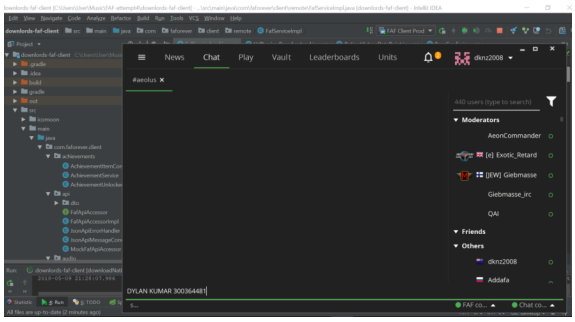
downlords-faf-client

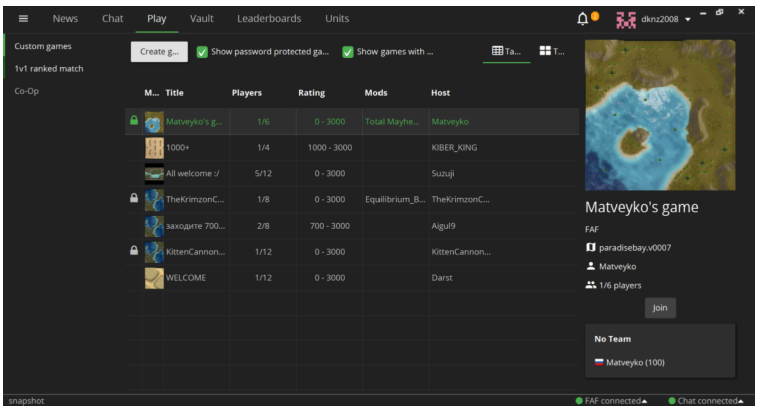


Built on IntellJ Ultimate Platform. Make sure to import project as gradle project. Then had to go alt-9 (version control), and revert the deletion of XML files for run configuration (as instructed on the repo), because IntellJ automatically deletes them for whatever reason. Lastly had to go to the right, gradle, refresh (and if run tasks in native dependencies if you get Uid error). Run 'FAF Client Prod' and set the 'class path of module' if it isn't already set. This video may help you (shows setup of FAF): https://goo.gl/EDj26i

History:

The 'downlords-faf client' project began as a game launcher for the video game Supreme Commander: Forged Alliance. The official support for the game ended by Gas Powered Games (GPG) in 2009, and the multiplayer servers GPGnet were shut down in 2012 [1]. Many in the Forged Alliance community still wished to support the development of the game, especially on the multiplayer side. Forged Alliance Forever started from a community of people that wished to do this. The original multiplayer client was first created in 2011. Forged Alliance Forever projects continue to be updating regularly with a large supportive community. These include community patches, new maps/mods, replays, live replays and much more [1]. Even new Game modes and campaigns were made. On June 23, 2015 v0.5-alpha was released for 'downlords-faf client'[2]. This was a new Java client that was supposed to replace the old client written in python. This new client lets you launch the game in a multiplayer lobby, features IRC chat, and other features. The primary contributors to the project are 'micheljung' who started the project, 'Cheyans' and 'axel1200'. The downlords-fav client has been licensed as an open source MIT project. For the rest of this report the 'downlords-faf client' shall be referred to as the FAF Client or just Client for simplicity.

Domain:



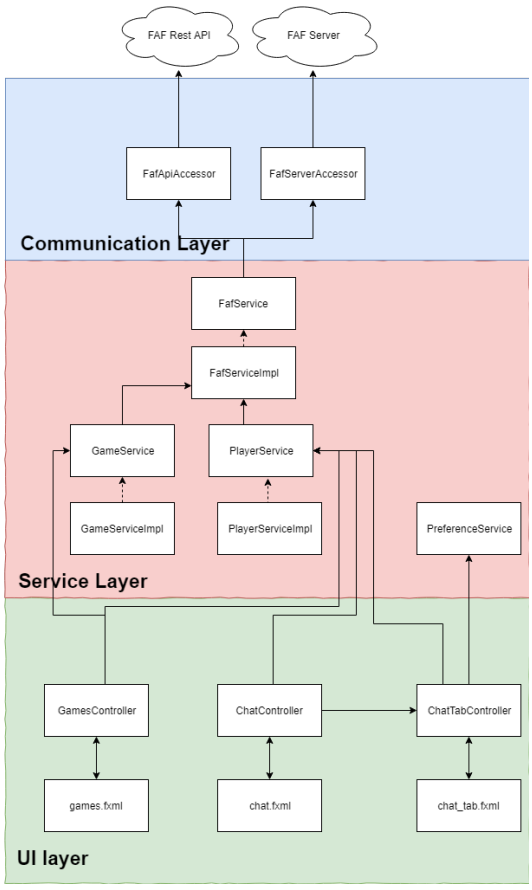
The FAF client's primary domain is to handle the multiplayer configuration side of the video game, Supreme Commander: Forged Alliance, as well as provide an environment where users can connect with each other. The FAF client allows you to

browse games that other users have started, and shows basic descriptive statistics such as the map, title, number of players, mods, and host a game via the GUI. The games that you can join can be private or public matches; with private matches requiring authentication via a shared password to join the lobby. Once you join a lobby, Supreme Commander: Forged Alliance will launch, presuming that you configured the installation directory in settings. You are still able to use the FAF client without owning or having the game installed on your system.

