Figure 3.3: When a round starts, players are presented with two articles, between which they need to select the article that minimizes the distance between them. Furthermore, the host screen actively shows who already submitted an article.

After looping through these phases for a set amount of rounds, the game goes into a final phase where the game results are shown, from which the host can decide to start a new game.

6. Show Game Results

In the following, we want to explain what is happening during these phases.

Selecting an Article When the round starts, players are presented with two articles on the host screen alongside the question, what article minimizes the distance between these two. The player then makes a choice on their personal device, where the client makes sure that only existing Wikipedia articles can be submitted. Upon submission, the host screen displays that the corresponding player has handed in an answer3.3. When all players have put in their selection, the game goes to the next phase.

Voting for an Article The host screen presents all articles that were submitted by the players but withholds the information, by whom the article was chosen3.4. On their personal devices, players now must vote for one of these articles, excluding their own. The player has full authority over which criteria they choose an article; they could give a vote to the article they find funniest or to the article they think truly minimized the path between the given articles.

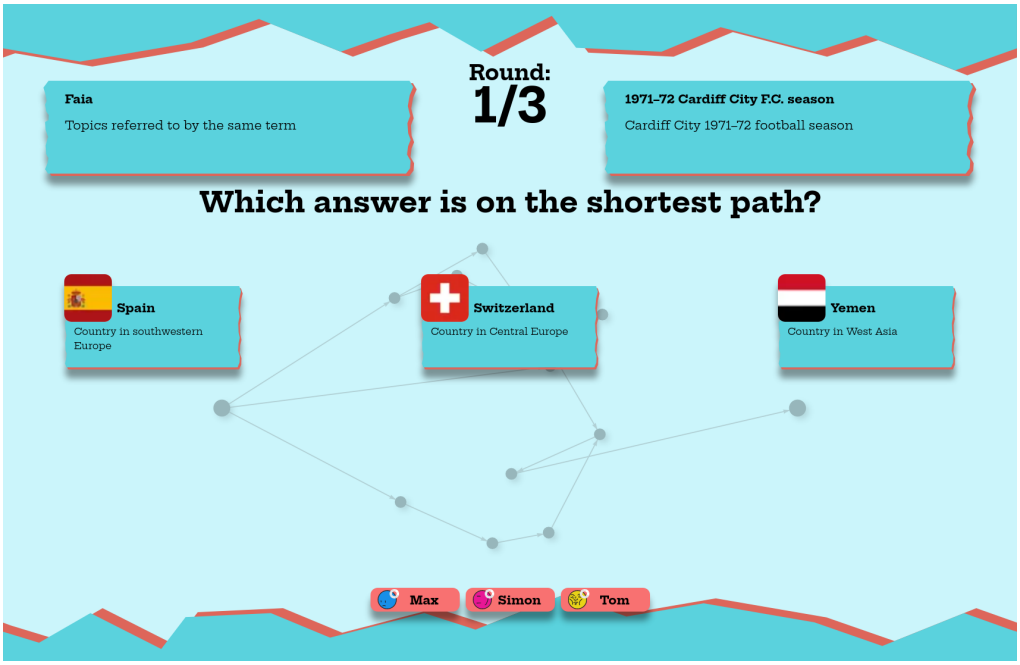

Figure 3.4: Player-selected articles are being displayed anonymously for all players to see. On the client-devices, players can choose for which article to vote, whereby their own article is not being displayed as an option.

