

Online Draughts

Custom Rules:

This application consists in a modified version of the draughts game.

The game take place in an 8x8 board where each player starts with 12 token.

The player can move the token forward or backward in oblique direction and eat only one enemy token at time.

The games ends when one player lose all the tokens.

Tecnologies used:

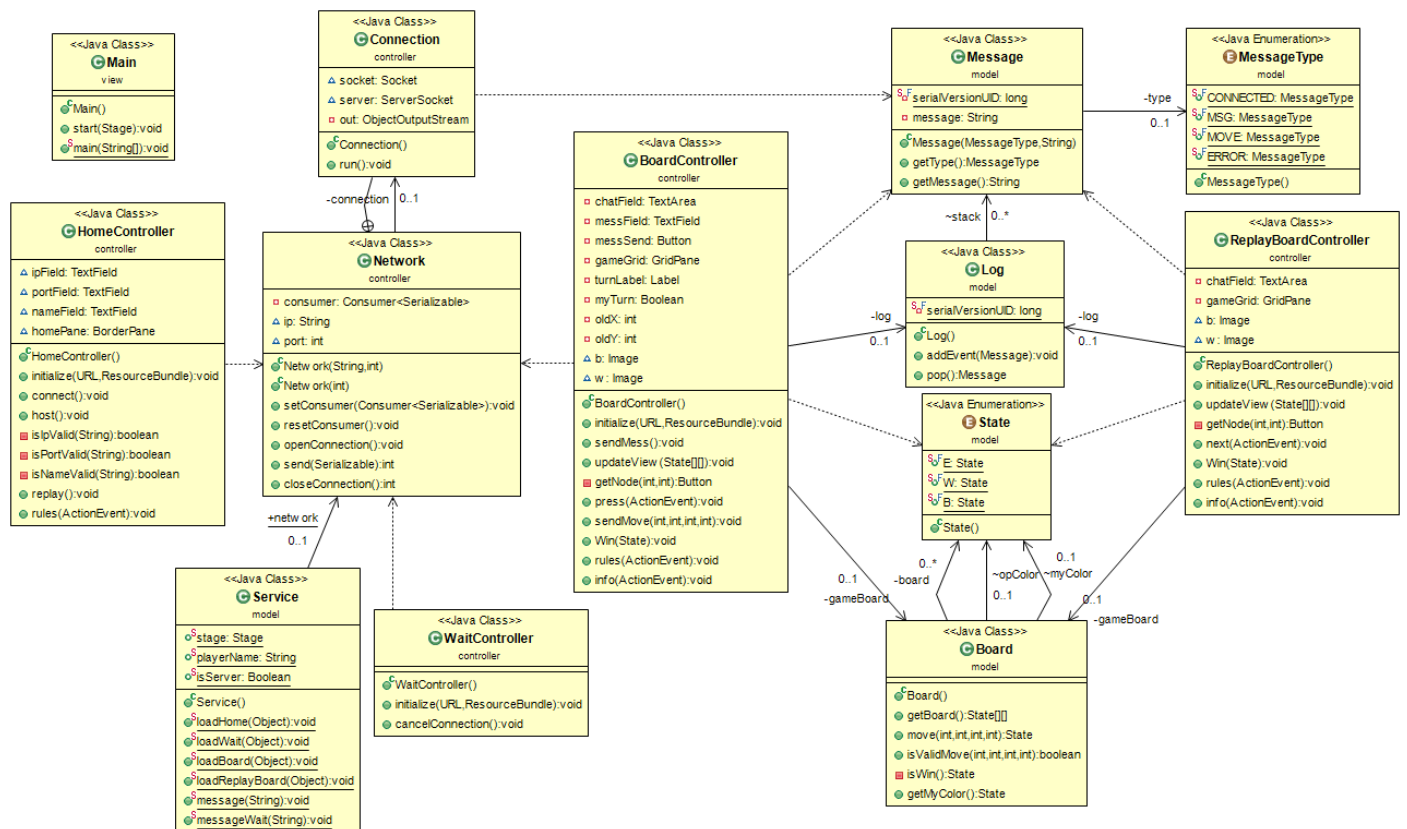
The application is written in java programming language, using the MVC architecture and Eclipse IDE.

