7 Wonders

Requirement and analysis document (RAD)

Emmie Berger
Gabriel Hagström
Samuel Hammersberg
Sebastian Selander
Björn Rosengren

Date 2021-10-10
Version 1.0

1 Introduction

This is a desktop and multiplayer variant of the board game 7 Wonders. The purpose of this application is to be able to play and enjoy the board game on desktop. Our target demographic are people who enjoy board games but have suffered because of the terrible disease that is Covid-19. The benefit for these poor souls is that they now can enjoy a wonderful game with friends.

1.1 Definitions, acronyms, and abbreviations

- Client - The part of the app that the user interacts with. Can be seen as a view/controller for the underlying model.
- Server - The link between the model and the clients. The model tells the server to serve the clients with the data.
- Client-server model - Sits between the view-controller and the model. Acts like an adapter pattern to convert java-representation to be network compatible.
- Model - An entity that holds all of the logic and information about the game.
- View - The graphical interface that represents the game.
- Controller - An entity that lets the player give input to the game.
- JSON - A hashmap-like data structure with key and values.

2 Requirements

2.1 User Stories

- As a user I want to be able to draw cards when it's my turn so that I can get resources and perform actions
  - A player can draw only one card each turn
  - A player can't skip drawing a card
  - If the card requires resources to perform an action, the player must have the resources to perform the action.
  - If the card requires resources to perform an action and the player has insufficient resources, the card must go into the trash can
  - If the card gives the player resources, they will be added to the player's pile of resources
- As a user I want to have a monument that I can level up to be able to win the game
  - A player must be able to see a graphical representation of their own monument
  - A player must be able to see what is required to level up their monument at all levels
  - All players must have exactly one monument
  - A player cannot change monument when the game has started
  - When the player wants to upgrade their monument, they must have sufficient resources for the specific level
- As a player I want to have a deck of cards to be able to get resources and beat my competitors
  - All cards in the deck must have a type
  - All cards in the deck must have an age
  - Cards can only be played during their specified age
  - If a card has a cost, it must be shown to the player who has the possibility to draw it.
  - If a card gives resources, it must be shown to the player who has the possibility to draw it.

- As a player I want to able to select a nickname so other players can see who I am
  - A nickname must be set before the game starts
  - A nickname must be at least three characters long
  - All player must be able to see each other's nicknames
  - A nickname cannot be changed during the game once it has started
- As a user I want to be able to play in the three ages so I can go forward in game
  - The game starts in age 1
  - The game changes to age 2 when all cards in age 1 has been drawn or thrown into the trash can
  - The game changes to age 3 when all cards in age 2 has been drawn or thrown into the trash can
  - At the end of age 3 the game should calculate all scores and decide a winner.
- As a user I want to be able to perform an action during each turn so that I can progress in the game
  - When a player has been given a pile of cards, they must choose to draw a card, build a stage of their monument or throw a card into the trashcan
  - If the player chooses to draw a card that requiers resources to be drawn or build a stage of their monument, they need to have efficient resources
  - The player must choose exactly one of the above actions
  - When the player has performed one of the actions, they will give the remaining pile of cards to the player on their left

2.2 Definition of Done

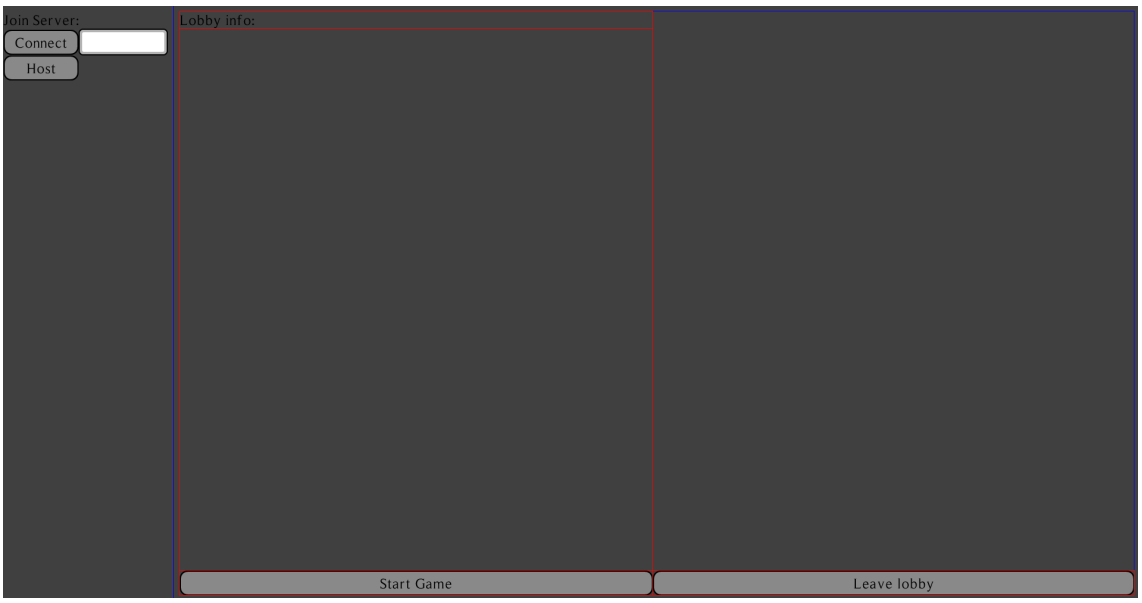
- Everything that can be tested must be tested ($\geq$ 90% line coverage)
- A task must be implemented both in back- and front-end to be considered done

