Simulador Heads-Up Texas Hold'em

Filipe Bicho
Licenciado em Informática, Univ. Aberta, mail@filipebicho.com

Paulo Shirley
Univ. Aberta, INESC TEC, Paulo.Shirley@uab.pt

Resumo

O poker é um jogo de cartas usualmente jogado em casinos e é considerado o jogo de cartas mais popular do mundo. Texas Hold'em é o estilo mais jogado em que cada jogador recebe duas cartas privadas e cinco cartas são dispostas à mesa, de forma a serem compartilhadas por todos os participantes do jogo. Quando é jogado com apenas dois jogadores, é designado por Heads-up Texas Hold'em, sendo um jogo muito interessante e extremamente competitivo de jogar. Este projeto tem por objetivo criar um simulador de Heads-up Texas Hold'em em que existem dois modos de jogo, utilizador contra computador e computador vs computador, proporcionando uma ferramenta que permite aos utilizadores treinarem e melhorarem o seu desempenho neste tipo de jogo, assim como explorarem variantes de algoritmos que simulam um jogador de poker. A implementação deste projeto contemplou dois grandes desafios: planear uma arquitetura de software para a implementação do jogo e desenvolver algoritmos para simular um jogador.

Palavras-chave: jogo de cartas, poker, texas hold'em, algoritmo jogador, arquitetura jogo

Title: Texas Hold'em Heads-Up Simulator

Abstract

Poker is a card game usually played in casinos and is considered the most popular card game in the world. Texas Hold'em is the most played style in which each player receives two private cards and five cards are arranged at the table in order to be shared by all participants of the game. When played with only two players, it is called Heads-up Texas Hold'em, being a very interesting and extremely competitive game to play. This project aims to create a Texas Hold'em heads-up simulator in which there are two modes of play, user against computer and computer vs. computer, providing a tool that allows users to practice and improve their performance in this type of game, as well as exploring variants of algorithms that simulate a poker player. The implementation of this project contemplated two major challenges: to plan a software architecture for the implementation of the game and to develop algorithms to simulate a player.

Keywords: card game, poker, texas hold'em, poker bot, game architecture

1. Introdução

O Texas Hold'em poker foi escolhido para este projeto porque é um jogo muito interessante de jogar e ainda mais desafiante de criar. Para a grande maioria dos jogadores que jogam poker por entretenimento, é possível observar no grupo de pessoas com que jogam um conjunto de características, nomeadamente o desprezo pelas probabilidades e qualquer tipo de abordagem estratégica, que espelham a imagem de que, para muitas pessoas, o poker é um jogo de sorte em que as decisões acertadas têm a ver mais com o resultado final, positivo ou negativo, e não com o facto de, à altura da decisão, esta ter sido a mais correta relativamente à probabilidade de ganhar a mão. Este projeto tem como objetivo criar vários simuladores de jogador de poker em que as decisões tomadas são baseadas em probabilidades e em quanto é necessário pagar para disputar o pote.

O poker é um jogo de cartas usualmente jogado em casinos e é considerado o jogo de cartas mais popular do mundo. É jogado entre duas a noves pessoas e pertence a uma classe de jogos nos quais os jogadores com as cartas total ou parcialmente escondidas fazem apostas para um pote central, no qual é atribuído ao jogador ou jogadores que possuir(em) o melhor conjunto de cartas entre os que permaneceram em jogo, ou ao jogador restante, caso os outros tenham desistido da aposta.

Texas Hold'em é o estilo de jogo mais popular no mundo do poker em que cada jogador recebe duas cartas privadas e cinco cartas são dispostas à mesa, de forma a serem compartilhadas por todos os participantes do jogo.

Este projeto baseia-se em Heads-up Texas Hold'em que é uma forma de poker jogada entre dois jogadores, as regras nesta forma de jogo são as mesmas utilizadas para o Texas Hold'em jogado com mais jogadores, exceto nas blinds, em que o jogador que está na posição de dealer é a small blind e o outro jogador é a big blind.

O objetivo deste projeto é o desenvolvimento na linguagem Java de um simulador Heads-up Texas Hold'em em modo texto e modo gráfico 2D. O simulador criado será constituído por dois modos:

- Utilizador vs Computador: neste modo o utilizador realiza um jogo contra o computador até ser encontrado um vencedor que ganhe o dinheiro do oponente, o utilizador pode escolher o tipo de adversário, ou seja, o computador pode ter vários estilos de jogo:
    - Certo: o computador apenas efetua apostas quando as probabilidades de ganhar a jogada lhe são favoráveis ou quando a aposta do adversário é relativamente baixa
    - Agressivo: o computador pode efetuar apostas altas mesmo com probabilidades baixas de ganhar a jogada e igualar apostas altas do adversário á espera de uma carta.
    - Misto: o estilo de jogo do computador pode alterar entre agressivo e certo consoante o momento do jogo.

    - Fórmula: o computador escolhe a efetua apostas baseado no resultado da fórmula Mathemattically Fair Strategy (MFS).
- Computador vs Computador: neste modo o computador efetua um jogo contra si mesmo, ou seja, o algoritmo desenvolvido irá se defrontar num estilo de jogo previamente escolhido, não necessariamente o mesmo estilo de jogo para ambos os algoritmos.