It use libraries like javafx, java.net, thread and the UI interface is written in fxml and css.

Extra information

The Project is about 1500 lines of code, for more detailed information about the single classes and functions please refer to the Javadoc attached with the project.

UML



Classes

Main: The entry point of the application, it will load the homeController and home UI.

HomeController: The controller of the home screen, it performs field checking and the buttons functions

WaitController: The controller of the wait screen it will take care of the establishment of the connection.

BoardController: The controller of the game it will provide functions for the execution of the games.

Board: The model of the board, it stores the board and is able to check if a move is valid or if there is winner.

State: Represent the state of a single board's space, it can be empty, black, or white.

Message: Instance of this object are sent through the net, used for communication between Client and Host.

MessageType: It represent the type of the Message instance.

Log: It stores all the Messages exchanged during the match.

ReplayBoardController: Similar to the BoardController it provides function for show the previous match

Network and Connection: Those class provide function for send and receive messages throw the network.

Service: This is a special class with a lot of static field and static function that are used by the controllers for perform their duty.

Start of the application

When the application is started the **Main** initialize the **HomeController** and load the home UI, at that point the application is waiting for the input from the user, that can choose between 4 options:



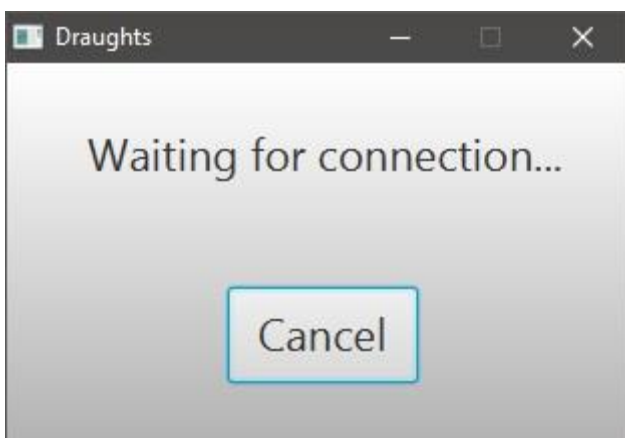
The screenshot shows a window titled "Draughts" with a standard macOS-style title bar (minimize, maximize, close buttons). The window contains three input fields labeled "IP", "PORT", and "Name". Below these fields are four buttons arranged in a 2x2 grid: "HOST", "CONNECT", "REPLAY", and "INFO". The "Name" field has a cursor inside it.

Host/Connect button

Both of those buttons will initialize a network object inside the service class.

