Ticket To Dave's Heart

Documentation

Ticket To Dave's Heart

Daniel Sause

Michael Vasile

Joshua Bugryn

Lucas Kohorst

Date: 29-Apr-2019

unaccepted Revision: 0.10

Table of Contents

Revision History

Name	Date	Version	Reason for Change
Lucas Kohorst	27-Mar-19	0.1	Initial release
Lucas Kohorst	27-Mar-19	0.2	Added the RMI Chat Server
Josh Bugryn	27-Mar-19	0.3	Created GameDocumentation.md and edited to include a draft image of the game board
Lucas Kohorst	27-Mar-19	0.4	Added support for multiple game servers concurrently, the user can either select active games or create a game
Lucas Kohorst	1-Apr-19	0.5	Limited each game to 4 players
Josh Bugryn	6-Apr-19	0.6	Functional GameBoard GUI added
Lucas Kohorst	7-Apr-19	0.7	Started to add start and end of turn sequences
Josh Bugryn	16-Apr-19	0.8	Calculated the longest path for the end of game
Michael Vasile	20-Apr-19	0.9	A game board in a single JFrame
Daniel Sause	29-Arp-19	0.10	Expanded upon executive overview

Executive overview

Ticket to Dave's Heart is an adaptation of the board game Ticket to Ride. The objective of the game is to build a railroad network on the USA map. The application supports multiple games, which can be created and joined in the lobby and multiplayer-play over a network. Additional features include an in-game chat.

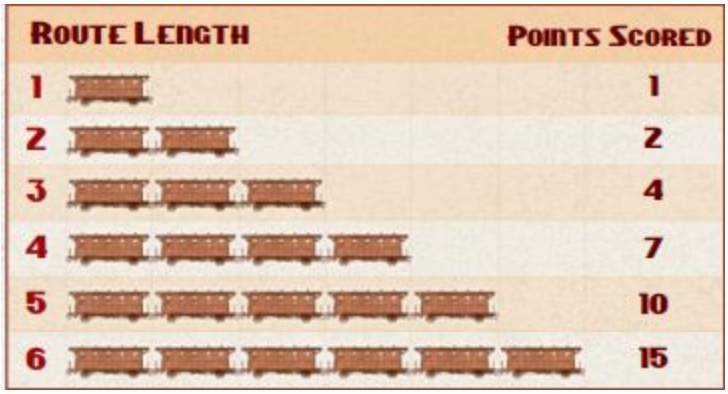
To play, after connecting to a server you choose between joining a pre-established game or creating a new game for yourself and friends to join. After joining, players are prompted for a unique nickname.

Once all the players are in the lobby, the game can start. It is recommended for players to wait until all of the members of the party are in the lobby before beginning, just as you would sitting around a table. This can be accomplished by communicating in the chat window adjacent to the game board.

Once the setup is complete, it is time to begin playing. You may find the detailed official rules to the game here: https://ncdn0.daysofwonder.com/tickettoride/en/img/tt_rules_2015_en.pdf.

To summarize the rules, the objective of the game is for players to gain the most points. In order to gain points, players can:

- complete destination cards. Players obtain between 2-3 of these at the beginning of the game and should attempt to connect the cities by the game's end.
- claim tracks. Every path claimed adds to the player's point total. The longer the track, the more the points are worth. However, if the track is gray, then it is only worth the number of trains that the route takes. When the track is colored, however, this is the point translation:



Route Length	Points Scored
1	1
2	2
3	4
4	7
5	10
6	15

- build the longest road. At the end of the game, the player with the longest road gets a 10 point bonus.

You may also lose points. The only way to lose points is by not completing any of the destination cards that you have in your hand. This includes any destination cards

that you may pick up mid-game. If you fail to complete these cards, the number on the card will be subtracted from your point total rather than added.

The game ends when any player has only 3 or fewer trains left in their inventory. Once this happens, the players will have one turn left each before the point totals are added up and the winner is announced.

Audience

This document is directed to any person/s that may be interested in the development of our java iteration of the game Ticket To Ride. This project will be far from perfect and may need further optimization. This document may help any fans of the game to take our work as a good starting spot to improve upon. Additionally, this program is relatively easy to re-use for different maps such as Ticket To Ride Europe.

