[RMG-1] Create digital version of Resource management game

Description

Resource Management Game

Data Model

The data that models the game consists of three major varieties: Simple, non-aggregate values; immutable objects/aggregates; and mutable objects.

Values

Enum `Skill`: heart, spade, diamond.

Enum `Role`: The player's role in the game.

Enum `Round`: Not used as of now.

Enum `RequestState`: Not used as of now.

The time that is required to model cards, project demands and the two types of board is represented as integers.

Immutable parts

The following entities are immutable and can not be modified after they have been created. They may be considered static assets: we could provide them in some data file, json, XML or something else. They have `GET` endpoints but no `PUT`/`PATCH`.

A `ResourceCard`. The set of resource cards is created when the game starts. The deck is fixed and given as in the paper version of the game.

- `GET /games/<gameId>/resources`
  - Fields: id, name, skill (heart, diamond, spade), time, homeBoardId

A `Demand`. Only used for storing the set of requirements for a project.

- `GET /games/<gameId>/projects/<projectId>/demands`
  - Fields: skill (heart, diamond, spade), projectTime

A `Project`. **Projects** are created as given in the paper version.

- `GET /games/<gameId>/projects`
  - Returned fields: id, name, start, finish, demands

Mutable parts

`ProjectPlan`: What can change? The placement of the Project (int projectStartTime), and the set of resource cards placed on the project plan, and which project manager, aka. owner the project plan belongs to (ex. the player id “pl-05”)

- Read all project plans:
  - `GET /games/<gameId>/projectPlans`
    Return the list/set of project plans. For representation of a single project plan, see exmple below.
- Read project plans for a specific player:
  - `GET /games/<gameId>/projectPlans?ownerId=<playerlId>`
- Read a single project plan:
  - `GET /games/<gameId>/projectPlans/<projectPlanId>`
    Example reutrn value: {{ {"id": "pp5", "ownerId": "pl02", "projectId": "prj5", "projectStartTime": 4}

    }}

- Read project plan allocation (aka. resource cards placed on the project plan board):
  - `GET /games/<gameId>/projectPlans/<projectPlanId>/allocations`
    Return the set/list of resource card IDs
    Example return value: `["res03", "res23", "res01", "res77", "res42"]`
- Set `ProjectPlan` owner:
  - `PATCH /games/<game-id>/projectPlans/<projectPlanId>`
    Example payload {{ { "ownerId": "pl-03" }

    }}

`ResourceBoard`: What can change? The set of cards that remain on the board.

- Read all resource boards:
  - `GET /games/<gameId>/resourceBoards`
    Returns the set/list of boards, each of which consists of the fields: `boardId`, `title`, `ownerId`.
- Read all available resource cards on a resource board:
  - `GET /games/<gameId>/resourceBoard/<boardId>/resources`
    Return list of resource card IDs, e.g. `["res01", "res02", "res03", "res17"]`
- Read resource board for a given resource manager / owner:
  - `GET /games/<gameId>/resourceBoards?ownerId=<playerlId>`
    Returned fields: see above.

`Request`: What can change? The state (open, approved, rejected).

- Read all requests:
  - `GET /game/<game-id>/requests`
    Returned fields: id, skill, time, requestingProjectPlanId
    Example: {{ { "id": "abc89", "skill": "diamond", "time": 4, "requestingProjectPlanId": "PP2" }

    }}

`Player`: What can change? The role.

- Read all players:
  - `GET /games/<game-id>/players`
    Return the list of all players in this game. For representation of a single player, see below.
- Read player data
  - `GET /games/<game-id>/players/<player-id>`
    Example return value: `["id": "p007", "name": "Paolo Projetto", "role": "RM"]`
  - `GET /games/<game-id>/players/<player-id>/score`
    Return the points that this player scored for all rounds played.
    Example return value: `[25, 20, 40]` for 25 points in the first round, 20 point in the second, 40 for the third round.
- Change player role:
  - `PATCH /games/<game-id>/players/<player-id>`
    Example payload {{ { "role": "PM" }

    }}

Changes

For all things that can change, we need to implement backend `PATCH` endpoints. Below are only suggestions, the URLs/payload could be very different from the made-up examples.

Create Game:

`POST /game`

- In the backend, a unique GUID (`game-id`) is generated for the game
- Response with game invitation URL: /games/<game-id>

Create Player:

`POST /games/<game-id>/player`

Example payload: {{

{"name": "xyz", "role": "DEALER"}

}} for dealer

Returns the player Id {{

{"id": "pl-01" }

}}

- In the backend, a unique GUID (`player-id`) is generated for the player

Move a project

`PATCH /games/<game-id>/projectPlans/<projectPlanId>/moveProject`
Example payload: {{

{ "projectId": "PR3", "startTime": 3 }

}}

Return an error when game rules are violated, e.g. a project if shifted to an invalid time slot (like time 10).

What is required in the background? Return only those cards that cannot be used after shifting the project, i.e. find the cards that do not intersect with the project's demands and return them back to the home board.

Requesting a Resource

- Create a new request:
  - `POST /game/<game-id>/request`
    Example payload: {{ { "skill": "diamond", "time": 4, "targetProjectBoard": "PB3" }

    }}
- Approve a request:
  - `POST /game/<game-id>/request/<requestId>/approve`
    An approval will move the first-matched card automatically.
- Revoke a request
  - `DELETE /game/<game-id>/request/<requestId>`

Return a Resource Card

Returns a card back to the home `ResourceBoard`.

`POST /game/<game-id>/resources/<resourceId>/return`

In the backend, this removes the card from the `ProjectPlan` that it is currently placed on and adds it back to its home `ResourceBoard`.

Reset the Game

Optional: Reset the round: Return all ResourceCards back to their home resource board.

{{POST /games/<game-id>/reset }}

Code

Java Code that models the basic data types can be found in this repo: https://github.com/itd-ext/resource-management-game