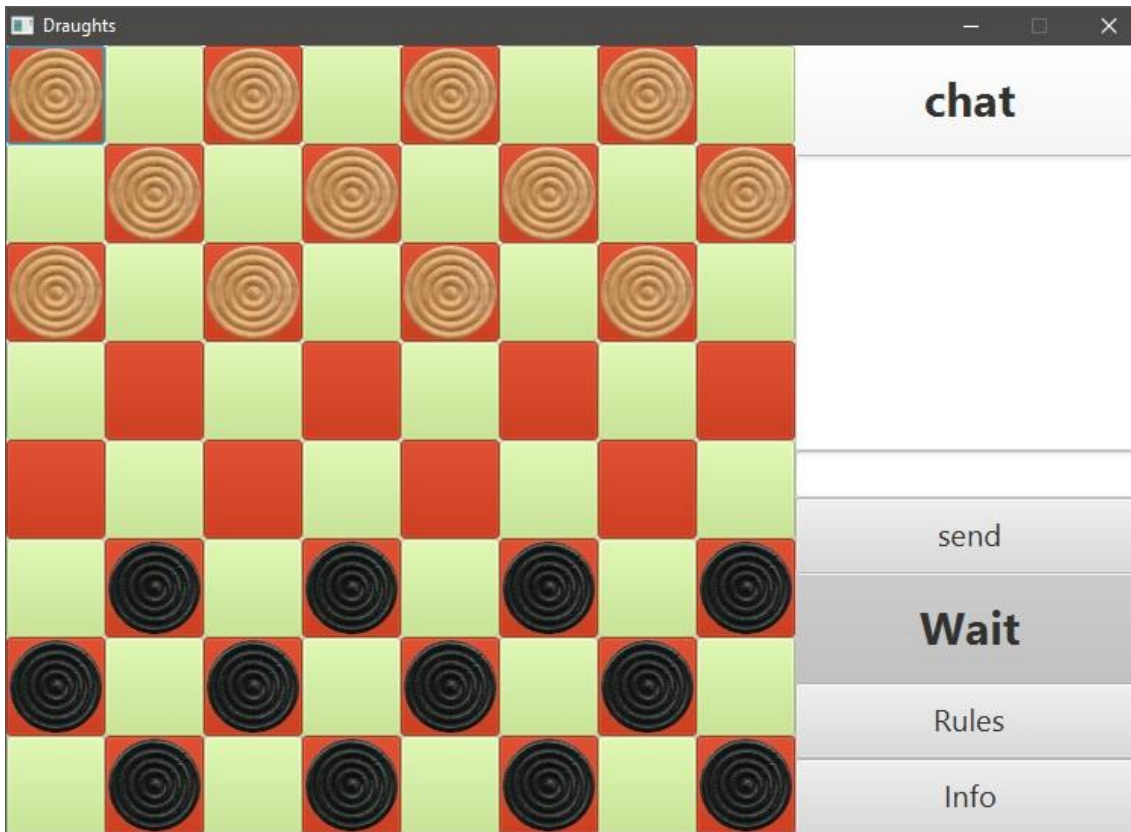
Client: check the input and try to connect to the server loading the **WaitController** and wait UI. Set the application as client in the **Service.isServer** field. Once in the Wait screen the client will send through the network a **Message** object of **CONNECTED** type, then the Gameboard is loaded.

Server: check the input and wait a connection loading the **WaitController** and wait UI. Set the application as server in the **Service.isServer** field. Once in the Wait screen the host should receive through the network a **Message** object of **CONNECTED** type, once received the Gameboard is loaded.

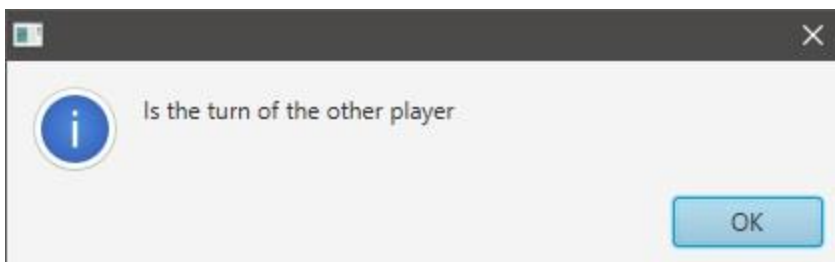


The screenshot shows a window titled "Draughts" with a standard macOS-style title bar. The window contains the text "Waiting for connection..." in a large, centered font. Below the text is a single button labeled "Cancel".

In the initializing process the **BoardController** will initialize a new **board** object, set a new consumer for the received messages and load the new UI.