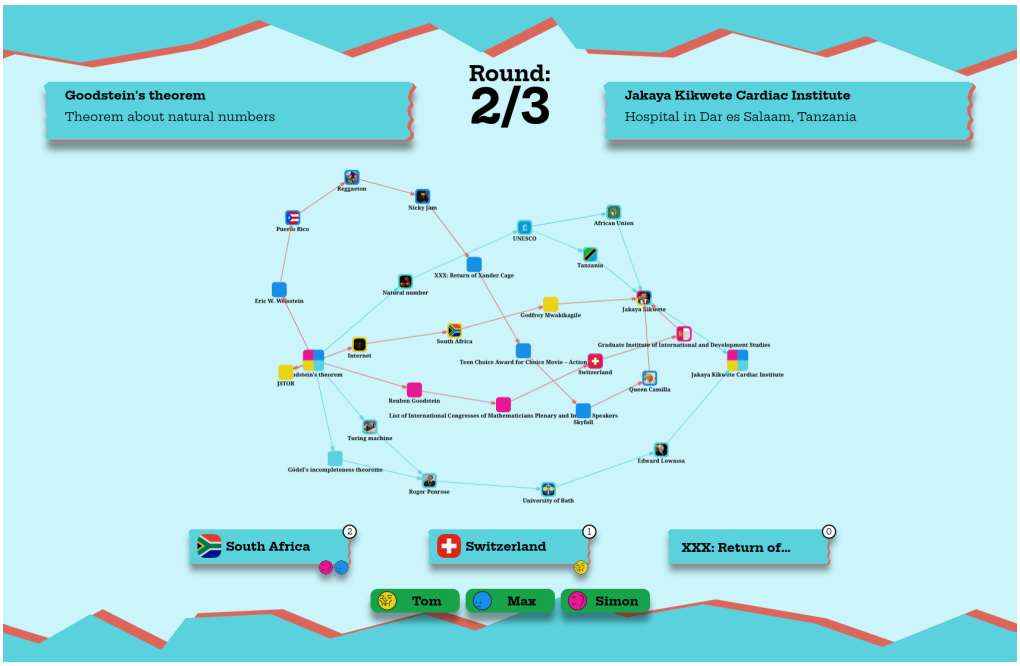

Figure 3.5: All player chosen articles are being added to the graph one by one, showing the paths and relation to the given articles. This is fully animated in order to build anticipation.

Presenting Paths When all votes have been submitted, the host screen switches to display a graph. Initially, it shows the given two articles alongside some possible articles that minimize the path between them. Then, one by one, the articles submitted by players are integrated in a manner that shows, via which hyperlinks one would go from the first given article to the player-submitted article, and from there to the second article. While doing this, it briefly highlights all nodes in-between. The idea behind this is to give player a sense of anticipation when their solution is being evaluated3.5.

Showing Scores When all paths have been shown, the host shows the rewarded scores for each player during the last round. Each player can see how many points accounts for their choice of article and how many points are assigned due to the votes received from other players. These are visually added to the sum of points for each players from the previous round3.6.

Showing Game Results When all rounds have been played, the host screen switches to a view showcasing the performance of each player for the entire game3.7. This screen then nominates the winner and shows the rank of the remaining players3.8. From here, the host can either decide to start a new game, which then switches back to the lobby, allowing players to leave or other newcomers to join, decide to end the game session.

