

**Escola Superior de Tecnologia e Gestão de Oliveira do Hospital**

**Licenciatura em Engenharia Informática**  
**Licenciatura em Sistemas e Tecnologias**  
**da Informação**

**Sistemas Distribuídos**

**Ano Letivo 2020/21**

**Projeto Final**

**Elaborado em: 2021/06/26**

**Francisco Mesquita, nº 2018056868**

**José Maurício, nº 2018056151**

**POLITÉCNICO**  
**DE COIMBRA**

## Índice

<b>Lista de Figuras.....</b>	<b>ii</b>
<b>1. Introdução .....</b>	<b>1</b>
<b>2. Projeto Final – Blackjack Final .....</b>	<b>2</b>
2.1. Descrição de mecanismos e protocolos de comunicação .....	2
2.2. Explicação de exclusão mútua .....	3
2.3. Manual de utilizador.....	4
<b>3. Conclusão.....</b>	<b>11</b>
3.1. Pontos Fortes.....	11
3.2. Limitações.....	11
3.3. Trabalho Futuro .....	11
<b>4. Referências .....</b>	<b>12</b>

## **Lista de Figuras**

FIGURA 2-1 - DIAGRAMA EXPLICATIVO SOBRE O FUNCIONAMENTO DO JAVA RMI .....	3
FIGURA 2-2 - INSERIR O PORTO DE COMUNICAÇÃO NO LADO DO SERVIDOR .....	4
FIGURA 2-3 - REGISTO DO CLIENTE NO SERVIDOR RMI.....	5
FIGURA 2-4 - INICIO DE UM NOVO JOGO DE BLACKJACK .....	6
FIGURA 2-5 - DISTRIBUIÇÃO DE CARTAS A PARTIR DO MOMENTO QUE COMEÇA UM NOVO JOGO .....	7
FIGURA 2-6 - REVELAÇÃO DAS CARTAS DO DEALER QUANDO TERMINA A RODADA.....	8
FIGURA 2-7 - ENTRADA DE UM OBSERVADOR NO JOGO DE BLACKJACK.....	9
FIGURA 2-8 - O OBSERVADOR ENQUANTO ESPERA POR A SUA VEZ DE JOGAR CONSEGUE OBSERVAR O JOGO DOS OUTROS JOGADORES.....	10

## **1. Introdução**

Ao longo da UC Sistemas Distribuídos abordamos vários mecanismos de comunicação em rede: Socket TCP e UDP, Java RMI e REST Web Services. Para realizarmos o projeto final desta UC foi escolhido um mecanismo dos que foram abordados para implementar um jogo de Blackjack online. Dessa forma escolhemos a tecnologia Java RMI para o desenvolvimento do mesmo projeto.

O RMI (Remote Method Invocation) é uma tecnologia suportada pela plataforma Java que facilita o desenvolvimento de aplicações distribuídas. Como o próprio nome indica, o RMI permite ao programador invocar métodos de objetos remotos, ou seja, que estão alojados em máquinas virtuais Java distintas, duma forma muito semelhante às invocações a objetos locais.

Ao longo deste relatório será explicado o mecanismo de comunicação em rede utilizado e a forma como foi implementado. Bem como, os mecanismos de exclusão mútua adotados para garantir que o projeto não sofre de race conditions. Por fim, demonstramos como é que o utilizador comum pode compilar, instalar e executar a aplicação.

## **2. Projeto Final – Blackjack Final**

Nesta secção será descrito os mecanismos e protocolos de comunicação utilizados para a implementação de um jogo de Blackjack online. Ou seja, de todos os mecanismos de comunicação estudados (e.g. Sockets TCP, Java RMI e REST Web Services) será apresentado o mecanismo que demonstrou ser mais eficiente na sua implementação. Ainda nesta secção será demonstrado como é que foram resolvidas todas as situações de exclusão mútua. E por fim, é apresentado o manual de utilização da aplicação.