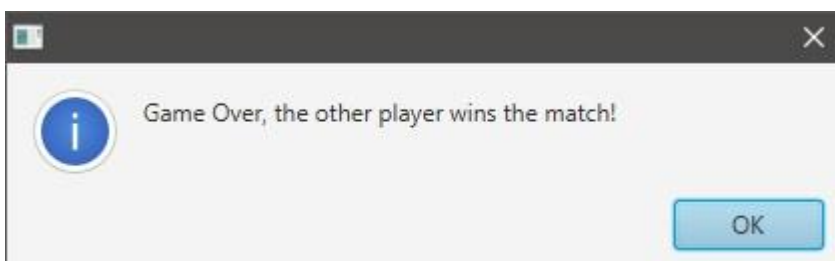


Once the game is started the host will have the first turn, in case a player tries to perform a move not in his turn an error will be displayed.



For move the token the player have to press before the token that he want to move and after his final destination if the move is declared valid by the **Board.isValidMove** function the game will accept the move (**Board.move**) update the view (**BoardController.updateView**) and send the move through the network (**BoardController.sendMove**).

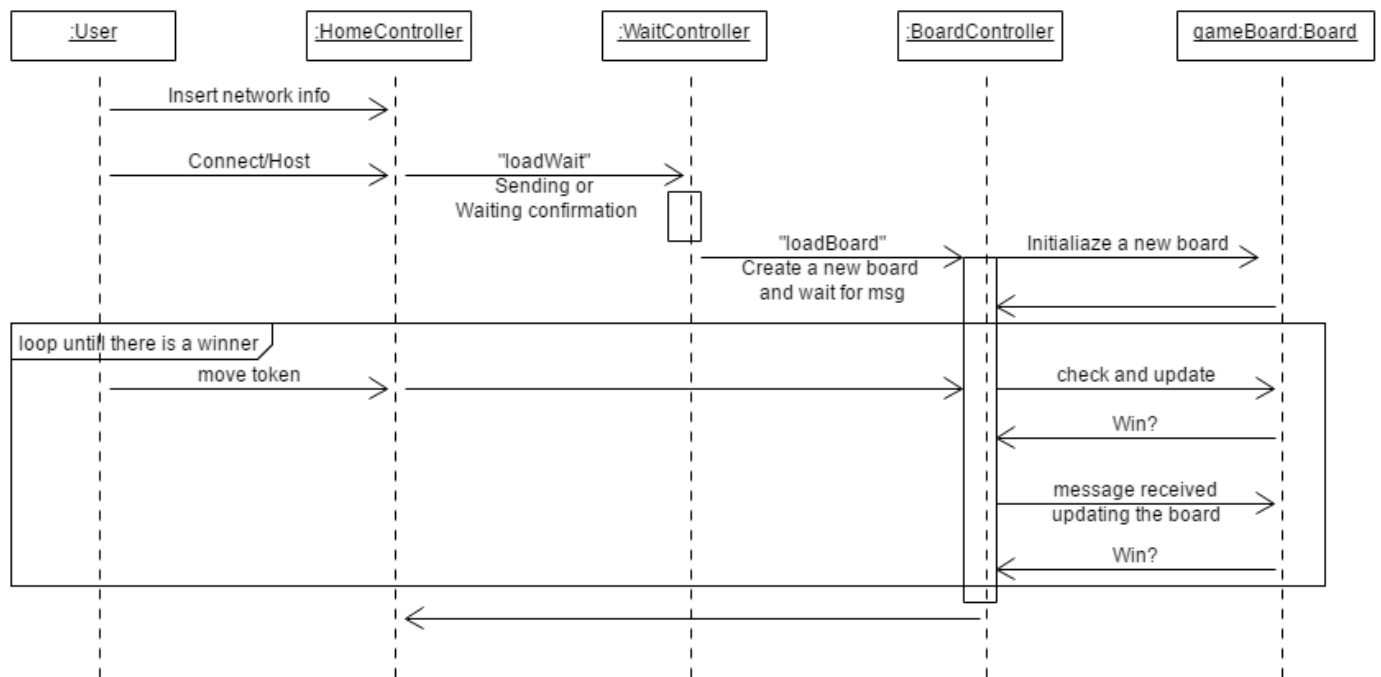
Every move **Board.isWin** check for winner and a message will be send in case someone wins .



