ICFP Programming Contest 2023

v1

Aymeric Fromherz

Introduction

After a successful stint as painters, the denizens of Lambda land are now turning to different artistic endeavors, and starting a new career in music. Passionate about their craft, they wish to make sure that their diverse audience enjoys their vibe to the greatest extent. However, while the Lambda band consists of talented musicians, organization is not their strongest suit. Your mission, if you accept it, will be to place each musician on stage to provide the best possible sound for attendees.

Timeline

Note that there will be updates to this specification, and more problems will be released during the contest. This will happen at these specific times:

- 24 hours into the contest (after the lightning division ends)
- 48 hours into the contest

Changelog

- Added clarification about validity of musician placements
- Added clarification about blocking sound radius

Problem Specification

The task is to place each musician on stage, to maximize the happiness of all attendees in the room.

As part of individual problems, you will be given a JSON file containing:

- The topology of the room, including its size, and the dimensions and location of the stage
- A list of musicians in the band, with the instrument they are playing
- A list of attendees, including their location in the room, and their taste in each instrument

As part of the task, you will also be given a scoring function to compute the total enjoyment of each attendee depending on the placement of the band.

Room Topology. The Lambda band will be playing in rectangular rooms; their size will be specified using two parameters `room_height` and `room_width`. The stage will also be rectangular; its size will be specified using two parameters `stage_height` and `stage_width`. The stage can be situated anywhere in the room, its position will be specified using the array `stage_bottom_left`, containing the `x` and `y` coordinates of its bottom-left corner.

Musician Placement. Musicians in the band must be placed in the area delimited by the stage. To ensure they have enough room for playing, they must not have any other musician or an edge of the stage in a circle of radius 10 centered on them. A placement where a musician is at distance exactly 10 from an edge or another musician will be considered as valid. Musicians will be specified through the array `musicians`. This array will contain the instruments that each musician is playing, represented as a numeric identifier. Solutions where musicians are incorrectly placed, or where all musicians have not been placed will not be counted.

Attendees. The specification of attendees is provided in the array attendees. Each attendee is represented as a structure containing the following elements:

- Its position, specified using the attributes `x` and `y`. Attendees will never be located on stage.
- Its taste in each instrument, represented as an array `tastes`. `tastes[i]` corresponds to the taste of the attendee in the instrument with identifier `i`.

Problem Solution. Problem solutions must be given as a JSON object with the attribute `placements`, consisting of structures with attributes `x` and `y`, corresponding to the coordinates of each musician. The i-th element of `placements` corresponds to the position in the room of the i-th musician specified in the attribute `musicians`.

Example Problem As an example, here is an input of a sample problem, containing three musicians and three attendees. The band consists of two different instruments, with identifiers `0` and `1`.

```
{
  "room_width": 2000.0,
  "room_height": 5000.0,
  "stage_width": 1000.0,
  "stage_height": 200.0,
  "stage_bottom_left": [500.0, 0.0],
  "musicians": [0, 1, 0],
  "attendees": [
    { "x": 100.0, "y": 500.0, "tastes": [1000.0, -1000.0
      ] },
    { "x": 200.0, "y": 1000.0, "tastes": [200.0, 200.0]
      },
    { "x": 1100.0, "y": 800.0, "tastes": [800.0, 1500.0]
      }
  ]
}
```

An example solution would be the following.

```
{
  "placements": [
    {"x": 590.0, "y": 10.0 },
    {"x": 1100.0, "y": 100.0 },
    {"x": 1100.0, "y": 150.0 }
  ]
}
```

Scoring Function For a given problem, the scoring function corresponds to the cumulated happiness of all attendees. The happiness of an attendee is the sum of the impact of each musician. For an attendee `i`, we compute the impact of musician `k` as follows:

We denote as d the distance between the attendee and musician, computed as $d = \sqrt{(attendees[i].x - placements[k].x)^2 + (attendees[i].y - placements[k].y)^2}$. In the general case, the impact of musician k on attendee i is defined as

$$I_i(k) = \left\lceil \frac{1,000,000 * attendees[i].tastes[musicians[k]]}{d^2} \right\rceil$$

However, another musician k' can block the sound coming from k. We consider that k' blocks the sound from k for attendee i if the line from k to i intersects the circle of radius 5 centered on k'. This boundary is inclusive: we consider the sound as blocked even if the intersection consists of one point. If the sound is blocked, $I_i(k) = 0$.