Para além da simulação do heads up Texas Hold'em o utilizador poderá fazer um login com a sua conta e assim ter acesso às suas estatísticas das sessões anteriores, tais como:

- Mãos mais jogadas. (as duas cartas do utilizador)
- Mãos de poker que ganharam mais jogos. (as duas cartas do utilizador)
- A mão de poker mais alta. (as cinco cartas do utilizador)
- Total de jogadas.
- Número de jogadas ganhas.
- Percentagem de vitórias.

2. Como jogar *Texas Hold'em Poker*

O Texas Hold'em é o estilo de jogo mais popular do *community card poker*, e o seu grande objetivo é ganhar o dinheiro dos oponentes. Todos os jogadores apostam dinheiro para um pote central e o jogador que tiver o melhor jogo no final recebe esse dinheiro. Para começar, cada jogador recebe duas cartas que somente o dono das cartas pode visualizar. A *small blind* e a *blind* (também por vezes designada por *Big blind*) são apostas mínimas obrigatórias a efetuar pelos jogadores no início do jogo. Um jogo é dividido em quatro rondas em que podem ocorrer apostas:

- Primeira ronda – *Pré-flop*: o primeiro a jogar é a *small blind* que significa que é obrigado a apostar metade do valor da *blind*. No caso de *heads up* a *small blind* também é o *dealer* enquanto o oponente paga a *big blind* podendo ainda aumentar a aposta.
- Segunda ronda – *Flop*: a segunda ronda começa com três cartas visíveis aos jogadores, e as apostas começam com o jogador que não é o *dealer*, nesta ronda de aposta não é obrigatório apostar dinheiro, o jogador pode apenas fazer *check* para continuar em jogo.
- Terceira ronda – *Turn*: na terceira ronda é mostrada uma carta aos jogadores, as apostas são iguais às apostas no flop.
- A quarta ronda – *River*: na ultima ronda é mostrada a ultima carta aos jogadores, sendo as apostas iguais às duas rondas anteriores.

No Texas Hold'em Poker existem os seguintes modos de apostar:

- *Fold* (desistir): significa que o jogador desistiu da jogada e o oponente leva o dinheiro presente no pote.
- *Call* (pagar): para continuar em jogo o jogador tem que apostar o mesmo que o oponente apostou.

- *Check* (passar): se o jogador não quiser apostar e se ainda não existiu nenhuma aposta então o jogador pode fazer *check*.
- *Bet* (apostar): quando ainda não existiu uma aposta o jogador pode efetuar uma aposta entre o valor de duas *blinds* e as suas fichas.
- *Raise* (aumentar): quando o oponente fez uma aposta (*bet*) o jogador paga a aposta do oponente e aumenta a aposta.
- *Allin* (apostar tudo): quando o jogador aposta o seu dinheiro todo.

Definem-se as seguintes posições dos jogadores na mesa:

- ***Dealer***: no inicio de cada jogo é escolhido um *dealer*, que é o ultimo a fazer uma aposta, mas como em *heads up* existem apenas dois jogadores o *dealer* também é a *small blind*.
- ***Small blind***: paga metade do valor da *blind*, ou seja, se o valor da *blind* for 20 fichas então a *small blind* tem de pagar 10 fichas. Em *heads up* quem está na posição da *small blind* é o primeiro a apostar.
- ***Big blind***: é obrigado a pagar o valor da *blind*, ou seja, se a valor da *blind* for 20 fichas então têm de pagar 20 fichas. Em *heads up* quem está na posição da *big blind* é o ultimo a apostar.

No final das quatro rondas o pote é ganho pelo jogador que possuir a "melhor mão", ou seja, o conjunto de cinco cartas obtido a partir de quaisquer das suas duas cartas privadas e das cinco cartas comunitárias da mesa, com o maior ranking. No poker definem-se os seguintes rankings:

- ***Royal flush*** (Sequência máxima de cor): consiste em ás, rei, dama, valete e dez do mesmo naipe. Esta é a mão mais forte do *poker*.
- ***Straight fush*** (Sequência de cor): cinco cartas em sequência do mesmo naipe.
- ***Four of a kind*** (*poker*): quatro cartas do mesmo valor e uma carta de desempate.
- ***Full house***: três cartas do mesmo valor (trio), e outras duas do mesmo valor(par).
- ***Flush***(Cor): cinco cartas do mesmo naipe que não estejam em sequência.
- ***Straight***(Sequência): cinco cartas de naipes diferentes em sequência.
- ***Three of a kind***(Trio): três cartas do mesmo valor e duas cartas laterais não relacionadas.
- ***Two pairs*** (Dois pares): duas cartas de valor idêntico, outras duas cartas de outro valor idêntico entre si e uma carta de desempate
- ***Pair***(Par): duas cartas de valor idêntico e três cartas laterais não relacionadas.
- ***High card*** (Carta alta): qualquer mão que não se qualifique numa categoria listada acima.