### **2.1. Descrição de mecanismos e protocolos de comunicação**

Para a implementação de um jogo de Blackjack online tínhamos vários mecanismos disponíveis para o fazer. Contudo, o nosso objetivo foi escolher um mecanismo de comunicação que fosse eficiente e que fosse simples de implementar, isto, devido à falta de tempo para realizar o desenvolvimento do projeto.

Por isso, o mecanismo utilizado para a implementação do jogo de Blackjack online foi o Java RMI com o conceito de callback. Ou seja, a partir do momento que o utilizador se regista na aplicação, o servidor guarda as suas informações do jogo (i.e. a posição em que se encontra na mesa, o nickname e a indicação se é observador ou jogador), assim como a referência remota desse mesmo utilizador. Logo, cada vez que o jogo entrar numa situação de atualização de dados, o servidor vai procurar todas as referências dos jogadores e dos observadores registados e vai lhe enviar não só a sua informação atualizada como também a informação atualizada dos outros jogadores e do dealer. E esses dados quando chegam em bruto ao lado do cliente são tratados e apresentados no ecrã do utilizador para que o mesmo consiga interpretar o que está a acontecer durante o jogo.

Portanto, desta forma a manipulação dos dados ocorre toda do lado do servidor, isto é, a contagem de pontos do dealer e dos jogadores, a atribuição das cartas aos elementos presentes no jogo, assim como, a troca de jogadores por observadores. Deste modo garantimos que os vários clientes registados no servidor RMI não conseguem falsificar as informações do jogo com o objetivo de ganhar vantagem perante os seus adversários.

Na Figura 2-1 é observável um diagrama explicativo sobre como funciona o Java RMI.

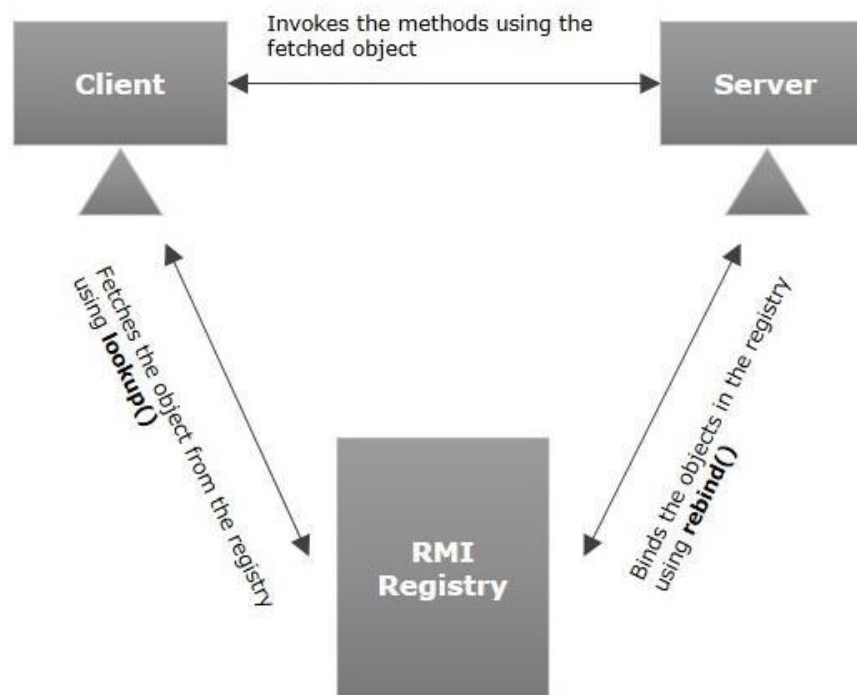


Figura 2-1 - Diagrama explicativo sobre o funcionamento do Java RMI

## 2.2. Explicação de exclusão mútua

O servidor RMI garante a exclusão mútua ao fazer a implementação do identificador `synchronized` na declaração dos métodos do objeto remoto como já tinha sido explicado no relatório das fichas práticas. Só desta forma que o servidor assegura que vários clientes acedem ao mesmo método de forma sincronizada. Isto é, só quando um cliente libertar um determinado método é que o cliente seguinte pode aceder ao mesmo método. Todavia, existirão problemas de *race conditions* no programa ao utilizar-se a estrutura de dados `HashMap` para armazenar os jogadores registados no jogo, que por sua vez não é sincronizada [1]. E nesse sentido quando estiver uma *thread* a iterar o objeto da classe `HashMap` para manipular os dados contidos dentro do mesmo e concorrentemente outra *thread* estiver a iterar o mesmo objeto para fazer a remoção de jogadores irá ser levantada uma exceção `ConcurrentModificationException`.

