Parallel & Distributed Processing

Java JMS — Playing The Twenty Four Game

Setup & Execution

The application is bundled as a package titled `*Twenty Four Game*`. For the convenience of the user/tester "build scripts" have also been provided. The package has been thoroughly tested on both MacOS (x86) & Ubuntu. To build and execute the program, follow the proceeding steps:

1. Change your directory to the one which contains the package along with the build scripts and external libraries [see figure below].

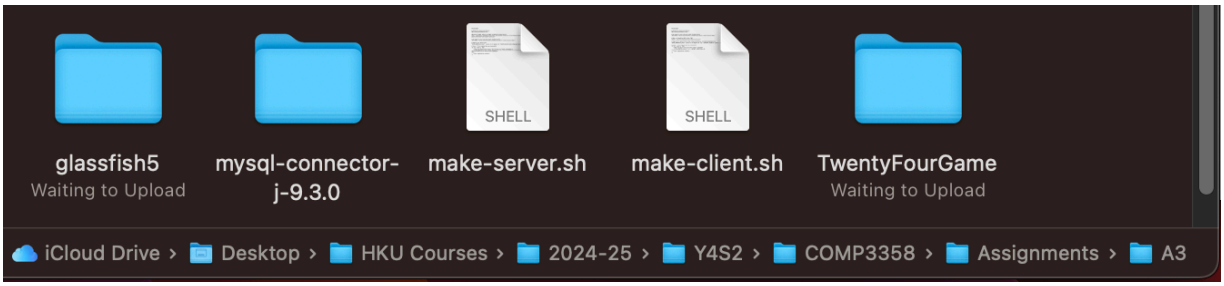


Fig 1a. The directory housing the project

2. Setup JMS services. Run the following command in your terminal (make sure glassfish is installed on your machine, you may also use the binary provided in the submission to run this otherwise)

```
as-asmin start-domain
```

3. Setup the resources as shown in the figure below. You need to setup a Connection Factory, a JMS queue and a JMS topic with the names as specified in the figure to the side. The GameQueue is a *Javax.jms.Queue*, the GameTopic is a *Javax.jms.Topic*.

Fig 1b. Setting up jms on *localhost:4848*

4. Make sure that a mysql instance is running on you machine, with the correct tables. See A2 for the setup of MySQL and the required tables.

5. Ensure that your system has Java 8 (also know as Java 1.8). Then run the following commands, in order:

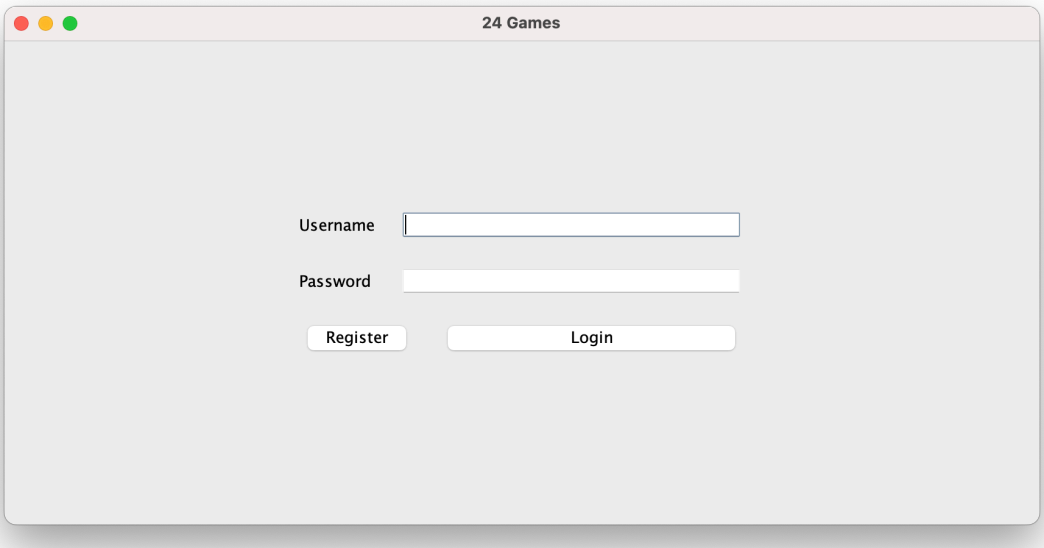
```
rmiregistry &
bash make-server.sh &
bash make-client.sh localhost
```

The first command sets up the rmi-registry locally, while the next command sets up the server locally where the rmi registry is launched. This can be set up remotely as well. Then for the client program, we would be required to pass in the ip of the remote server. For convenience we are testing everything locally. If the user/tester has access to multiple terminal sessions then running the first two commands in a separate shell than the last command will be *cleaner*.