3. Arquitetura do Software

Este projeto foi desenvolvido em linguagem Java e encontra-se estruturado por classes. Foram criadas classes para guardar cartas, baralhar as cartas, distribuir as cartas pelos dois jogadores, avaliar a mão de cada jogador, calcular qual a melhor mão entre os dois jogadores, calcular probabilidades de cada mão de ganhar a jogada ou melhorar na próxima

ronda, implementar algoritmos que simulem um jogador de *poker*, guardar as estatísticas do utilizador e por fim implementar uma interface gráfica. Nas subseções seguintes são resumidas as principais propriedades das classes desenvolvidas.

3.1. Classe Cartas

O objetivo desta classe é definir uma estrutura de dados para guardar uma carta e criar métodos para a aceder. Uma carta é definida por dois atributos, carta e naipe, que correspondem ao número e naipe da carta.

Para aceder às cartas foram definidos dois métodos, "getNumCarta()" e "getNaipe()", o primeiro retorna o número da carta, como por exemplo, às retorna o número 0 e rei retorna o número 12, o segundo método retorna o naipe da carta. Foi também definido o método "toString()" que gera uma string apropriada para impressão com a representação do número da carta seguido do símbolo do naipe. A figura 1 mostra a impressão de todas as cartas possíveis.

```
A♥ 2♥ 3♥ 4♥ 5♥ 6♥ 7♥ 8♥ 9♥ 10♥ J♥ Q♥ K♥
A♦ 2♦ 3♦ 4♦ 5♦ 6♦ 7♦ 8♦ 9♦ 10♦ J♦ Q♦ K♦
A♣ 2♣ 3♣ 4♣ 5♣ 6♣ 7♣ 8♣ 9♣ 10♣ J♣ Q♣ K♣
A♠ 2♠ 3♠ 4♠ 5♠ 6♠ 7♠ 8♠ 9♠ 10♠ J♠ Q♠ K♠
```

Figura 1 - Cartas.java

3.2. Classe Baralho

O objetivo desta classe é criar e baralhar 52 cartas como definidas na classe "Cartas". Para isso foi criado um "ArrayList" do tipo "Cartas" que vai guardar as cartas todas juntas e que depois são baralhadas através do algoritmo de *Fisher–Yates* descrito na figura 2.

```
-- To shuffle an array a of n elements (indices 0..n-1):
for i from n−1 downto 1 do
     j ← random integer such that 0 ≤ j ≤ i
     exchange a[j] and a[i]
```

Figura 2 – Algoritmo de Fisher-Yates

O algoritmo consiste num ciclo *for* que vai desde o tamanho do *array* até 0, e a cada iteração é gerado um número aleatório entre 0 e o número da iteração, após gerado o número aleatório é realizado a troca de cartas. Nesta classe foram criados dois métodos, o primeiro método retira a carta que está no topo do baralho, "tirarCarta()", e o segundo método retorna o número de cartas presentes no baralho, "totalCartas()". A figura 3 mostra uma instância da classe "Baralho".

```
6♦ A♦ J♦ Q♦ 6♣ Q♣ 8♣ A♣ A♥ 10♥ Q♥ 2♠ K♠
4♥ 9♦ 4♣ J♣ 4♠ 3♦ 10♦ K♣ 3♥ 10♠ 5♣ 3♣ 5♠
10♣ 9♠ 8♦ K♥ A♠ Q♠ 3♠ 7♥ 5♥ 8♥ 2♥ 9♣ 4♦
2♣ K♦ 7♠ J♥ 5♦ J♠ 6♠ 6♥ 7♦ 8♠ 7♣ 9♥ 2♦
```

Figura 3 - Baralho.java

3.3. Classe Dealer

Esta classe tem o propósito de distribuir as cartas pelos jogadores e pela mesa, a distribuição tem de seguir uma certa ordem. Como o jogo desenvolvido é apenas de dois jogadores, o primeiro jogador a receber uma carta é a *big blind*, de seguida o *dealer* recebe a sua primeira carta e a segunda carta segue a mesma ordem de distribuição. Após os jogadores receberem as suas cartas, queima-se uma carta, ou seja, tira-se uma carta do baralho sem mostrar e mostra-se as 3 cartas do *flop*, volta-se a queimar uma carta e mostra-se a carta do *turn*, queima-se a ultima carta e por último mostra-se a carta do *river*. Todo o processo é representado na figura 4.

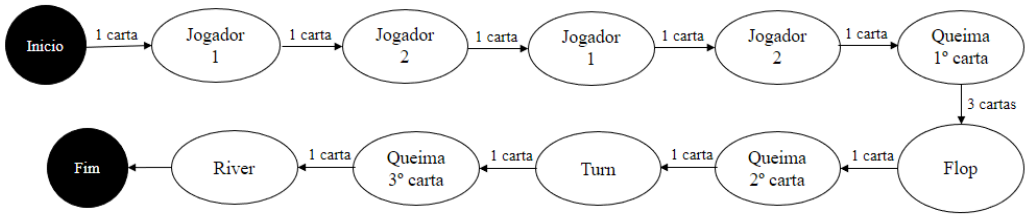


Figura 4 - Grafo distribuição de cartas

Para distribuir as cartas foram criados quatro métodos. O primeiro método "darCartas()" recebe como argumentos o baralho e os dois jogadores e distribui as cartas do baralho através do método "tirarCarta()" pelos jogadores.