Para mitigar a ocorrência deste problema foi substituído a estrutura de armazenamento de dados dos jogadores e dos observadores por um `ConcurrentHashMap` que é uma cópia da estrutura `HashMap`. No entanto, a estrutura de dados `ConcurrentHashMap` é sincronizada [2], o que permite evitar quaisquer problemas de *race conditions* que existam. Isto, porque enquanto uma *thread* estiver a iterar a estrutura, outra *thread* fica em lista de espera para iterar o objeto da estrutura.

Depois ainda foram implementadas estruturas `CopyOnWriteArrayList`, em alternativa à estrutura `ArrayList`, para guardar as cartas dos jogadores e as cartas do dealer, isto, para evitar problemas

de race conditions ao manipular as estruturas de armazenamento das cartas. Como este tipo de estrutura é sincronizada vai permitir mitigar todas situações em que existam concorrência entre threads [3], [4]. Portanto, com a utilização da estrutura CopyOnWriteArrayList é possível termos múltiplas threads a manipular o objeto concorrentemente sem que ocorra problemas de concorrência.

### 2.3. Manual de utilizador

No momento de utilizar o jogo de Blackjack desenvolvido neste trabalho, necessita de ter acesso a dois programas: o programa desenvolvido para executar o servidor RMI e o programa que foi desenvolvido para executar o cliente RMI. Estes programas devem ser abertos com o IDE NetBeans 8.2 e para compilar os programas na mesma ferramenta deve garantir que tem instalado na sua máquina o Java 8.

A partir deste momento deve proceder primeiro à execução do programada destinado para o servidor RMI, visto que, o servidor vai ter que captar todos os pedidos de ligação enviados pelo cliente. Depois de o servidor RMI ser executado será pedido o porto em que se encontrará à escuta de novas ligações por parte do cliente RMI. Como mostra a Figura 2-2. No qual deve ser introduzido um valor inteiro com o número do porto.

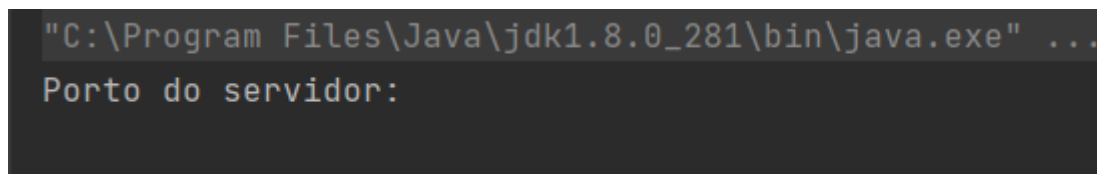
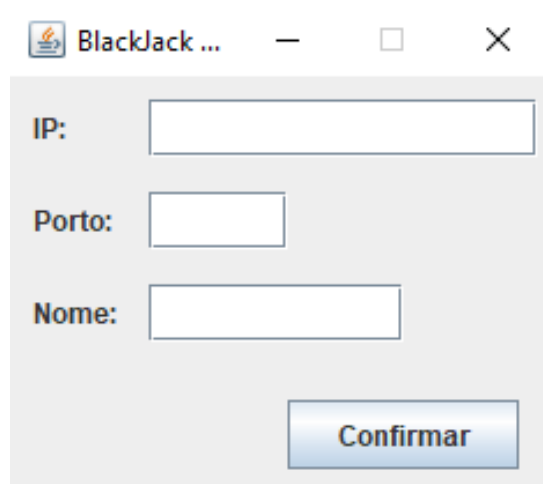


Figura 2-2 - Inserir o porto de comunicação no lado do servidor

No lado do cliente RMI, depois de ser executado o seu programa é aberto uma janela do Java Swing que vai solicitar o endereço de IP e o porto do servidor ao qual vai estabelecer uma ligação. Bem como, o nickname por o qual será identificado durante o jogo. Como mostra a Figura 2-3. Ao clicar no botão “Confirmar” será enviado um pedido de ligação ao servidor. E se o endereço de IP e o porto estiverem corretos vai permitir registar o jogador.