Upon successfully executing the first two commands you should see the message `AuthenticationManager service registered`, along with other warnings about deprecation and security that can be ignored safely because our application has no users to steal money or data from. You will also see messages detailing the JMS classes being loaded.

If you are using a different version of Java you can modify the script to point to the specific version of Java 8 by modifying the script (e.g. replacing `Java` with `./path/to/my/Java_8/binary` in both the *make-client.sh* and *make-server.sh* scripts. You can also adjust the paths to the mysql JDBC connector plugin and ther glassfish installation in the build scripts if you encounter any issues.

6. Once these commands have run, a window running the application should pop up as long as your system has a GUI. If not please consider switch a system that has once to use the application.



Once tired, you may want to run the command `*rm TwentyFourGame/*/*.class*` to clean the package.

The following application was run on MacOs for the sake of convenience but will compile and run on Ubuntu just as well. It has been tested on my ubuntu machines as well.

Features

Fig 2. Logged in as Player ‘A’, fresh profile

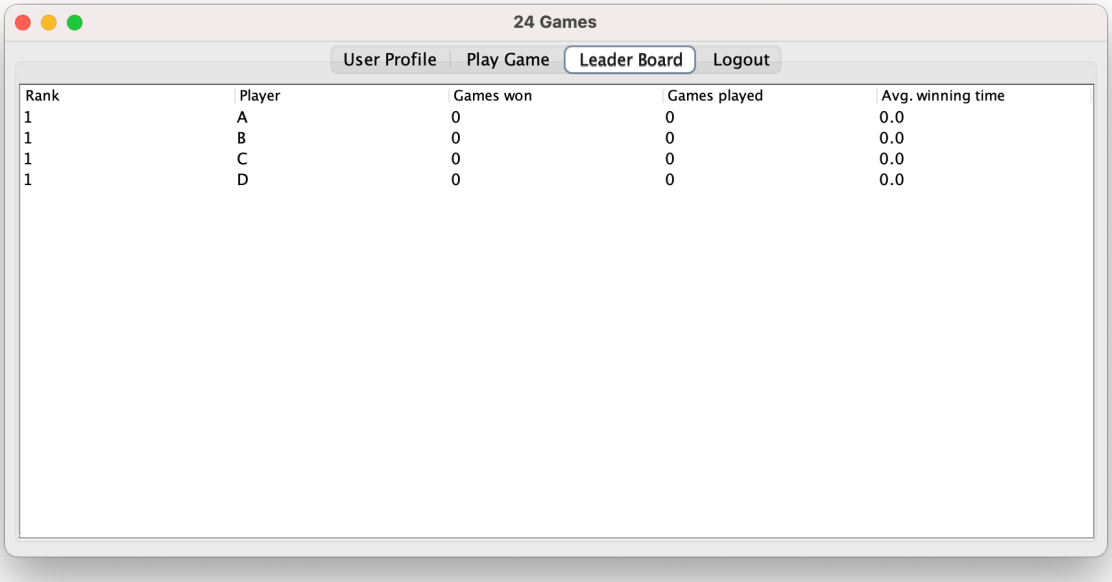


Fig 3. Leaderboard displays all four newly registered players

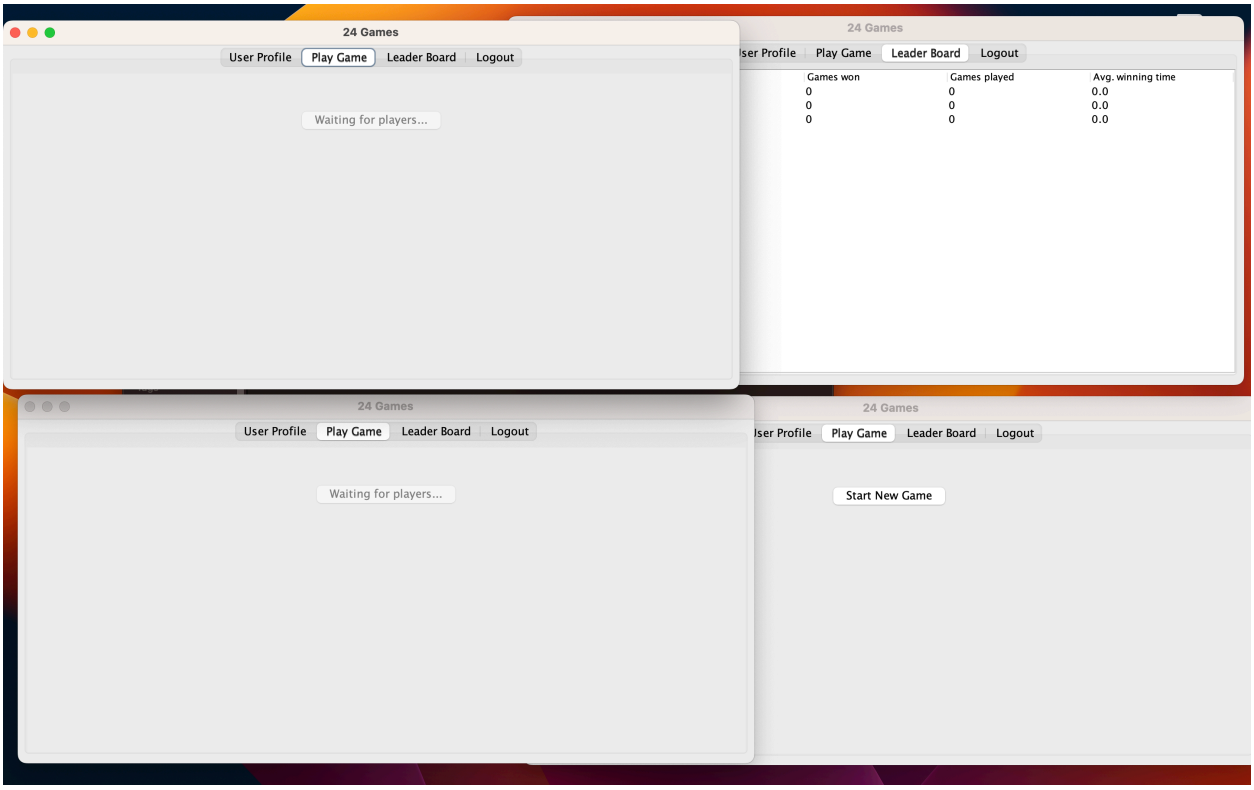


Fig 4. Start game with two players, waits until 10 secs have elapsed

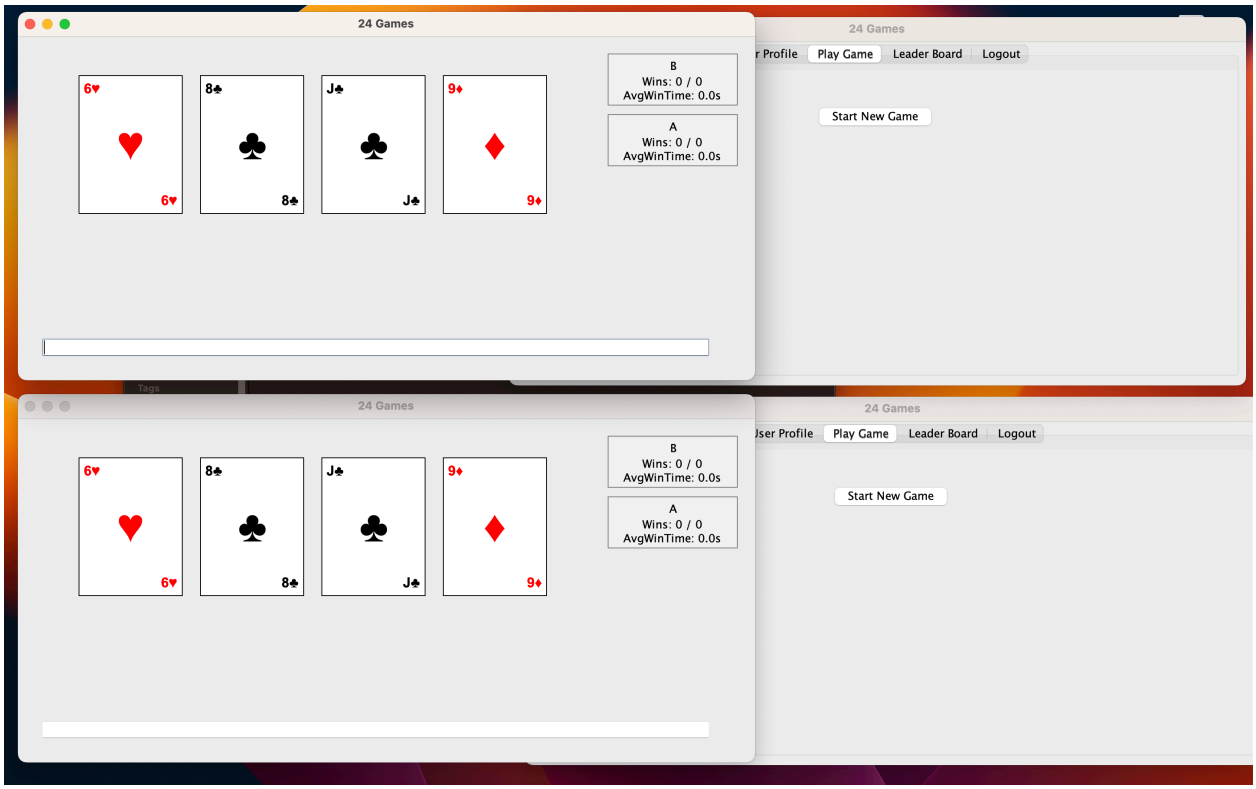


Fig 5. Play game with two players

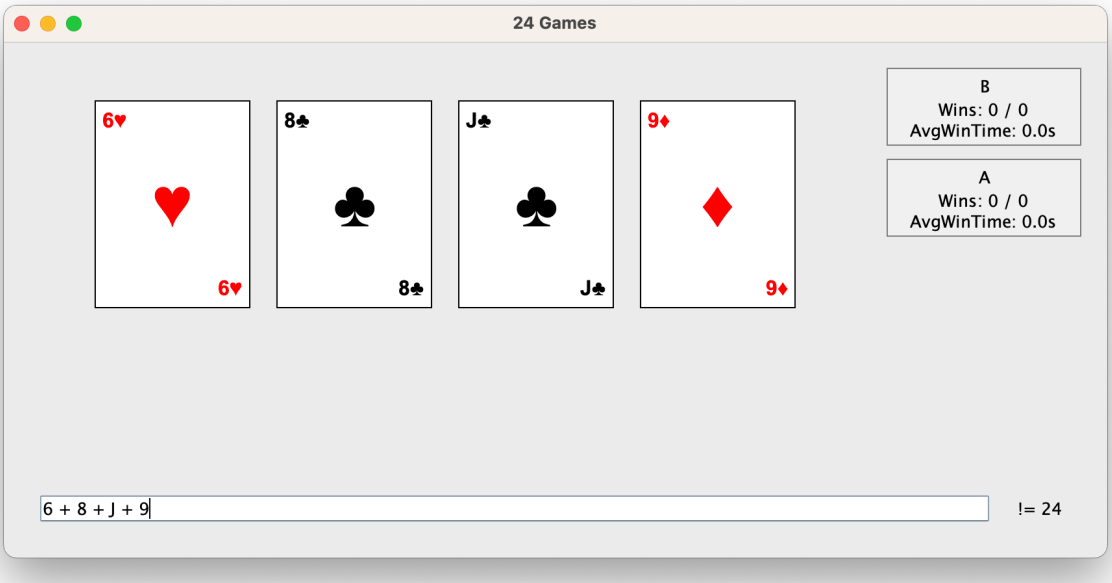


Fig 6. Trying an incorrect expression

Fig 7. Legal cards, evaluation result (if valid expression) logged to standard error. Any other errors with invalid expressions will also be logged here.