Para mostrar as cartas na mesa existem três métodos que são chamados sempre pela mesma ordem, o primeiro método a ser chamado é "darFlop()", este método recebe como argumentos o baralho e as cartas da mesa, e começa por queimar uma carta, após a carta estar queimada atribui as 3 cartas seguintes à mesa. O método seguinte é "darTurn()" que segue a mesma lógica que o método anterior com a diferença de apenas atribuir uma carta à mesa. Por fim, vem o método "darRiver()" que é semelhante ao método anterior atribuindo mais uma carta, a última, à mesa.

```
Jogador 1: K♠ 4♦
Jogador 2: 10♣ 10♠
Flop: 4♥ 8♥ Q♣
Turn: 5♦
River: 9♠
```

Figura 5 - Dealer.java

3.4. Classe AvaliarMao

O grande objetivo desta classe é receber as cartas de um jogador e as cartas da mesa, e assim retornar o melhor *ranking* em que essa mão se encontra, conforme descrito na seção 2. Esta classe é mais complexa que as classes anteriores, sendo necessário criar vários métodos que auxiliem o cálculo do *ranking* de uma determinada mão. O desenvolvimento desta classe começa no método "avaliar()" que recebe como argumento as cartas do jogador, as cartas da mesa, e o número de cartas a avaliar, dado que a mesa pode conter 3, 4 ou 5 cartas conforme estivermos no *flop*, *turn* ou *river*.

O primeiro passo para avaliar o *ranking* das cartas recebidas em argumento é agrupá-las obtendo-se um conjunto de 5, 6 ou 7 cartas. O segundo passo após as cartas estarem todas reunidas é ordenar as cartas pelo seu número. O terceiro passo é o mais importante porque verifica o *ranking* em que o conjunto de cartas se enquadra. Cada *ranking* é analisado individualmente, começando pelo maior até ao menor conforme ilustrado na figura 6, através de um método dedicado a cada *ranking*: "isRoyalStraightFlush()", "isStraightFlush()", "is4ofAKind()", "isFullHouse()", "isFlush()", "isStraight()", "is3ofAKind()", "is2Pairs()", "is1Pair()" e por fim "isHighCard()". Estes métodos recebem como argumento o conjunto de cartas a analisar. O método "avaliar()" retorna um número correspondente ao *ranking* determinado, como por exemplo, *royal straight flush* retorna o número 10 enquanto carta alta retorna o número 1.

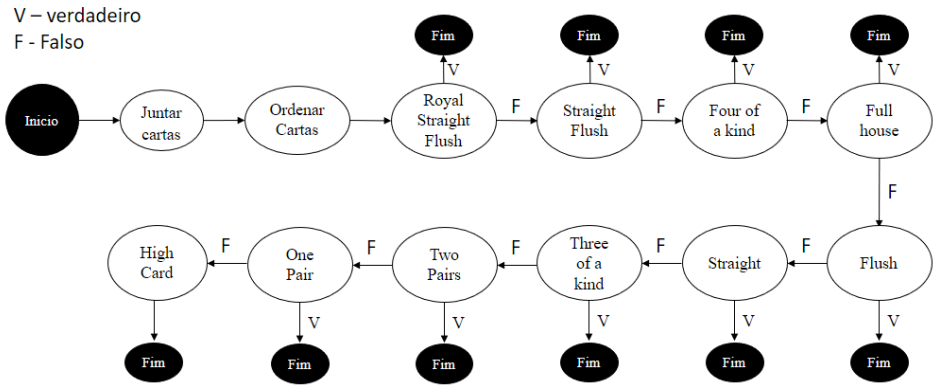


Figura 6 - Grafo Ranking de cartas

Existem outros dois métodos na classe, "obterMaoFinal()" que retorna o conjunto de cinco cartas correspondente ao ranking determinado e "mostrarClassificacao()" que retorna numa *string* apropriada para impressão a descrição textual do *ranking* de uma determinada mão, conforme exemplifcado na figura 7.

3.5. Classe CalcularVencedor

Esta classe tem como objetivo calcular qual o vencedor entre os dois jogadores. Através do método central "calcularVencedor()" que recebe como argumentos a mão do jogador 1, a mão do jogador 2 e seus respetivos *rankings*, calcula qual dos dois jogadores é o vencedor.

Para calcular o vencedor é necessário apenas comparar os *rankings* de ambos os jogadores, se o *ranking* do jogador 1 for maior ganha o jogador 1, se for menor ganha o jogador 2, mas se ambos tiverem o mesmo resultado tem que haver desempate e para isso é necessário comparar as cartas de ambos os jogadores. Para a comparação foi utilizado um método para cada *ranking* exceto para *Royal Straight Flush* dado que apenas pode existir um jogador com o *ranking* mais alto. A figura 8 exemplifica o resultado da comparação das mãos dos jogadores.

```
10♥ J♥ Q♥ K♥ A♥  - Royal Straigth Flush
9♥ 10♥ J♥ Q♥ K♥  - Straigth Flush
A♣ A♥ A♦ A♠ K♥   - Four of a kind
A♦ A♥ A♣ K♥ K♠   - Full House
3♥ 8♥ 10♥ Q♥ K♥  - Flush
6♥ 7♠ 8♦ 9♥ 10♥  - Straight
A♦ A♥ A♣ J♥ 10♥  - Three of a kind
A♦ A♥ J♣ J♥ 10♥  - 2 pares
A♦ A♥ J♥ 10♥ 2♣  - 1 par
2♦ 4♣ 10♥ J♥ A♥  - Carta alta
```