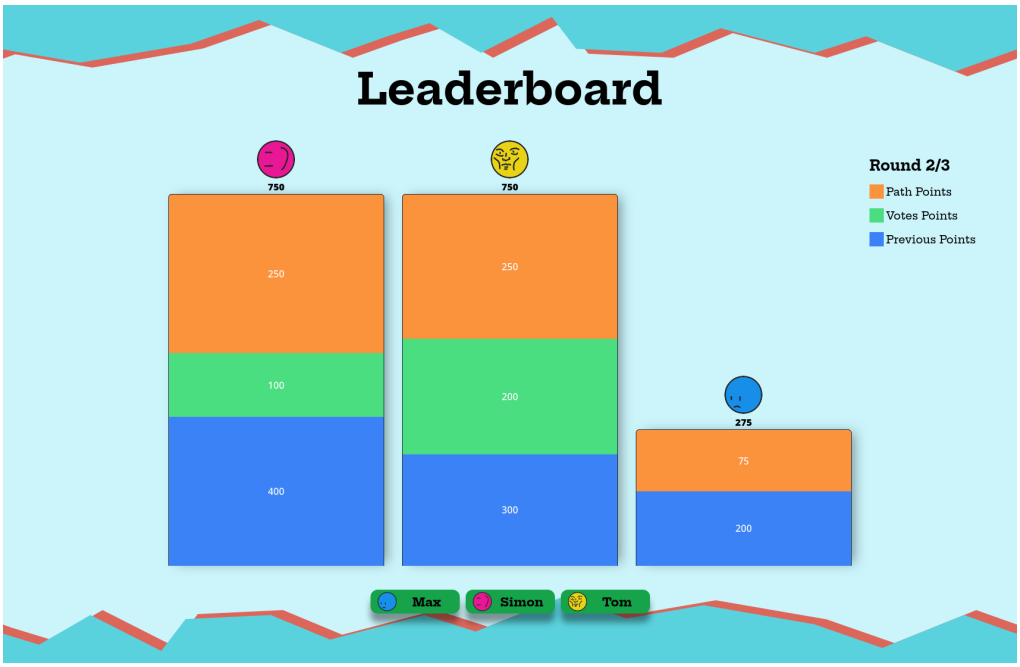

Figure 3.6: Points rewarded during past rounds are being accumulated visually between each round on the host screen, allowing players to see how well they are performing.

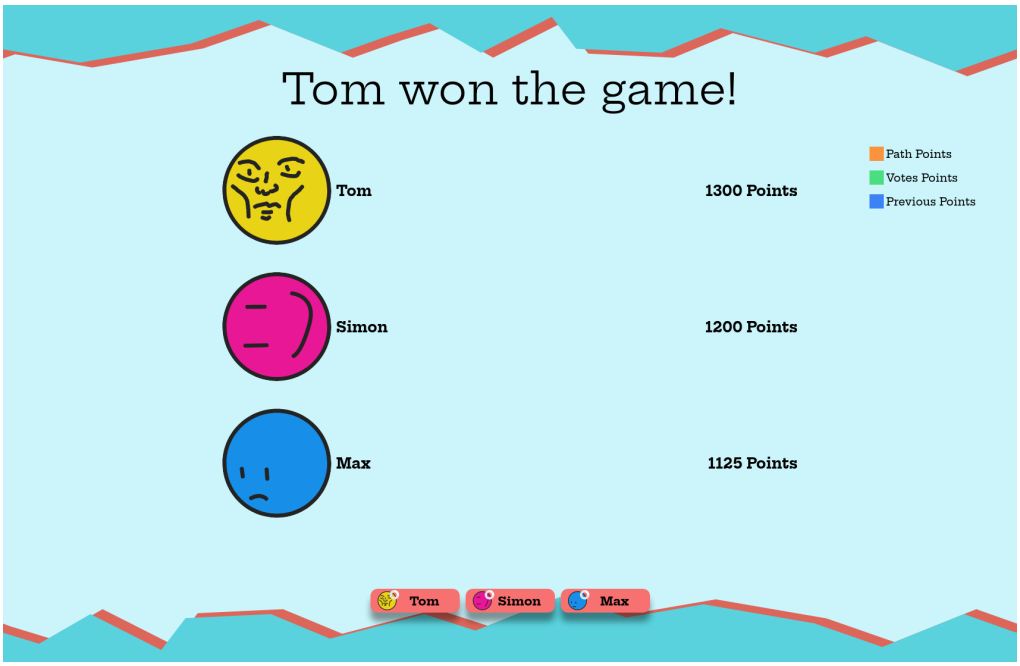

Figure 3.7: After all rounds have been played, the host screen shows a leaderboard.