The FAF client requires you to sign up for an account, and login to view this information. This imposes a constraint that you have a valid email address, to be able to sign up for an account, and an online internet connection so that you can play with others, as well as see data extracted from requests to the API to Server.

The subdomains of the FAF client also include a leaderboard of the highest ranked multiplayer users, information of the in games units such as pictures and statistics, a news board of information about the client and community, as well as an IRC chat, and messaging client. The IRC chat/messaging client allows users to communicate before they join a game. These are all interactable elements in the GUI.

The FAF Client is limited to run on the Windows and Linux operating systems. Running it requires the Unix Uid or Windows Uid to be installed; which are native dependencies for the operating system you are using and are installed through gradle or the Install4J installer. The client requires that you have some way of navigation (e.g. mouse) and way of inserting characters into a GUI (e.g. keyboard, or on screen keyboard).



Component Architecture

FAF is split into three primary layers:

1) The Communication Layer; whose purpose is to assess information from the Server and API.
2) The Service Layer; which provides functionality for the various services such as games, players or maps service
3) The UI Layer; which controls the GUI and visual layout and display of the View.

The Communication layer consists of Assessors which query the server for information. The job of the Accessors is to communicate with external services; those being the FAF

REST API, and the FAF Server. Assessors understand how to talk to a specific server and what function they offer. Each assessor is used by exactly one service, though a service may use multiple assessors.

The Services then access this information through calls to the Accessors. The Service classes handles data manipulation, such as the loading of information depending on preferences, situation and their respective domain. Each Service provides the functionality of exactly one domain. For instance, for the domain “maps” there is a **MapService** and for the domain “players” there is a **PlayerService**. These services handle the functionality and data manipulation of the FAF client, which are then fed into the Controllers which specify the users interaction with the FAF client.

The Service Layer is made up of two layers: the Service Interface Layer, and the Service Implementation Layer. Currently every Service implementation is implementing a Service Interface, which is a blueprint of method declarations. In the majority of cases however, only one object is implementing the particular Service causing redundancy as a particular Service often only needs to do one type of thing; not multiple, whilst controllers are passing in these Services in their constructor as their interface type, not implementation.

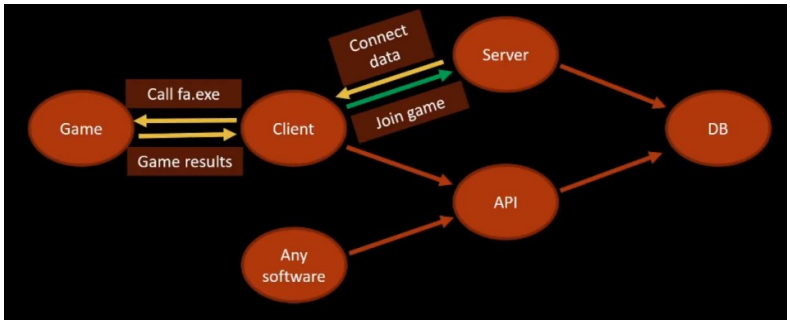
The Controllers are responsible for managing UI, and are part of the View. Each Controller has one FXML file associated to it. FXML files are JavaFX markup files that handle the layout of the View of the system. These FXML stylesheets represent the different graphical panes of the GUI. The FXML stylesheets contain the layout, associated Controller, ID’s of the different GUI elements such as textField’s, and other information relevant to the layout of the view. Interacting with the user interface tells the controller to display user interface elements differently, and may result in a GUI element such as a textField to have its value changed, which will then show up on the View due to the associated ID. The Controllers make calls to the Services, and therefore chooses what information to show on user intention and the information received from the Service. Services do not depend on Controllers, while a Controller may depend (make calls to) one or more services for functionality that it may require.

The FAF client uses Spring Boot to tie together the various components of the system together. The Controllers are not actually being called anywhere ‘visibly’ in the code. Instead it looks at the annotations of classes, for instance, the @component annotation for controllers and @configuration for configuration files [3]. Spring Boot is a framework designed to reduce boilerplate configuration and handle the life cycle of objects by automatically injecting dependencies when needed, and allow for cleaner and more modular code. Using @component tells Spring Boot to make those controllers Spring beans. Beans are objects that are managed by the Spring IoC (inversion of control) container which handles the location and state of dependencies when needed [4].