Figura 7 – AvaliarMao.java

```
Jogador 1: 10♥ J♥ Q♥ K♥ A♥   - Royal Straigth Flush
Jogador 2: 5♠ 6♥ 10♥ J♥ Q♥  - Carta alta
Ganha o jogador 1

Jogador 1: 9♥ 10♥ J♥ Q♥ K♥   - Straigth Flush
Jogador 2: 8♥ 9♥ 10♥ J♥ Q♥  - Straigth Flush
Ganha o jogador 1

Jogador 1: A♣ A♠ A♦ A♥ K♠   - Four of a kind
Jogador 2: K♠ K♥ K♦ K♣ A♣  - Four of a kind
Ganha o jogador 1

Jogador 1: A♥ A♦ A♣ 2♠ 2♣   - Full House
Jogador 2: K♦ K♥ K♣ 2♠ 2♣  - Full House
Ganha o jogador 1

Jogador 1: 2♥ 8♥ 10♥ Q♥ A♥   - Flush
Jogador 2: 2♥ 8♥ 10♥ Q♥ K♥  - Flush
Ganha o jogador 1

Jogador 1: 4♦ 5♥ 6♥ 7♣ 8♠   - Straight
Jogador 2: 5♣ 6♥ 7♣ 8♠ 9♦  - Straight
Ganha o jogador 2

Jogador 1: A♣ A♦ A♥ K♠ 10♥   - Three of a kind
Jogador 2: K♥ K♦ K♠ A♣ 10♥  - Three of a kind
Ganha o jogador 1

Jogador 1: A♥ A♣ 10♦ 10♠ K♣   - 2 pares
Jogador 2: K♣ K♥ 10♠ 10♣ A♣  - 2 pares
Ganha o jogador 1

Jogador 1: A♥ A♣ K♣ 10♥ 7♦   - 1 par
Jogador 2: K♣ K♥ A♣ 10♥ 8♦  - 1 par
Ganha o jogador 1

Jogador 1: 6♣ 8♠ 10♥ Q♣ K♠   - Carta alta
Jogador 2: 6♣ 8♠ 10♥ Q♣ K♠  - Carta alta
Empate
```

Figura 8 - CalcularVencedor.java

3.6. Classe Apostas

O objetivo desta classe é criar um sistema de apostas de acordo com as apostas oficiais do Texas Hold'em. Como descrito na secção "Como jogar Texas Hold'em Poker" existe várias opções de apostas que o utilizador pode realizar no *poker*, tais como, desistir, passar, apostar, aumentar e apostar tudo, também é necessário ter atenção às *blinds* obrigatórias que os jogadores têm de pagar, a *small blind* e a *big blind*. Para este projeto ambos os jogadores começam com 1500€, a *small blind* tem o valor de 10€ e a *big blind* 20€. Esta classe tem várias variáveis globais, das quais se destacam:

- "apostaInicial": é uma variável *boolean* inicializada a *true*, esta variável é apenas usada nas apostas pré-*flop* envolvendo a *small blind* e a *big blind* e vai-se tornar *false* quando a jogada inicial tiver terminado.
- "checkInicial" é uma variável *boolean* inicializada a *false*, esta variável é usada nas apostas no *flop*, no *turn* e no *river* e vai-se tornar *true* quando o primeiro jogador fizer a sua aposta.

Esta classe é constituída por dois métodos centrais, o primeiro método "apostasPreFlop()" é utilizado nas apostas antes do *flop* e o segundo método "apostas()" é utilizado nas rondas após o *flop* inclusive. Ambos os métodos orientam os jogadores para um conjunto de menus de opções de aposta conforme o estado do jogo em que se encontram, seguindo o diagrama da figura 9.