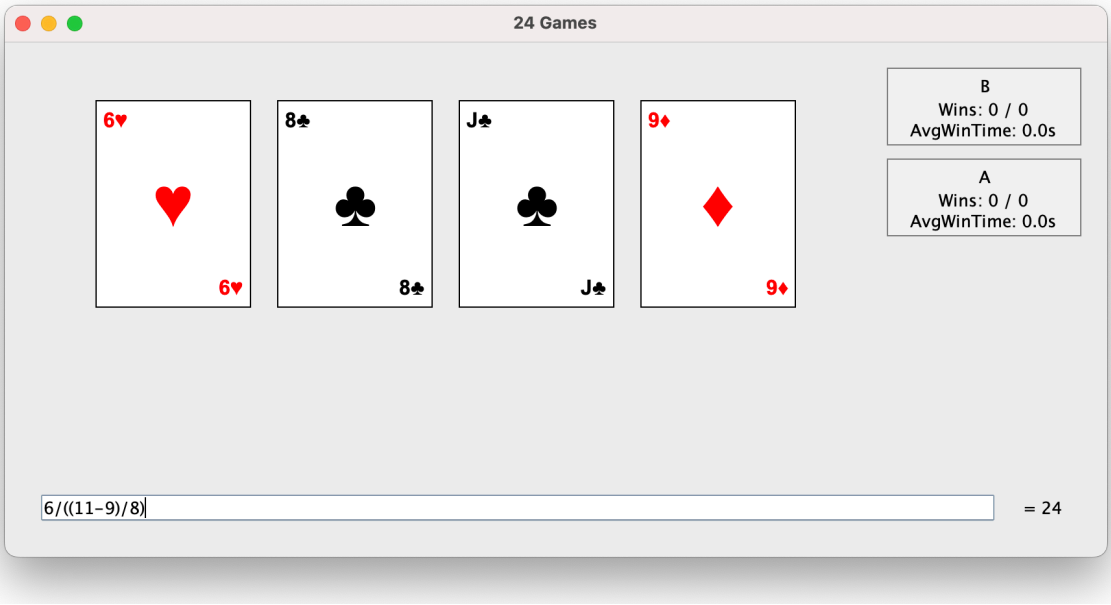


Fig 8. Plugging in a valid solution ends the game for everyone, updates the statistics and broadcasts it to everyone

Fig 9. The screen after game ends. Same screen is shown for the winner and other players that were in the game. Since only one game can be played at a time even if players are logged in (and not in the game) they will be able to see the result after the game ends. See Fig 10