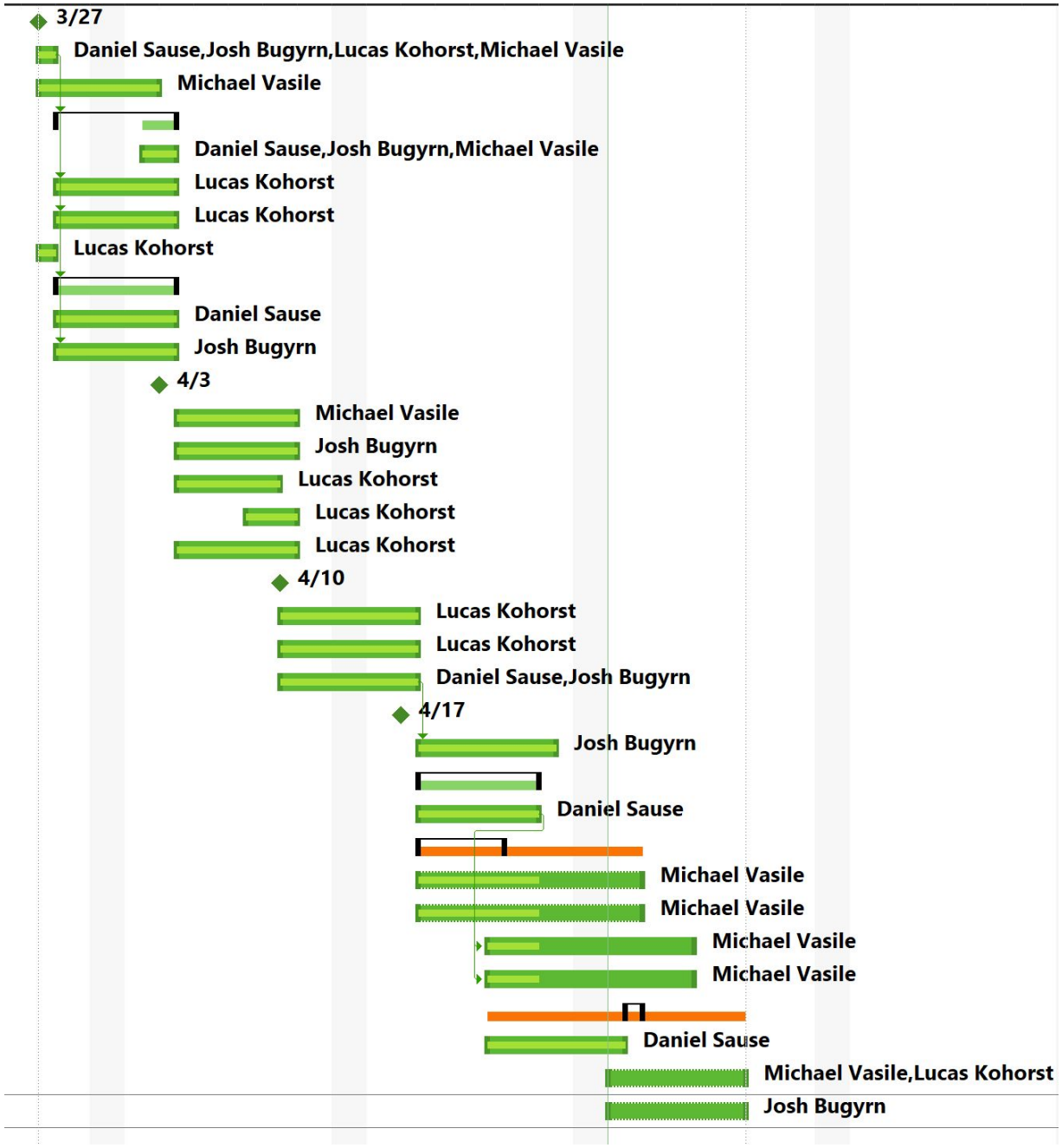
Application intentions

This application is for Ticket To Ride enthusiasts who may want to play the game with others over the internet rather than on the table.

Assumptions made for this project

The project requires client-server programming, so it is assumed that a network connection is required. Java Runtime 8 is also assumed to be required.

Gantt chart


3/27
Daniel Sause,Josh Bugyrn,Lucas Kohorst,Michael Vasile
Michael Vasile
Daniel Sause,Josh Bugyrn,Michael Vasile
Lucas Kohorst
Lucas Kohorst
Lucas Kohorst
Daniel Sause
Josh Bugyrn
4/3
Michael Vasile
Josh Bugyrn
Lucas Kohorst
Lucas Kohorst
Lucas Kohorst
4/10
Lucas Kohorst
Lucas Kohorst
Daniel Sause,Josh Bugyrn
4/17
Josh Bugyrn
Daniel Sause
Michael Vasile
Michael Vasile
Michael Vasile
Michael Vasile
Daniel Sause
Michael Vasile,Lucas Kohorst
Josh Bugyrn

Gantt task descriptions:

Task Name	Duration	Start	Finish	Predecessors	Resource Names
Meeting 1	0 days	Wed 3/27/19	Wed 3/27/19		Michael Vasile
Base Project Design	1 day	Wed 3/27/19	Wed 3/27/19		Daniel Sause, Josh Bugyrn, Lucas Kohorst, Michael Vasile
Documentation Update 1	5 days	Wed 3/27/19	Tue 4/2/19		Michael Vasile
Develop Naming Standards for Methods and Variables	**5 days**	**Thu 3/28/19**	**Wed 4/3/19**	**2**	**Daniel Sause, Josh Bugyrn**
Designate Return Types	2 days	Tue 4/2/19	Wed 4/3/19		Daniel Sause, Josh Bugyrn, Michael Vasile
Methods for Stub and Server	5 days	Thu 3/28/19	Wed 4/3/19	2	Lucas Kohorst
Limit Client Connections on Server	5 days	Thu 3/28/19	Wed 4/3/19	2	Lucas Kohorst
Support for muliple games	1 day	Wed 3/27/19	Wed 3/27/19		Lucas Kohorst

Proof of Concept for Route Logic	**5 days?**	**Thu 3/28/19**	**Wed 4/3/19**	**2**	**Daniel Sause**
Game Logic: Find Longest Route	5 days	Thu 3/28/19	Wed 4/3/19		Daniel Sause
Researching UI Elements	5 days	Thu 3/28/19	Wed 4/3/19	2	Josh Bugyrn
Meeting 2	0 days	Wed 4/3/19	Wed 4/3/19		Michael Vasile
Server GUI	5 days	Thu 4/4/19	Wed 4/10/19		Michael Vasile
Game Board	5 days	Thu 4/4/19	Wed 4/10/19		Josh Bugyrn
Bug Fixes in Server/Client	4 days	Thu 4/4/19	Tue 4/9/19		Lucas Kohorst
Updating Design Document for Client/Server	3 days	Mon 4/8/19	Wed 4/10/19		Lucas Kohorst
Webpage Document	5 days	Thu 4/4/19	Wed 4/10/19		Lucas Kohorst
Meeting 3	0 days	Wed 4/10/19	Wed 4/10/19		Michael Vasile

JAR Files	6 days	Wed 4/10/19	Wed 4/17/19		Lucas Kohorst
Chat Client Bug Fixes	6 days	Wed 4/10/19	Wed 4/17/19		Lucas Kohorst
Graph Theory to Code (find the longest path)	6 days	Wed 4/10/19	Wed 4/17/19		Daniel Sause, Josh Bugyrn
Meeting 4	0 days	Wed 4/17/19	Wed 4/17/19		Michael Vasile
Fixing Longest.java to find the longest trail	6 days	Thu 4/18/19	Thu 4/25/19	21	Josh Bugyrn
Server/Client Attributes	**5 days**	**Thu 4/18/19**	**Wed 4/24/19**		**Lucas Kohorst**
Turn Token System	5 days	Thu 4/18/19	Wed 4/24/19		Daniel Sause
Unify GameClient to include GameBoard and ChatClient	**3 days**	**Thu 4/18/19**	**Mon 4/22/19**		**Michael Vasile**
Cards	9 days	Thu 4/18/19	Tue 4/30/19		Michael Vasile

Player Information	9 days	Thu 4/18/19	Tue 4/30/19		Michael Vasile
Turn Token (Display)	10 days	Mon 4/22/19	Fri 5/3/19	25	Michael Vasile
End Turn Button (Display)	10 days	Mon 4/22/19	Fri 5/3/19	25	Michael Vasile
Meeting 5	**1 day**	**Tue 4/30/19**	**Tue 4/30/19**		
Validating the player's name and re-prompt user if nickname exists	6 days	Mon 4/22/19	Mon 4/29/19		Daniel Sause
Display Card Counts	6 days	Mon 4/29/19	Mon 5/6/19		Michael Vasile, Lucas Kohorst
Game Optimization	6 days	Mon 4/29/19	Mon 5/6/19		Josh Bugyrn

Class and method overview

Overview of the classes and functionality.
**** NOTE: Classes are bolded ****

GameClient

- Main method
    - Creates a **GameClient** and a **ChatClient**

- Constructor
    - Creates GUI
    - Locates registry for RMI

ChatClient

- Constructor
    - Create GUI
    - Locates registry for RMI
- **MessageTimer** inner class
    - Class for the messageTimer
- sendMessage method
    - Method to send a message to the **Server**
- getAllMessages method
    - Method to get messages from the **Server**