The image shows a Java Swing window titled "BlackJack ...". Inside the window, there are three text input fields arranged vertically. The first field is labeled "IP:", the second is labeled "Porto:", and the third is labeled "Nome:". Below these fields is a blue button with the text "Confirmar". The window has standard OS window controls (minimize, maximize, close) in the title bar.

Figura 2-3 - Registo do cliente no servidor RMI

Na Figura 2-4 está demonstrado a mesa do jogo de Blackjack com todos os jogadores que se registaram no servidor RMI e o dealer. A mesma figura ilustra o início de um novo jogo. Em que ainda não foram atribuídas cartas aos jogadores e os botões de Hit e Stand encontram-se desativados. No lado direito da Figura 2-4 pode-se observar dois tipos de informação. Pode-se observar uma tabela com o nickname de todos os observadores que se encontram em espera para jogar. E pode-se observar uma TextArea com todas as mensagens que estão a ser enviados por o servidor, acompanhadas pela hora em que o servidor as enviou.



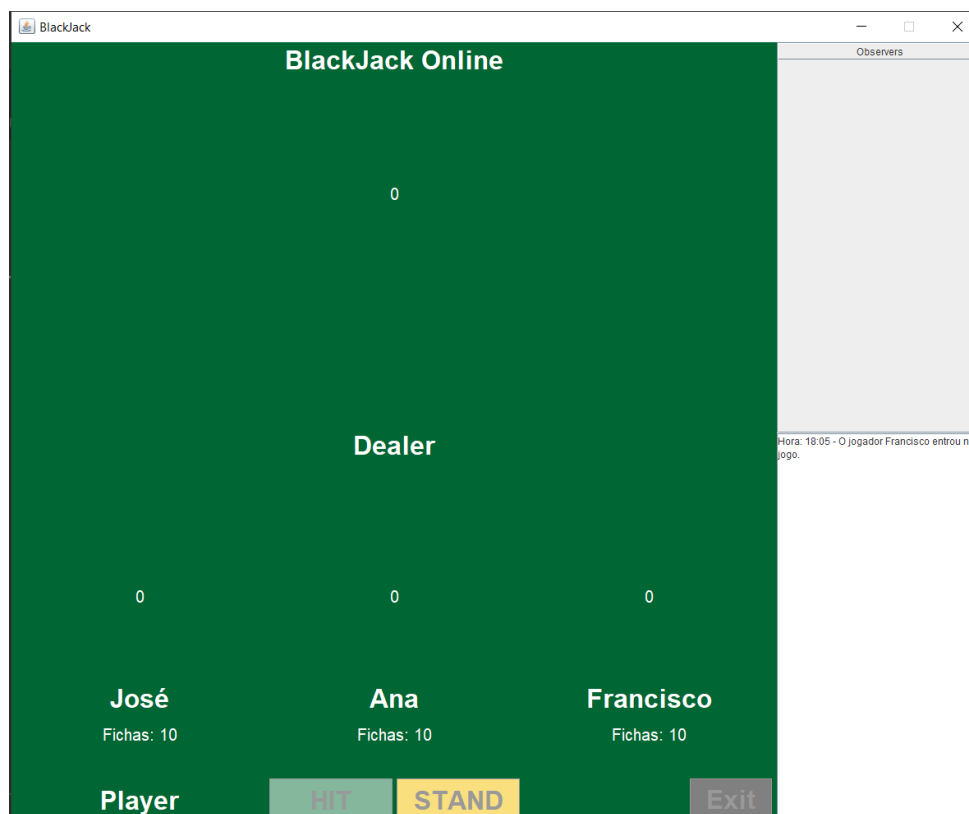


Figura 2-4 - Início de um novo jogo de Blackjack

Antes de serem distribuídas novas cartas a todos os jogadores e ao dealer são aguardados 10 segundos. Isto, para permitir que todos os jogadores se registem a tempo de começarem um novo jogo. Passados 10 segundos são distribuídas 2 cartas por cada elemento presente na mesa de jogo. E é feita a