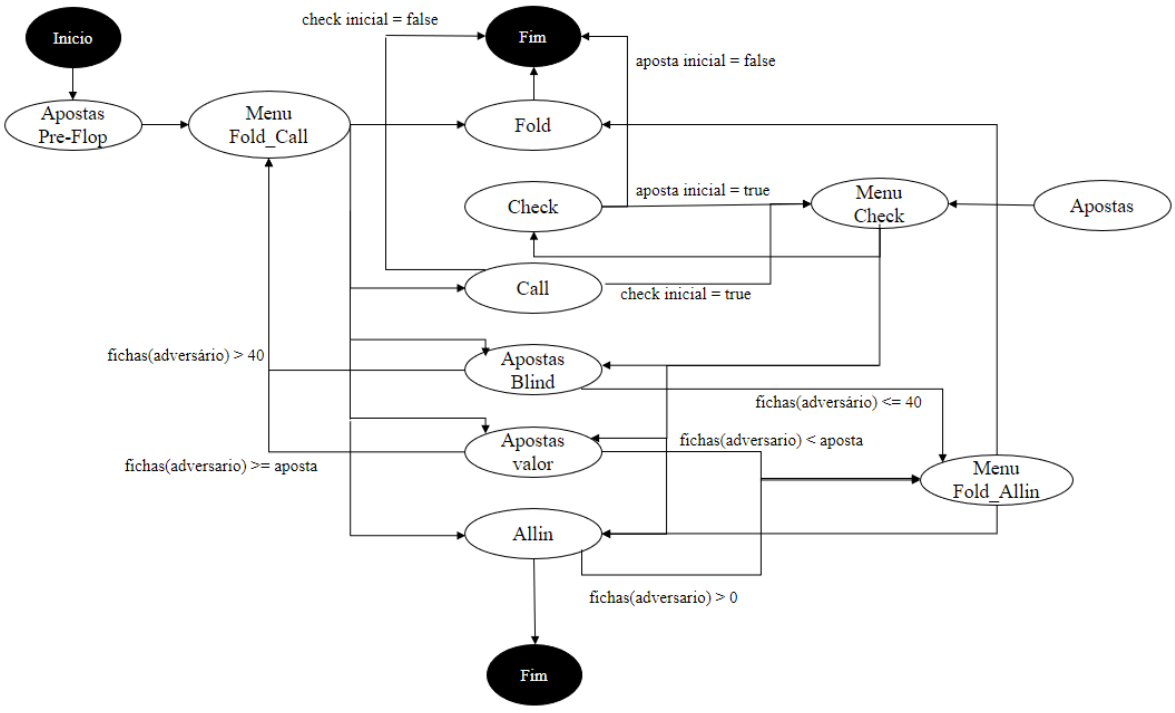


Figura 9 - Grafo das apostas

O método "apostasPreFlop()" é particularmente detalhado devido às apostas iniciais obrigatórias.

- "apostasPreFlop()": Este método é chamado antes de serem mostradas as cartas do *flop*. O método começa por verificar qual dos jogadores é o *dealer* e qual é a *blind*.

Depois verifica se a *blind* tem fichas suficientes para pagar a *big blind* e a *small blind*, se não tiver fichas suficientes para pagar ambos então têm que apostar as suas fichas todas e o *dealer* iguala a sua aposta, acabando assim a ronda de apostas. Se a *blind* tiver fichas para pagar apenas a *small blind* então é também obrigada a apostar todas as suas fichas mas o *dealer* tem a opção de igualar a aposta ou desistir da jogada. Se o *dealer* não tiver fichas suficientes para pagar a *small blind* então faz automaticamente *all-in*, sendo a *blind* obrigada a igualar aposta, acabado assim a ronda de apostas. Se ambos têm fichas suficientes para pagar a *small blind* e a *big blind* então o *dealer* é obrigado a pagar a *small blind* e a *blind* é obrigada a pagar a *big blind*, sendo assim decrementado das fichas de cada jogador o valor das suas apostas, aumentando assim o pote, passando de seguida para o método "menuFold_Call()".

- "apostasFlop()": após as cartas do *flop* serem apresentadas na mesa este é o método que será utilizado para fazer as apostas. Tal como no método anterior começa-se por verificar qual dos dois jogadores é o *dealer*, e desseguida é chamado o método "menuCheck()" para o jogador que está na posição da *big blind*.

O sistema de apostas é constituído por três menus diferentes, cada menu mostra as opções disponíveis ao utilizador através de um *switch*:

- "menuFold_Call()": este menu é constituído com as opções *fold*, *call*, aposta *blind*, aposta valor e *allin*, e também tem o cuidado de verificar se é a primeira jogada do jogo, ou seja, se "apostaInicial" é verdadeira, se sim e o jogador decidiu fazer *call* então chama o menu "menuCheck()".
- "menuCheck()": este menu tem as opções *check*, apostar *blind*, apostar valor e *all-in*. Na opção *check* é necessário verificar se "checkInicial" é verdadeiro se sim então o oponente ainda tem de apostar sendo assim chamado este menu novamente, mas com o oponente.
- "menuFold_Allin()": este menu simplesmente tem as opções *fold* e *all-in*.

Para terminar a explicação desta classe descreve-se cada método que efetua as apostas possíveis que o jogador pode efetuar:

- "fold()": este método recebe como argumento o jogador que deseja desistir da jogada e simplesmente soma as fichas que estão no pote às fichas do oponente, terminando assim a jogada.
- "call()": este método tem como objetivo igualar a aposta do adversário, e para isso é necessário verificar se o jogador que quer igualar a aposta tem dinheiro suficiente para pagar, ou seja, se "aposta[oponente] - aposta[jogador]) >= fichas[jogador]", se o jogador não tiver dinheiro suficiente então terá que fazer *all-in*, apostando assim as suas fichas todas, e assim a aposta do oponente será igual ao *all-in* do jogador, se o jogador tiver dinheiro suficiente para igualar a aposta então faz uma aposta no valor de "aposta[oponente] - aposta[jogador]" que é a diferença da aposta jogador com a aposta do adversário. Quando este método chega ao fim a ronda de apostas termina.
- "apostaBlind()": este método serve para o jogador fazer uma aposta no valor da *big blind*, ou seja, para fazer uma aposta no valor de 20 fichas. É necessário algum

cuidado neste método, para começar é necessário verificar se é a aposta inicial, se sim então a aposta tem de ser no valor da *big blind* + *small blind* porque o jogador é o *dealer* e ainda só pagou o valor da *small blind*. Se não for a aposta inicial então a aposta será de "aposta[oponente] + bigBlind" que é a aposta do adversário mais a aposta da *blind*, se o jogador não tiver dinheiro para fazer uma aposta no valor da *blind* então é chamado o método "allIn()". Quando este método chega ao fim é chamado de novo o menu "menuFold_Call()".