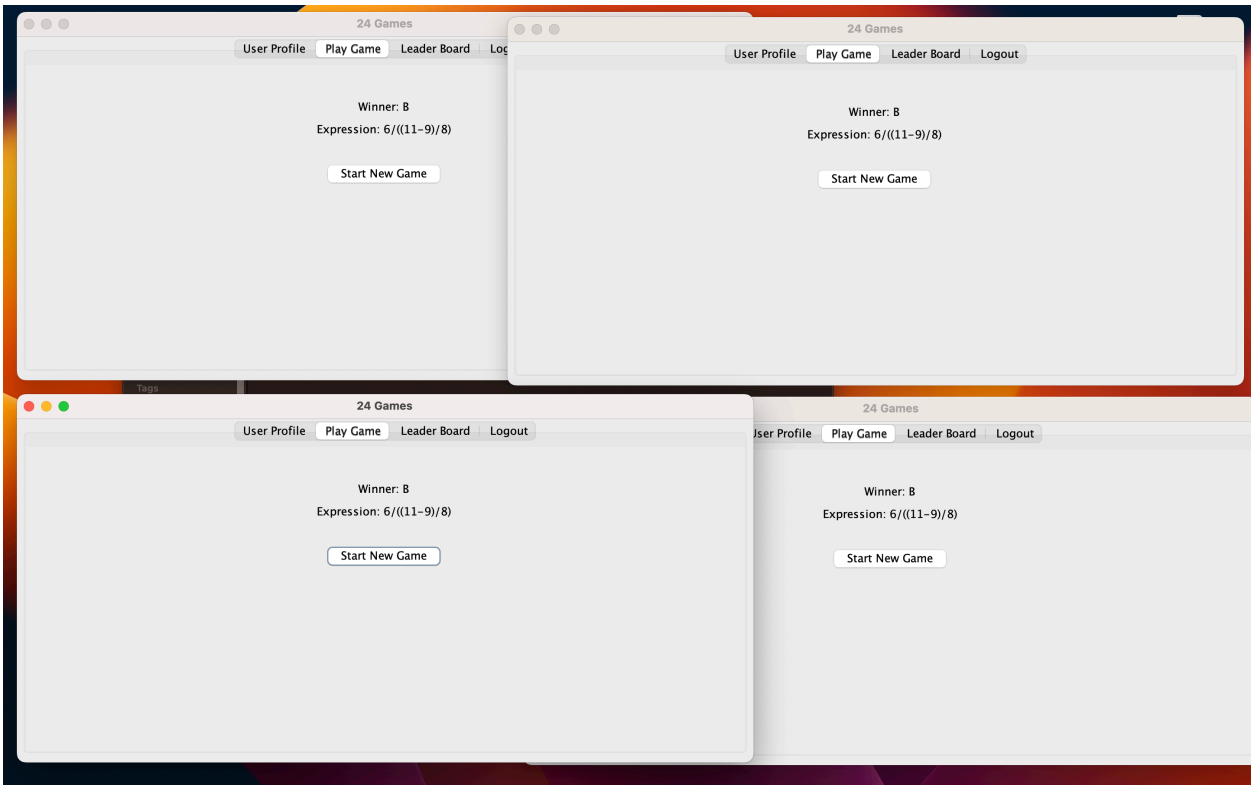


Fig 10. The result broadcast to all players logged in and the option to begin a new game