atualização da informação que o jogador observa na sua janela da aplicação. Como mostra a Figura 2-5. O jogo segue as regras do Blackjack normal. Em que o primeiro jogador a jogar é o que se encontra mais à direita e depois evolui no sentido dos ponteiros do relógio. Quando chegar ao jogador mais à esquerda e o mesmo terminar a sua jogada é encerrada aquela ronda. Os botões Hit e Stand serão ativos apenas no momento que chegar a vez de um determinado jogador jogar e inativos quando passar a vez de jogar do mesmo. O botão Hit serve para o jogador pedir novas cartas ao servidor RMI e o botão Stand serve para passar a vez para o jogador seguinte.

As cartas do dealer são reveladas e é determinado o vencedor da ronda. Neste momento o botão “Exit” fica ativo e o observador pode abandonar a mesa de jogo e dar lugar ao observador que se encontra há mais tempo à espera. O programa volta a aguardar mais 10 segundos e depois desse tempo passar é novamente atualizada a janela dos jogadores e observadores com os novos dados de uma nova ronda.

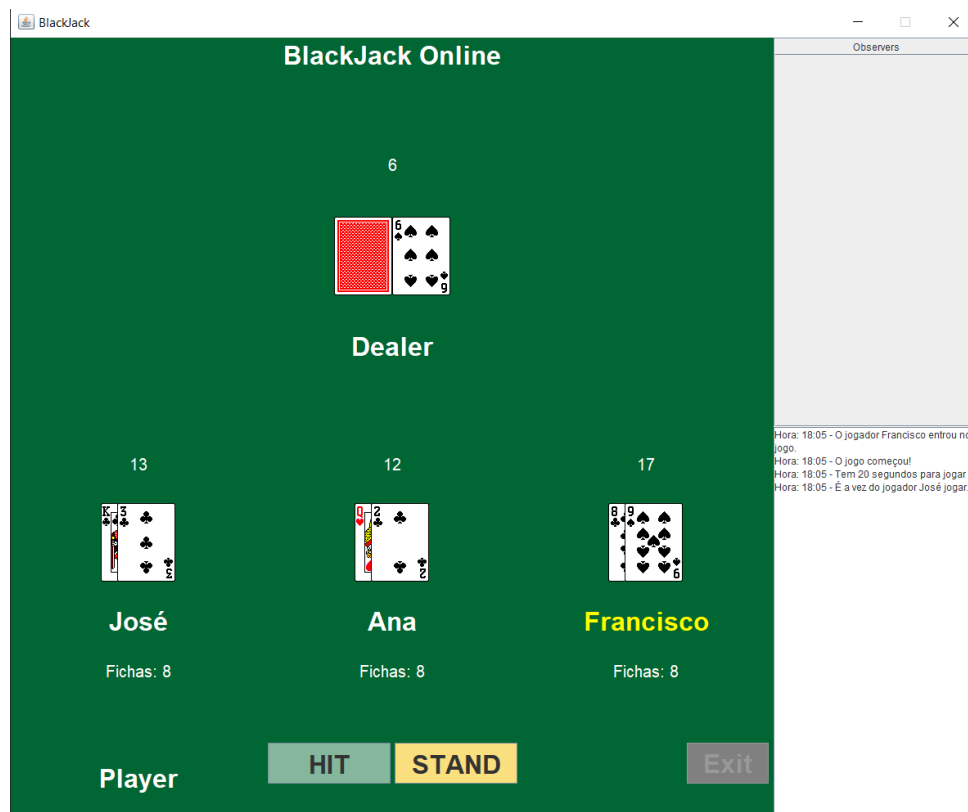


Figura 2-5 - Distribuição de cartas a partir do momento que começa um novo jogo

Na Figura 2-6 é possível observar-se qual será o resultado visível quando uma ronda terminar. A partir deste momento o servidor remove as cartas da mesa de jogo e fica a aguardar 10 segundos por o início de um novo jogo. Atribuindo novas cartas ao dealer e aos jogadores.