GameBoard

- Creates the game board and runs the logic for the game
- Communicates with the **GameServer** to send the players moves and update the board from the previous players move

CButton

- A custom class to create all of the route buttons
- Communicates with the GameServer when a button is clicked

GameServer

- Main method
    - Creates a registry for RMI
    - Exports the **Stub** class
    - Binds the **Stub** class to the registry for RMI
- getMessages method (Overridden from **Stub** class)
    - Gets all of the messages on the **Server**
- sendMessasge method (Overridden from **Stub** class)
    - Sends the message from the **ChatClient** to the **Server**
- Contains all the other methods that control the game logic and flow

GameStub

- Interface
    - getMessages method (stub to get messages for the **ChatClient** from the **Server**)

    - sendMessage method (stub to send a message to the **Server** from the **ChatClient**)
- Contains all other methods that control the game logic

Server

- The main server that holds all of the games that have been created

ServerStub

- Contains stub methods for creating and joining a game

Route

- A helper class that contains information about each route

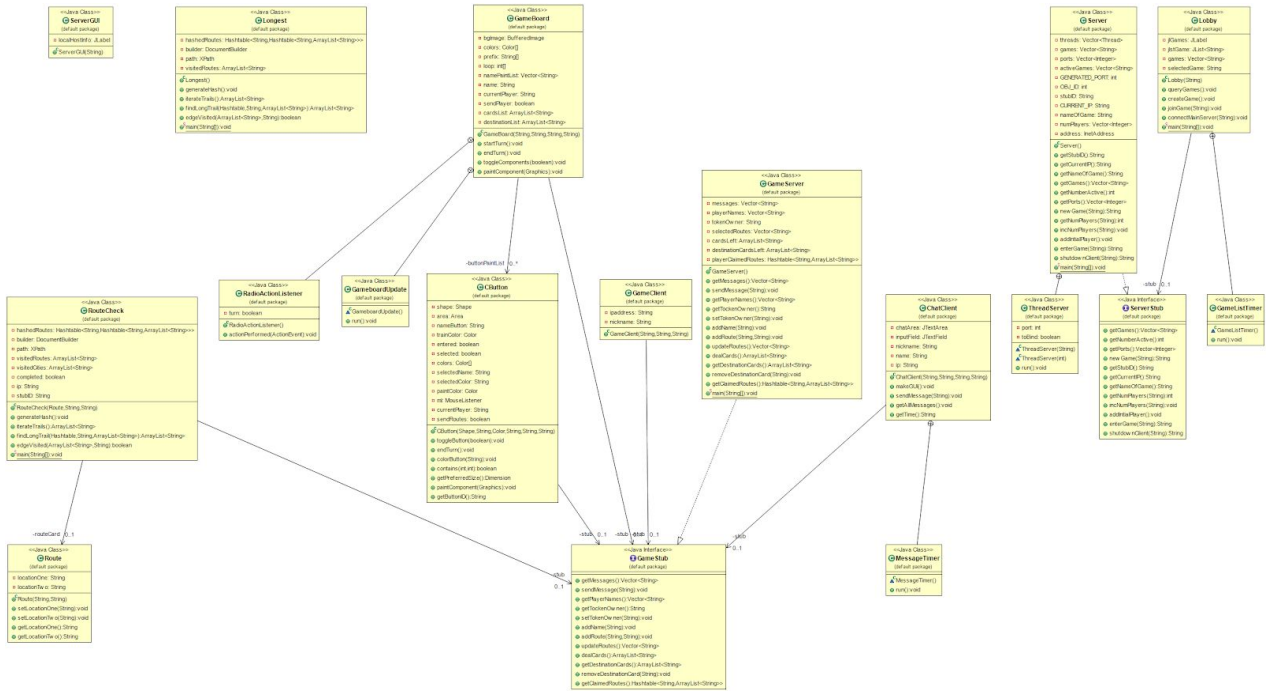
RouteCheck

- Run after the game has ended to calculate the scores
- Generates a Hashtable from all routes that have been claimed
- Recursively finds the longest path in the game

Lobby

- Is the main class that the client creates
- Displays a list of all active games and gives the user an option to create or join a game

UML

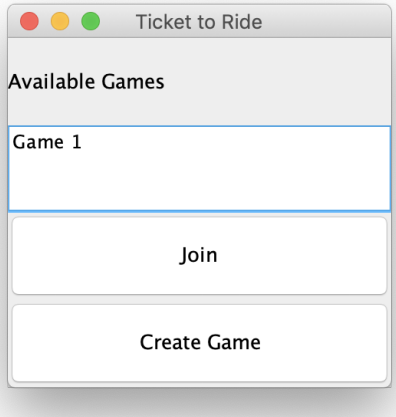


View full UML diagrams https://tickettodavesheart.github.io/ticketToRide/uml.html