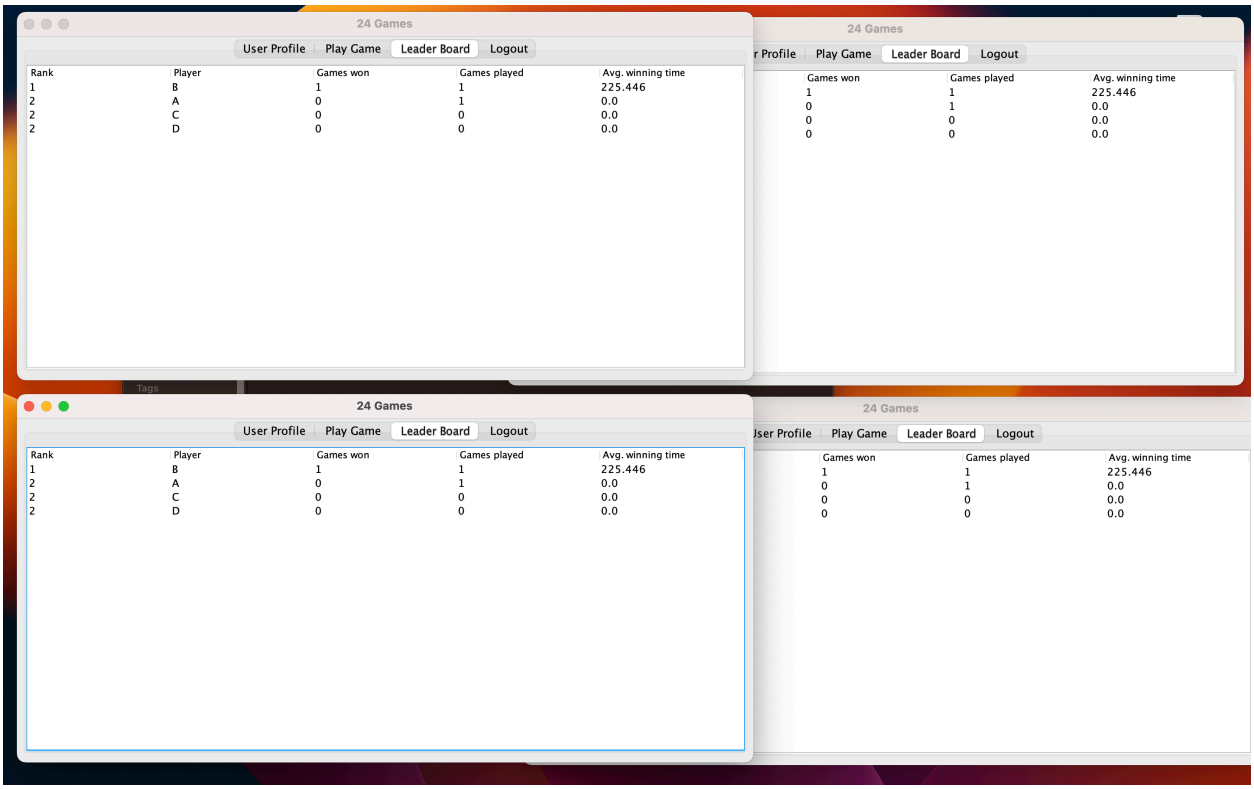


Fig 11. We can only see that the leaderboards for all users have been updated with the correct avgTime, number of games played and number of wins. Let's try a new game with all four players.

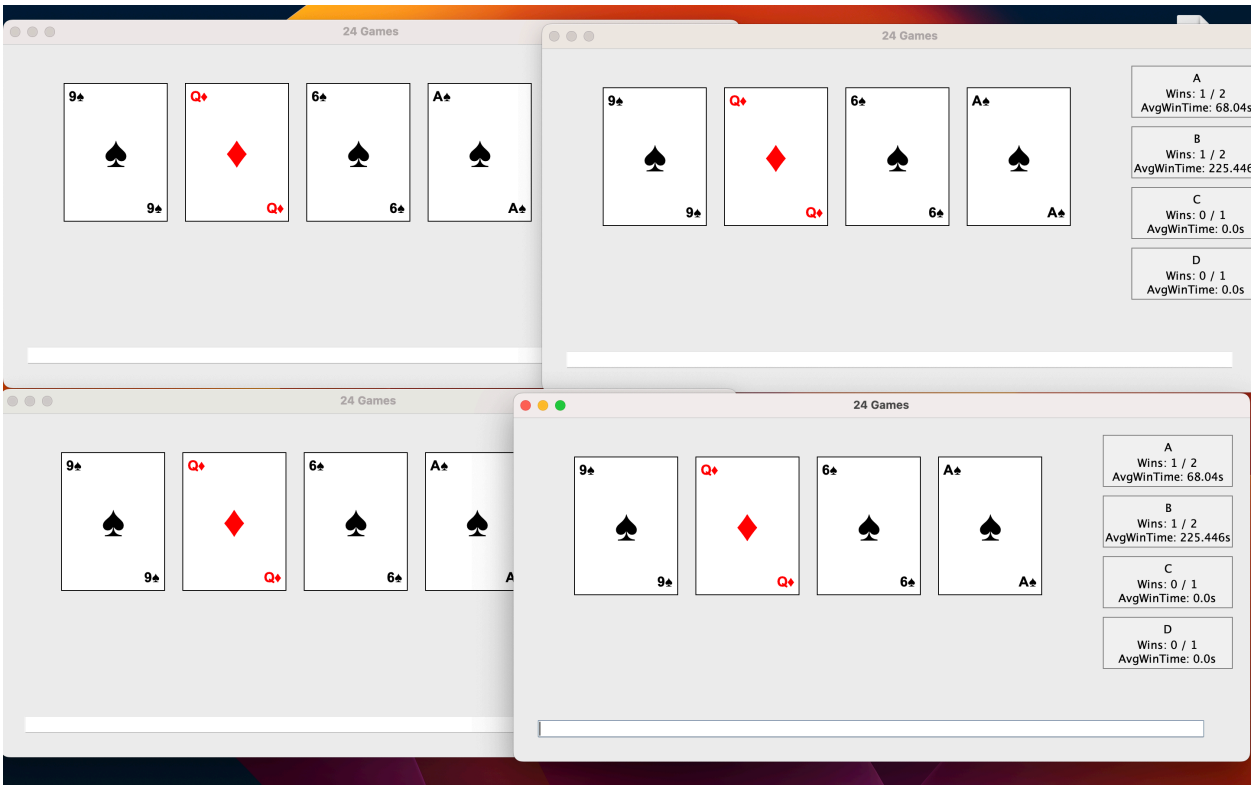


Fig 12. Game being played with all four players, starts immediately if all players present