Deadlines

As traditional, the contest will have a Lightning Division spanning the first 24 hours. To qualify for the Lightning Division prize, submit your solution by July 8 2023, 12:00pm (noon) UTC.

To qualify for the Full Divison prize, submit your solutions by July 10 2023, 12:00pm (noon) UTC.

In order to qualify for any prizes, your source code must be submitted by the end of the contest as well. You can do this through the web portal.

Determing the Winner

We will use the same procedure to determine the winner in both the lightning and full divisions, ranking the teams by cumulative score, computed as the sum of scores for each task.

Submission

In addition to the web interface, you can use the following endpoints for your submissions.

/submission : Get Get submission with ID. You can only get submissions linked to your account.

Usage: `api.icfpcontest.com/submission?submission_id=[submission-id:string]`

Requires: Authorization header set to `Bearer <token>`. You can find the token string on your UI dashboard or receive a token via login endpoint.

Returns:

```
{
    "Success": {
        "submission": {
            "_id": string, // submission id
            "problem_id": number,
            "user_id": string,
            "score": {
                "Failure": string | "Success": number //
                    submission result from judge
            },
            "submitted_at": string // submission time as
                UTC
        },
        "contents": string // submission contents
    } |
    "Failure": string // failure message
}
```

/submission : Post Post submission with contents and problem id.

Usage: `api.icfpcontest.com/submission`

Requires: Authorization header set to `Bearer <token>`. You can find the token string on your UI dashboard or receive a token via login endpoint.

Body

```
{
    "problem_id": u32,
    "contents": string // submission contents
}
```

Returns `submission_id as plain tex`

/submissions: Get Get [limit] number of your past submissions starting from a given [offset], sorted by submission time. offset=0 indicates you start from your most recent submission and receive [limit] number of submissions back. Setting [problem_id] allows you to only receive the submissions for that specific problem.

Usage:

```
api.icfpcontest.com/submissions?offset=[offset: u64]&
   limit=[limit: i64]
api.icfpcontest.com/submissions?offset=[offset: u64]&
   limit=[limit: i64]&problem_id=[problem_id: u32]
```

Requires: Authorization header set to Bearer <token>. You can find the token string on your UI dashboard or receive a token via login endpoint.

Returns:

```
{ "Success" :
    {
        "_id": string;
        "problem_id": number;
        "submitted_at": string;
        "score": "Processing" | { "Failure": string } |
           { "Success": number };
    }[]
}
| { "Failure": string }
```

/problem: Get Get problems contents with problem id.

Usage: api.icfpcontest.com/problem?problem_id=[problem_id:u32]

Returns:

```
{
    "Success": string // problem definition
}
| {
    "Failure": string // error message
}
```

/problems : Get Get number of problems.

Usage: api.icfpcontest.com/problems

Returns:

```
{
    "number_of_problems": number,
}
```

/scoreboard : Get Get the global scoreboard. The scoreboard is updated at most once a minute. Frozen means scoreboard is frozen at the last updated point, so you are seeing the snapshot from that moment.

Usage: `api.icfpcontest.com/scoreboard`

Returns:

```
{
    "frozen": bool,
    "scoreboard": { // sorted by scores
        "username": string,
        "score": number,
    }[],
    "updated_at": string // date
}
```

/userboard: Get Get scoreboard for your account. Provides the highest score for each problem.

Usage: `api.icfpcontest.com/userboard`

Requires: Authorization header set to `Bearer <token>`. You can find the token string on your UI dashboard or receive a token via login endpoint.

Returns:

```
{
    "Success": {
        "problems": (number | null)[] // if no
           submission to a question, then null. If all
           failing, then 0. Otherwise, highest score.
    }
}
| {
    "Failure": string // failure message
}
```

/register : Post Register your account to the contest.

Usage: `api.icfpcontest.com/register`

Body:

```
{
    "username": string,
    "email": string, // must be a valid email
    "password": string
}
```

Returns:

```
{
    "Success": string // your JWT access token
}
| {
    "Failure": string // failure message
}
```

/login : Post Login with your account

Usage: `api.icfpcontest.com/login`

Body:

```
{
    "username_or_email": string,
    "password": string
}
```

Returns:

```
{
    "Success": string // your JWT access token
}
| {
    "Failure": string // failure message
}
```