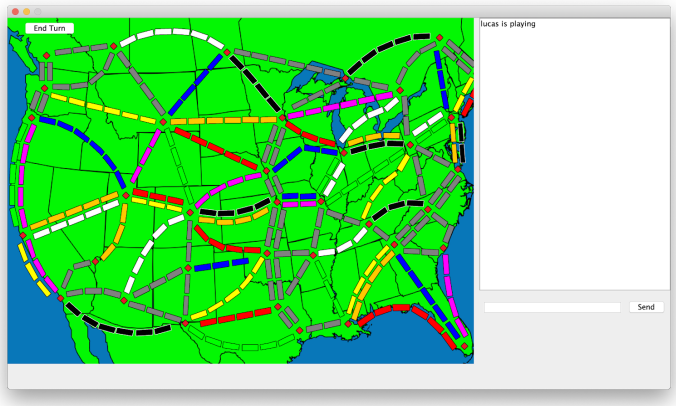
Client GUI

Game Lobby

Displays the active games on the server and allows the client to connect or create their own game



Game Board
The main interface for the client that contains the actual game and a chat for the users to communicate over.



Networking connections & Protocols

Client connection to Server:

- In the **Lobby,** the client specifies the main server's IP
- When a game is created the **Lobby** sends the server the name of the game. The **Server** then creates a new **GameServer** and binds it to a newly generated port. The **Server** then sends the information to connect to this game back to the **Lobby**.

- If a user joins a game in the **Lobby** the **Lobby** sends this request to the main **Server.** The correct information is sent back to the **Lobby** so that the **GameClient** can connect to the specific **GameServer**.
- If a **GameClient** exits the **GameServer** then the **GameClient** creates a connection to the main **Server** and sends it a shutdown request. The **Server** then processes a **GameClient** leaving the **GameServer**.

Port number(s):

- 1099 for the RMI registry
- 16789 + **n** where **n** is the number of games that have been created

Protocol interface code for both client and server:

- RMI is used for all Client and Server interactions
- There is a main **Server** which is bound to the **ServerStub**
    - This class process all connections to active games and creating new games
- There is a **GameServer** which is bound to the **GameStub**
    - A new **GameServer** is created every time someone in the **Lobby** requests to create one. On this request, a new **GameServer** and **GameStub** are bound to a newly created port.
- The **Lobby** is connected to the **Server** using RMI
- Once a user chooses or creates a new **GameServer** the **GameClient** is connected to the **GameServer** on the specified port number.

Game and Communication Classes

GameClient

On connection to the **GameServer,** a game client sends all interactions necessary in the game logic to the server. The server then renders the updated game back to all of the clients.

ChatClient

On connection to the **game server,** a game client starts which renders the chat client. When a user sends a message to the chat the message is sent to the server. The server then sends all of the messages out to the clients.

Client (Game Lobby)	Communication	Main Server

		Startup
Client connection	<< Connection, all active joinable games	Waits for a client to connect Accept connection
Client clicks on join game or on create game	If the user is creating or joining a game with the needed information >>	The server reads information On create a new game and sends the client the information on how to join and adds the game to the list to display in the lobby On Join sends the information on how to join the active game
The client reads the information, and joins the game	<< Server sends back information on how to join the specified game to the client	Sends the information to the client necessary to join the game

Client (Game)	**Communication**	**Game Server**
		Startup when bound from the main server
Client connection	<< Connection, all game info	Sends all game info to client
Client makes a move	Move information is >>	The server reads information and process it in the game
Client sends a message	A message is sent >>	The server reads information and process it for the rest of the clients chat areas

Data used

{List and describe each file, URL, network connection being used.
Do not include the data.}

map.png - The image of the US map behind an instance of the GameBoard.

Data files

buttonlist.dat - contains all formatting and placement fo buttons on the GameBoard.

routes.xml - contains all route information (i.e. names of routes, cities on each end of the route, route numbers, etc.)

Punch List used

To do:

- Calculate the points a player has to determine the winner
- Display the train and ticket cards a player has
- Display the decks for choosing new ticket and train cards
- Detect when a player has 3 or fewer trains left to end the game

Done:

- Longest.java outputs the city with the most adjacent cities instead of the longest path.
- Validated names
- Assigned unique colors to each player and sent claimed routes over the server to update all clients game boards

Unresolved Issues

- Unable to limit players per game (low priority)
- Clicking on a route has about a 70% chance that it will change color (high priority)