If a chat message is send by the player the game will send it using the **BoardController.sendMess** function.

Each **Message** sent or received will be saved in a **Log** object and stored in a stack that is saved on a file named **log.dat**.

Sequence of a normal game

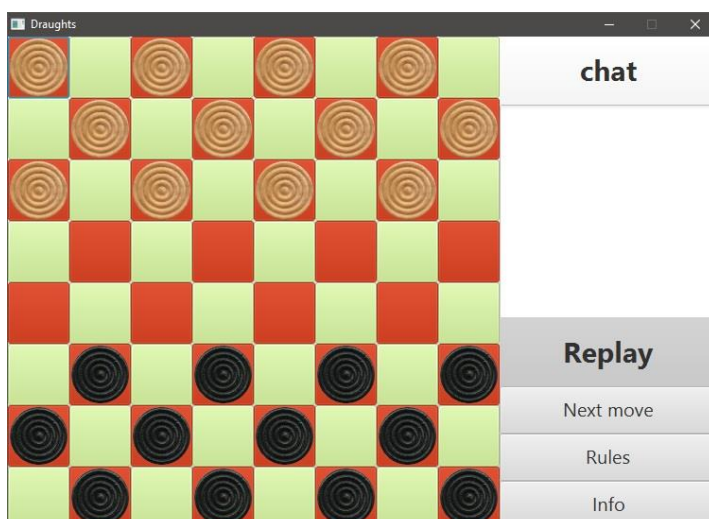


Replay

When the replay button is pressed the HomeController check if the file log.dat exist and start the **ReplayBoardController**, if doesn't exist it will send an error message.



The **ReplayBoardController** will load the log.dat file and will start to pop the **Message** objects when the player will press the **Next move** button and update the view using the information inside the message.