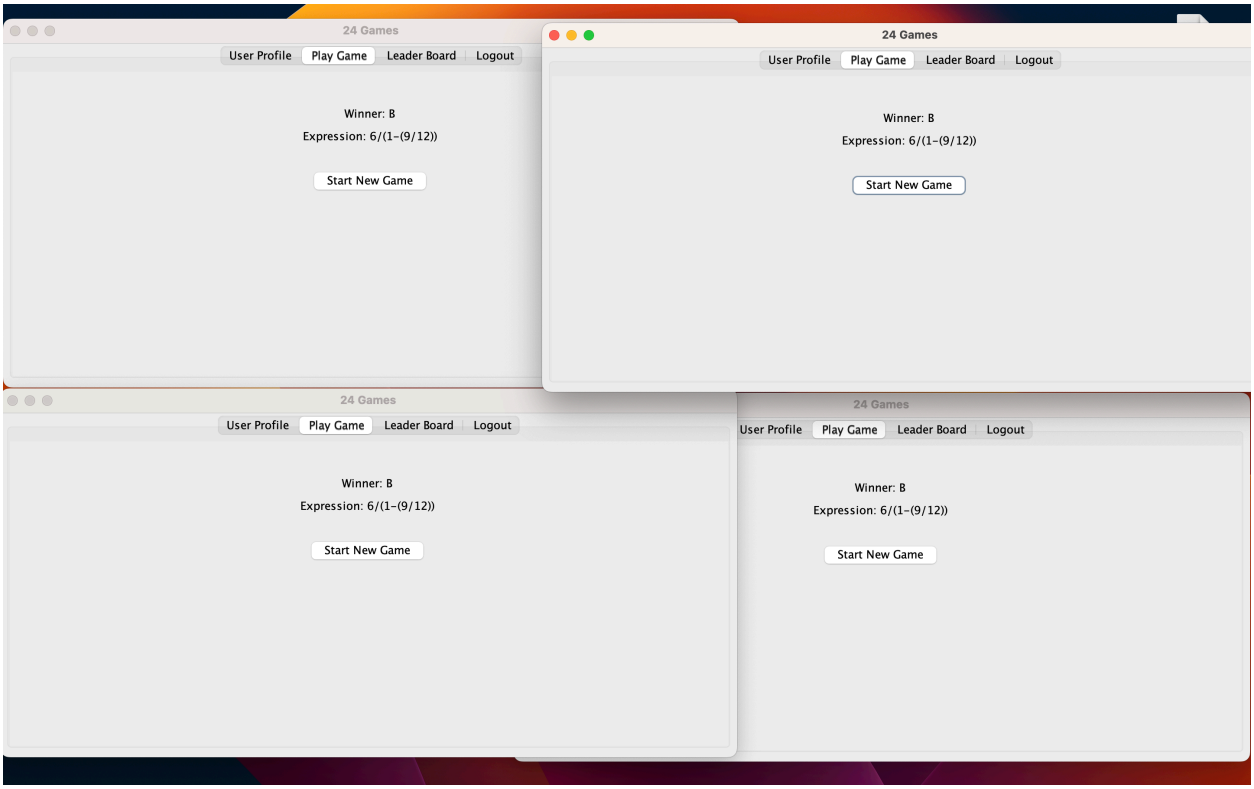


Fig 13. Once a valid expression is input, game ends and the result is broadcast to all players

If you investigate deeper into the code base itself you will see that some security design decisions have been made. For example, instead of directly storing the passwords a hash (SHA-256) is stored instead.

There are even more error prompts that have been omitted from this report for the sake of brevity (such as for server disconnection etc.).

The primary file of interest in the server sub-package is the `AuthenticationManager.java` file, which contains the business logic for the server.

Whereas, in the client sub-package, the `AppPanel.java` file contains the UI for the user after being logged in and `LoginManager.java` and `RegistrationManager.java` as the names imply handle login and registration logic.

All other files are either utility modules, define interfaces or enums. The features from the previous assignments carry over as well and can be tested. Feel free to reach out to me at shaheer@connect.hku.hk in case there are build issues.

ChatGPT was used to aid design the UI for this project but the code was stringently reviewed by the developer and all important logic, including the read-write lock in the server, was written by the developer— me :) —as well!