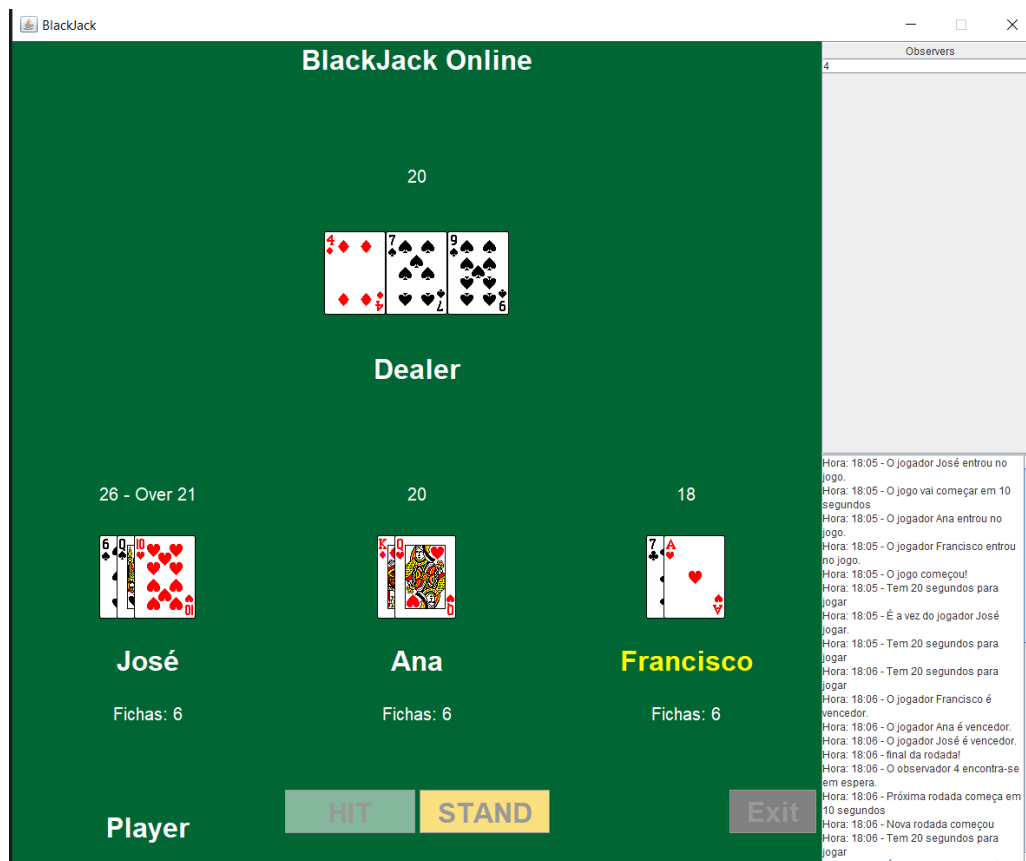


Figura 2-6 - Revelação das cartas do dealer quando termina a rodada

A Figura 2-7 ilustra o início de uma nova ronda do jogo. E pode-se observar que as cartas foram removidas da mesa e que o número de fichas que os jogadores possuem foi atualizado. Na mesma figura é visível ainda o nickname de um observador na tabela de observadores, que representa a entrada de um observador no jogo.