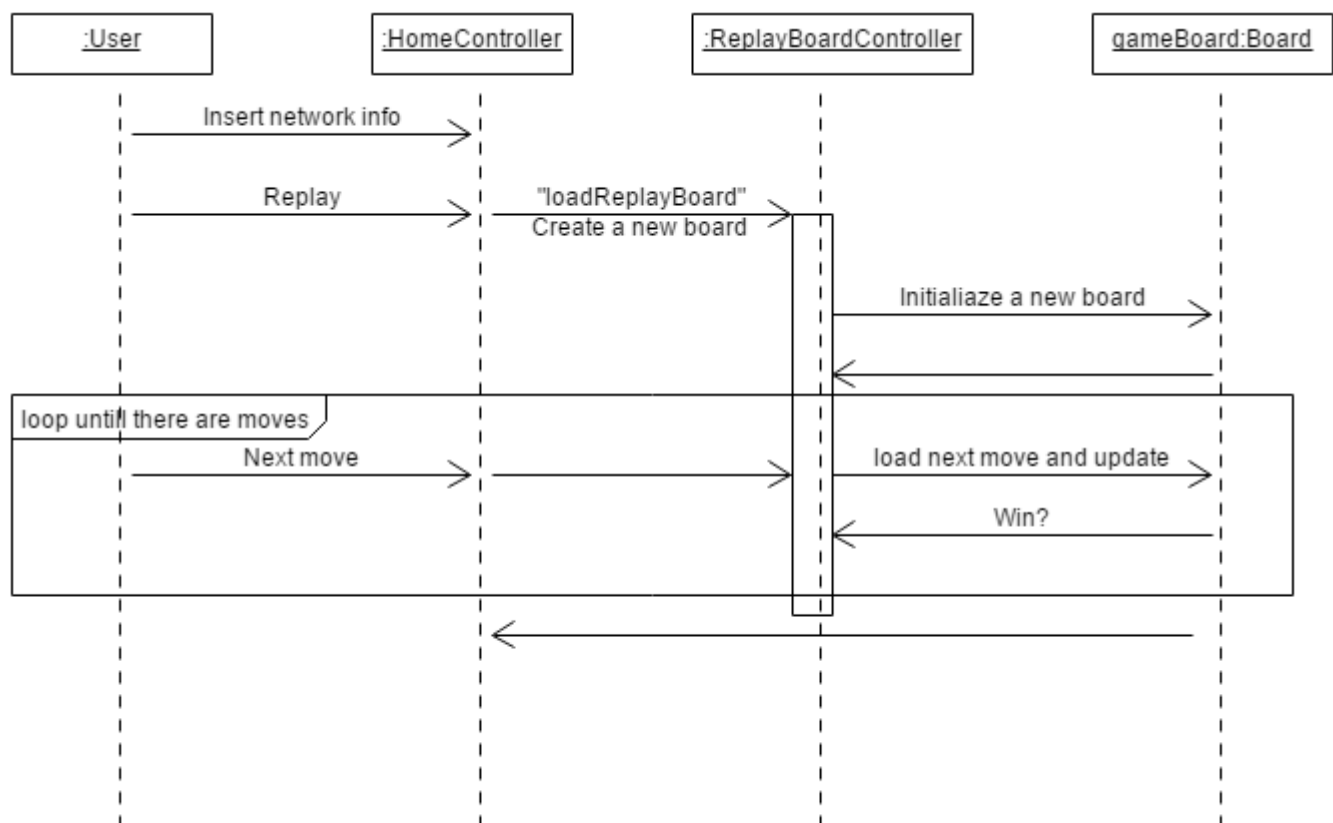


Once the stack of Messages is empty a message will be send to the user saying that the log is over.



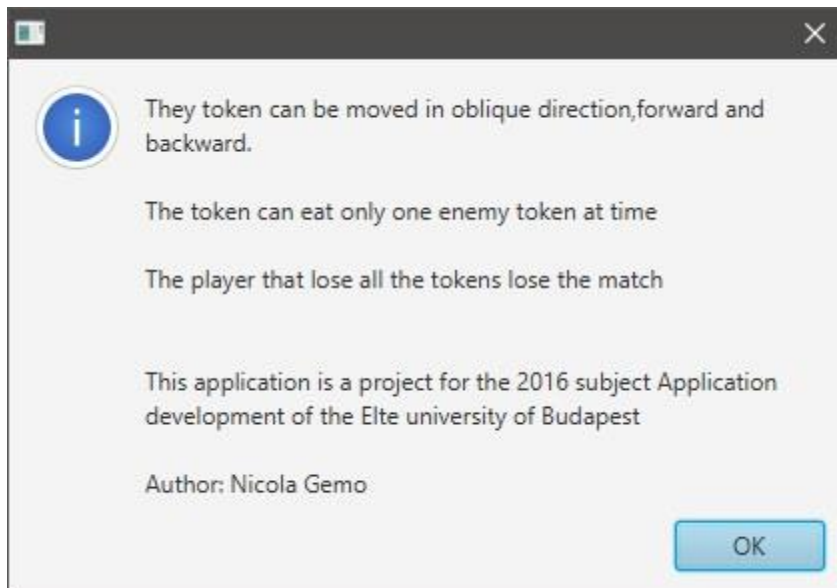
Then the application will return to the home screen.

Sequence of a replay



Info

It will show a message with the rules and information about the application.



Network and communication:

For the communication the **Network** and **Connect** class are used, the first is used to manage the TCP connection and the second one contain the connection itself that run in a separate thread, a consumer is used for manage the incoming message that can be set in different ways by the Controllers.

For send information between the games a **Message** object is sent through the network that can be of different types such as **MSG** or **MOVE** and contain a string that will be processed accordingly by the controllers.