- "apostaValor()": neste método o jogador introduz um valor especifico entre o valor de duas *blinds* e as suas fichas, após o jogador efetuar a aposta tem que se verificar se o oponente tem fichas suficientes para pagar a aposta, se o oponente não tiver dinheiro suficiente então é chamado o menu "menuFold_Allin()" senão é chamado o menu "menuFold_Call()".
- "allIn()": neste método o jogador efetua uma aposta no valor total das suas fichas, no fim da aposta é necessário verificar se o oponente tem fichas suficientes para pagar o *all-in*, se sim então chama "menuFold_Call()" senão termina a jogada.

A figura 10 exemplifica a execução completa de uma mão de poker entre dois utilizadores utilizando os métodos da classe.

A classe apostas tem ainda um outro método importante, "modoJogo()". Este método recebe informação sobre a natureza de cada jogador, ou seja, se cada um é um utilizador ou o computador através de um algoritmo que simula um jogador, designado por "Bot". No caso de um utilizador são apresentados menus para a introdução da decisão pelo utilizador, no caso de um simulador são invocados métodos da classe "Simulador" que retornam a decisão.

- "modoJogo()": este método recebe três argumentos, o jogador, o modo de jogo e o menu onde o jogo se encontra. Existem onze modos de jogo, Jogador contra Jogador, Jogador contra "Bot_certo", Jogador contra "Bot_agressivo", Jogador contra "Bot_misto", Jogador contra "Bot_probabilidades", "Bot_certo" contra "Bot_agressivo", "Bot_certo" contra "Bot_misto", "Bot_certo" contra "Bot_probabilidades", "Bot_agressivo" contra "Bot_misto", "Bot_agressivo" contra "Bot_probabilidades" e "Bot_misto" contra "Bot_probabilidades". Através de um *switch*, com opções para os onze modos de jogo são chamados os métodos referentes a cada simulador de jogo.

3.7. Classe Jogar

O objetivo desta classe é criar três tipos de jogo em modo de texto, os tipos de jogo são, jogador (ou utilizador) contra jogador, jogador contra o computador e o computador contra o computador. A diferença entre jogador e computador é que o jogador vai buscar as suas ações das apostas aos menus de texto enquanto o computador vai buscar as suas ações ao seu algoritmo. Esta classe recebe em argumento o nome do utilizador e o modo de jogo, e antes de entrar no ciclo do jogo inicia as fichas dos jogadores a 1500€ e através de "Random()" define qual dos jogadores é o *dealer*.

```
(1500.0) jogador 1: 4♣ J♦
(1500.0) jogador 2: 10♠ 7♠
Pote: 0.0
Jogador 1 defina a sua aposta:
1 - Fold
2 - Call
3 - Apostas blind
4 - Apostar valor
5 - All in
2
O jogador 1 fez Call (10.0)
Jogador 2 defina a sua aposta:
1 - Check
2 - Apostas blind
3 - Apostar valor
4 - All in
2
O jogador 2 fez uma aposta no valor de 40
Jogador 1 defina a sua aposta:
1 - Fold
2 - Call
3 - Apostas blind
4 - Apostar valor
5 - All in
2
O jogador 1 fez Call (20.0)

(1460.0) jogador 1: 4♣ J♦
(1460.0) jogador 2: 10♠ 7♠
Pote: 80.0
Flop: 7♥ K♥ 10♦
Jogador 2 defina a sua aposta:
1 - Check
2 - Apostas blind
3 - Apostar valor
4 - All in
1
```

```
Jogador 1 defina a sua aposta:
1 - Check
2 - Apostas blind
3 - Apostar valor
4 - All in
3
Escolha um valor entre 40 e 1460.0
1000
O jogador 1 fez uma aposta no valor de 1000.0
Jogador 2 defina a sua aposta:
1 - Fold
2 - Call
3 - Apostas blind
4 - Apostar valor
5 - All in
5
O jogador 2 fez all in com 1460.0
Jogador 1 defina a sua aposta:
1 - Fold
2 - Call
2
O jogador 1 fez All in com (460.0)

(0.0) jogador 1: 4♣ J♦
(0.0) jogador 2: 10♠ 7♠
Pote: 3000.0
Turn: 7♥ K♥ 10♦ K♦

(0.0) jogador 1: 4♣ J♦
(0.0) jogador 2: 10♠ 7♠
Pote: 3000.0
River: 7♥ K♥ 10♦ K♦ 3♠

(0.0) jogador 1: 4♣ J♦
(0.0) jogador 2: 10♠ 7♠
Pote: 3000.0
Jogador1: K♦ K♥ J♦ 10♦ 7♥ 1 par
Jogador2: K♥ K♦ 10♦ 10♠ 7♠ 2 pares
Jogador 2 ganha 3000.0 com 2 pares
Jogador 2 Venceu o jogo!!!!
```