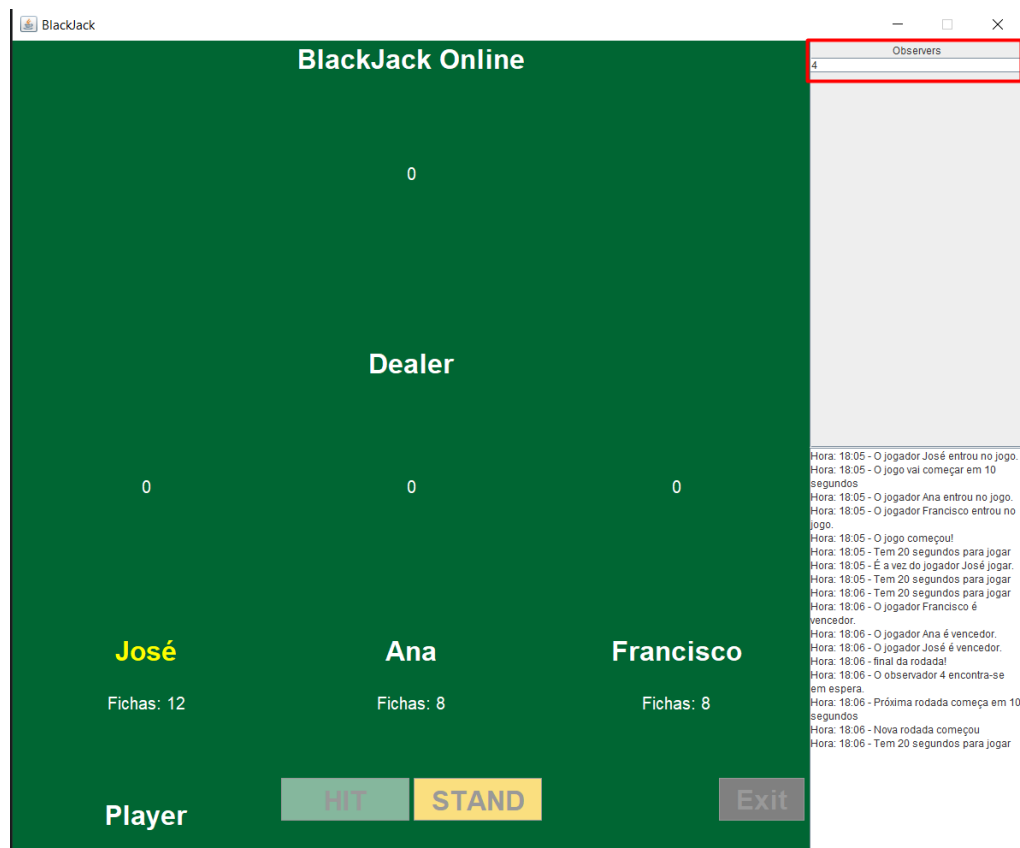


Figura 2-7 - Entrada de um observador no jogo de Blackjack

Após começar uma nova ronda no jogo de Blackjack, o observador vai poder acompanhar toda a evolução do jogo. No entanto, está impedido de participar no jogo que está a decorrer, visto que, os botões Hit e Stand se encontram inativos. Como é mostrado na Figura 2-8.