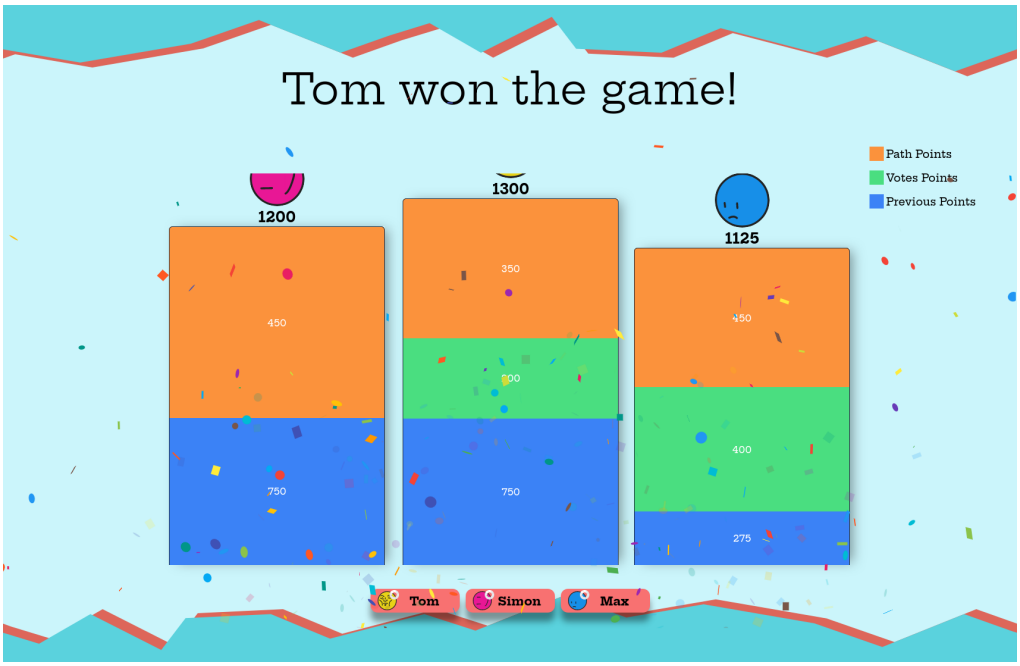

Figure 3.8: Similar to in-between rounds, scores are added, but after the last round has been played, the host screen presents the winner of the game.

3.3 Scoring Mechanism

The scoring system in our project is designed to incentivize both participation and optimization of article paths. Points are allocated based on two primary components: votes-based points and path length-based points.

Votes Based Points Participants must vote for one article, and each vote is assigned a value of 100 points. The number of points a participant receives is directly proportional to the number of votes their article has garnered. Mathematically, this is expressed as:

$$votes\_points = n\_votes \times 100$$

where n_votes denotes the number of votes received by the participant's article.

Path Length Based Points The path length based points are determined by the proximity of an article to the shortest path from the start to the end articles. This calculation involves several steps:

- Determine the *minlen*, representing the length of the shortest path from the start to the end articles.

- Calculate the length of the path from the start to the end, which includes the participant's article.
- Compute the difference from the *minlen*, and add 1 to avoid potential division by zero errors. This difference serves as the weight for each article.
- Points are distributed proportionally to the inverse weights of each article, with a fixed total amount of path length based points. For instance, if the total path length based points are 600 and a participant's inverse weight amounts to 40%, they would receive $600 \times 40\% = 240$ points.
- To maintain neatness, points are rounded up to the nearest 25.

The total amount of path lengths-based points is determined as $total\_path\_points = n\_players \times 200$, ensuring there are double the number of path length based points compared to votes-based points.

This scoring mechanism ensures a balanced evaluation of participants' contributions, incorporating both the quantity of support received and the quality of their submissions in relation to the shortest path.

Bibliography

[1] "Jackbox games - fibbage," https://www.jackboxgames.com/games/fibbage.

[2] N. Pohl, "Wikipedia Walker," https://pub.tik.ee.ethz.ch/students/2023-FS/BA-2023-07.pdf.

[3] "Python," https://www.python.org/.

[4] "FastAPI," https://fastapi.tiangolo.com/.

[5] "Uvicorn," https://www.uvicorn.org//.

[6] "Next.js," https://nextjs.org/.

[7] "Redis," https://redis.io/.

[8] "Tailwind CSS," https://tailwindcss.com/.