2.3 User interface

We have created a prototype for our game.

- Starting page: When the user starts the game, they will be presented to a starting screen with different options.

- If the user chooses to start the game, they will be able to choose from different servers to join.

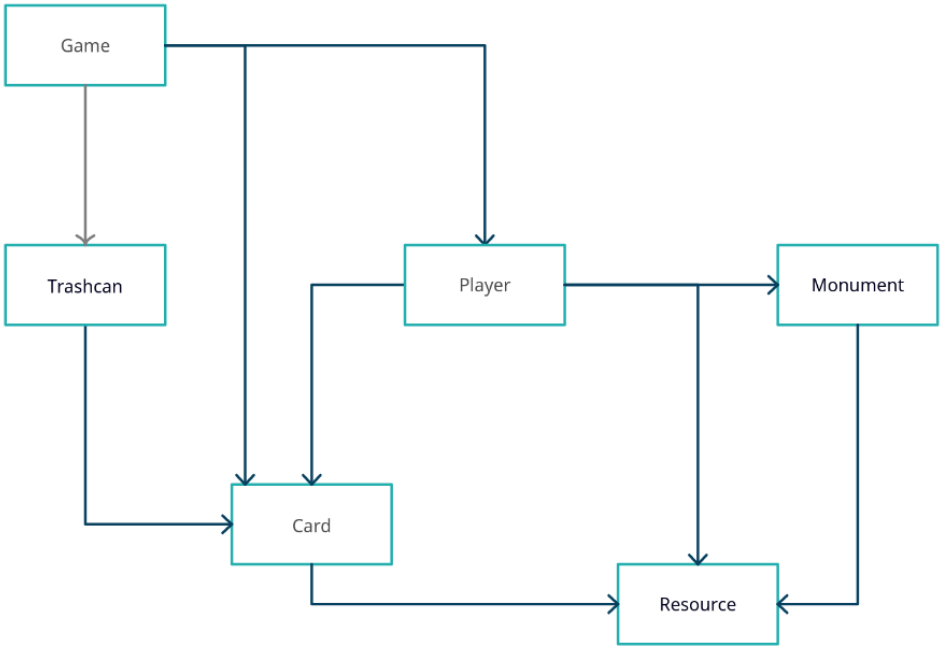


- After the user has chosen a lobby and enough players have joined, the game will start. Each player will be given a monument and a set of cards to choose from. In this example, the user ha been given *The great Pyramid of Giza* and has 5 cards to choose from. Their left neighbor has the monument *The temple of Artemis at Ephesus*. The view will remain the same during the rest of the game, whereas the user continues to draw cards and upgrade their monument.

- Here is what it looks like after the player has drawn a card in the example of the previous picture.


Välj ett kort
BANK
BANK
GIZAH
Wood: 2
Ore: 0
Clay: 0
Stone: 0
Cloth: 0
Water: 0
Gold: 11
War: 1

3 Domain model



3.1 Class responsibilities

- Game: Is responsible for the overall logic during the game when it's played. Initializes all objects of Player, Monument, Trashcan and Card.
- Trashcan: Keeps cards that have been thrown away during the game and needs to be saved for later.
- Player: A module that represents a user and it's data. Player has a class Player-State that keeps track of all resources and cards a user has.
- Card: Represents a card in the game that a user can draw. A card can give and/or cost resources to draw.
- Monument: A module that represents all monuments in the game. Stores data about what is required to upgrade each monument and how every upgrade will reward the player.
- Resource: An Enum that represents each possible resource in the game (wood, ore, stone, clay etc). Also includes coins, victory points and war tokens.

4 References

- Travis. Automatically testing after git push.
- Maven. Be able to fetch packages/modules.
- Libgdx. For the client to be able to render graphics.
- Org.json. To package data into a json-object which the server and client uses for communication.