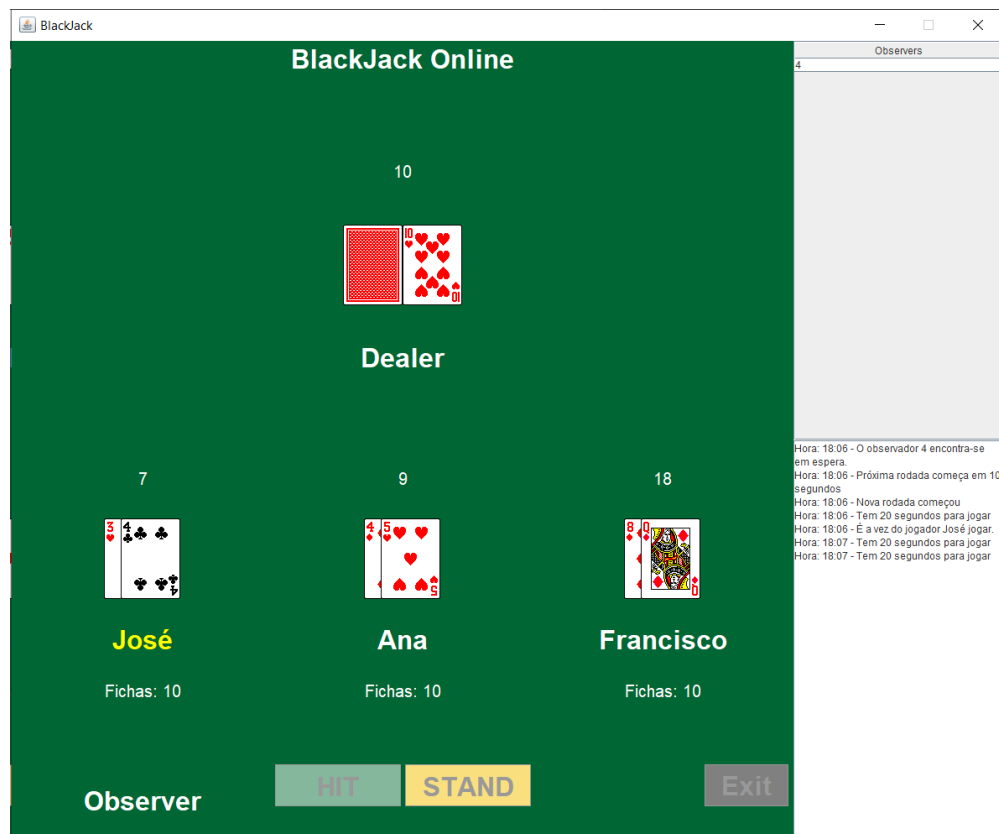


Figura 2-8 - O observador enquanto espera por a sua vez de jogar consegue observar o jogo dos outros jogadores

### **3. Conclusão**

Java RMI traz um nível de funcionalidade a programas distribuídos com características como gestão distribuída e automática de objetos e passagem dos próprios objetos de máquina para máquina através da rede.

Terminado o desenvolvimento do projeto é o momento de retirarmos conclusões de todos os procedimentos adotados. Desse modo, reconhecemos que o projeto sofreu alguns desvios em relação ao que tínhamos planeado, isto, devido à excessiva carga horária que tínhamos no momento do desenvolvimento deste projeto. Contudo, o trabalho destaca-se pela implementação do conceito do jogo de Blackjack e por a implementação de todos os requisitos essenciais ao funcionamento da aplicação

Em suma, o trabalho foi realizado com sucesso, sendo este essencial para a melhor compreensão teórica e prática tanto de Java RMI assim como de métodos de exclusão mútua.

#### **3.1. Pontos Fortes**

Como pontos fortes deste trabalho destaca-se todas as funcionalidades principais implementadas. Assim como, uma interface simples e agradável de fácil uso por o utilizador comum.

#### **3.2. Limitações**

A aplicação apresenta pequenas falhas quando o jogador não possui fichas suficientes para continuar a apostar e nesse momento não existe um observador presente no jogo para entrar na sua vez. Para além disso, apresenta alguns pequenos problemas em funcionalidades não principais, tendo estas sido detetadas perto da entrega e a carga de trabalho não permitir a sua resolução.

#### **3.3. Trabalho Futuro**

Num trabalho futuro procuramos resolver todas as limitações detetadas. Assim como, procuramos melhorar o código fonte de modo a otimizar a aplicação.

## **4. Referências**

- [1] «HashMap (Java Platform SE 8 )». <https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html> (acedido Jun. 26, 2021).
- [2] «ConcurrentHashMap (Java Platform SE 8 )». <https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html> (acedido Jun. 26, 2021).
- [3] «CopyOnWriteArrayList (Java Platform SE 8 )». [https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CopyOnWriteArrayList.html#toArray\(T\[\]\)](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CopyOnWriteArrayList.html#toArray(T[])) (acedido Mai. 02, 2021).
- [4] «CopyOnWriteArrayList in Java», *GeeksforGeeks*, Jan. 09, 2018. <https://www.geeksforgeeks.org/copyonwritearraylist-in-java/> (acedido Mai. 02, 2021).