Spring Boot uses the annotation '@inject' to automatically inject dependencies into the constructor of the class. Dependencies are other objects that your class needs to function. This means that code written with Spring Boot does not have many objects manually being created; instead it takes them as a parameter in the constructor that are automatically injected in from Spring Boot.

The Client can send messages through the ServerAccessor to login to server, and then host, join games or start a game. The Server tells the client periodically the list of games that are available and new messages and users from the chat. The API was built to offer more general services. These would allow third party developers to make calls to the API to offer some other service or integrate their service into FAF.

The Client can host a new game or join an existing game. When a user wishes to join a game, the Client sends a request to the Server. If the Client is able to join the game, the Server sends the connect data to the Client, such as the IP address of the host of the game, the Map, Mods to use, as well as other game options. The client uses this information to call the game Supreme Commander: Forged Alliance (fa.exe), and makes a new process for the game (opens it), with a set of command line arguments that includes the IP address of the host.



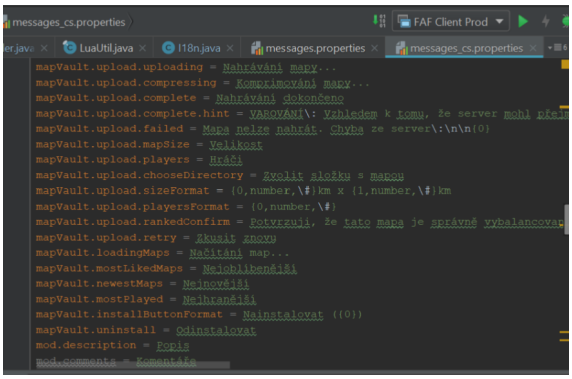
After the game has begun, the Client only forwards Server messages to the Game and Game messages to the Server (such as if game options are changing in the Lobby). Once the Game has ended, the Game sends the results to the Client, which forwards it to the Server, and the Server stores it in the database. An example of this is shown in the diagram above [5].
The in-game networking between users is managed by pre-existing code hardcoded into the Game and is seperate from the Client.

Data Structures:

The main data structures are Collections such as Lists and Maps that are used to store information that are to be displayed on the View. These Collections are populated from calls to the API and Server made by the Accessors.The API and Server query an SQL database to retrieve some of this information. The FAFApiAssessor assesses the FAF REST API over HTTP to get non core services such as as Achievements, Map Upload, etc.

Once the Client has authenticated the user via OAuth 2.0, the Client can send or request any data specified in the API, such as getting the Players Achievements, Maps, Mods, Replays, Players, or uploading or downloading a map.
On the other hand, the FAFServerAssessor listens for new messages, handles the login of users to the Client, searching for new games, adding and removing friends, and other server requests.

Some of these collections include lists of Players, Mods, Events and Leaderboards, that are populated from queries to the Server or API from the Assessors (communication layer). These are then stored in the Services, which handle the data processing. For instance, cookies are stored in cookieServiceImpl which are a map of URI object to List of HTTPCookie Object.



The FAF client uses internationalization to allow for multiple languages and for GUI elements to adapt to these languages. Separate style sheets have been made for multiple languages for the different GUI elements and text fields. These are handled by calls to an i18n object, which handles the users preferences regarding text and language. For the six currently implemented languages there are .properties files in resources/i18n/... that are associated to each language. The language is defaulted to look at ‘messages.properties’, otherwise separate .properties file for the specific language. This data structure allows for a fast way to change elements in the View, whereby new languages to be added easily.

[1]"History Of Supreme Commander - FA Forever Wiki", Wiki.faforever.com, 2018. [Online]. Available: https://wiki.faforever.com/index.php?title=History_Of_Supreme_Commander. [Accessed: 11- May- 2018]

[2]"FAForever/downlords-faf-client", GitHub, 2018. [Online]. Available: https://github.com/FAForever/downlords-faf-client/releases. [Accessed: 11- May- 2018]

[3]"Spring Framework Annotations - Spring Framework Guru", Spring Framework Guru, 2018. [Online]. Available: https://springframework.guru/spring-framework-annotations/. [Accessed: 11- May- 2018]

[4]"5. The IoC container", Docs.spring.io, 2018. [Online]. Available: https://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/beans.html. [Accessed: 11- May- 2018]

[5]"Developer Tutorial 1 - Architecture Basics", YouTube, 2018. [Online]. Available: https://www.youtube.com/watch?v=lUUDdL05QAA. [Accessed: 11- May- 2018