Figura 1 - Apostas.java

- "jogadorVSjogador()": este método simula um jogo entre dois utilizadores, mas para isso os jogadores tem que saber as suas cartas de cor, ou seja, são mostradas as cartas do primeiro jogador sem o segundo jogador as ver e mostradas as cartas do segundo jogador sem o primeiro jogador as ver, e no decorrer do jogo ambas as cartas estão escondidas.
- "jogadorVScomputador()": este método é responsável por criar um jogo entre o utilizador e um algoritmo escolhido previamente pelo utilizador. O método é semelhante ao método anterior com os seguintes passos dentro de um ciclo *while*:
    1. Obter um baralho de cartas baralhado.
    2. Distribuir as cartas pelos dois jogadores.
    3. Obter as probabilidades das cartas do computador antes do *flop*.
    4. Mostrar as fichas do jogador e do computador.
    5. Efetuar as apostas antes do *flop.*

6. Verificar se a jogada continua, se não continua, troca de *dealer* e volta ao passo 1.
7. Dar as três cartas do *flop*.
8. Obter as probabilidades das cartas do computador no *flop*.
9. Se algum dos jogadores não tiver fichas mostra as probabilidades de ambos os jogadores ganhar a jogada, senão mostra apenas as fichas dos jogadores e passa para o próximo passo.
10. Efetua as apostas no *flop*.
11. Verificar se a jogada continua, se não troca de *dealer* e volta ao passo 1.
12. Dar a carta do *turn*.
13. Obter as probabilidades das cartas do computador no *turn*.
14. Se algum dos jogadores não tiver fichas mostra as probabilidades de ambos os jogadores ganharem a jogada, senão mostra apenas as fichas dos jogadores e passa para o próximo passo.
15. Efetua as apostas no *turn*.
16. Verificar se a jogada continua, se não troca de *dealer* e volta ao passo 1.
17. Dar a carta do *river*.
18. Obter as probabilidades das cartas do computador no *river*.
19. Efetua as apostas no *river*.
20. Mostra as fichas dos jogadores.
21. Avalia a mão de ambos os jogadores.
22. Mostra o *ranking* da mão de ambos os jogadores.
23. Calcula qual a mão vencedora e soma o pote ás fichas do jogador que possui a mão vencedora.
24. Troca a posição do *dealer* e inicializa o pote a 0.
25. Verifica se algum dos jogadores não tem fichas, se alguém não tiver fichas então o outro jogador ganhou o jogo e o ciclo termina senão volta ao passo 1.

- "computadorVScomputador()": o ultimo método desta classe é semelhante aos métodos anteriores, mas apenas recebe uma variável em argumento que é o modo de jogo. Este método simula um jogo de *poker* entre dois algoritmos que simulam um jogador de *poker*. Através do modo de jogo recebido em argumento verifica-se qual os algoritmos a usar, como por exemplo, se o modo de jogo for igual a 6 então o jogador 1 usa o algoritmo do "Bot_certo" e o jogador 2 usa o algoritmo do "Bot_agressivo". O funcionamento deste método é igual ao método anterior, apenas com a diferença que agora precisa das probabilidades para ambos os jogadores em vez de apenas para o jogador 2.

3.8. Classe Estatísticas

O objetivo desta classe é criar estatísticas para o utilizador, tais como, as cartas mais jogadas, as cartas que ganharam mais jogos, a mão mais alta, número de jogadas ganhas e total de jogadas efetuadas. Esta classe funciona com duas versões, a versão apenas em modo de texto e a versão com a interface gráfica.

3.9. Classe Interface Gráfica

O objetivo desta classe é semelhante à da classe "Jogar" com a diferença que em vez de modo texto agora são criados dois tipos de jogo em modo gráfico. Os tipos de jogo são jogador (ou utilizador) contra o computador e o computador contra o computador. A diferença entre jogador e computador é que o jogador vai buscar as suas ações das apostas aos botões da interface gráfica enquanto o computador vai buscar as suas ações ao seu algoritmo. Para a realização desta classe foi necessário adaptar alguns métodos que estavam construídos na classe "Apostas.java". A construção gráfica é realizada através de *Swing* que é uma *framework* que disponibiliza um conjunto de elementos gráficos, nomeadamente, "JFrame" que representa a janela do simulador de *poker*, "JPanel" que é responsável pelo conteúdo da janela, "JLabel" para inserir texto e imagens, "JButton" para inserir botões, "JTextField" que é um campo de texto para o utilizador inserir dados e "JTextArea" que é uma caixa de texto.

As figuras 11 a 14 ilustram respetivamente a interface gráfica em pré-flop, no flop com utilizador vs computador, no flop com computador vs computador, e no final da mão com a determinação do vencedor.

Figura 11- Interface gráfica em Apostas pre-flop

3.10. Classe Main

Esta é a classe principal, onde tudo se inicia. Esta classe tem um pequeno menu que começa por perguntar ao utilizador em que modo deseja jogar, em modo de texto ou em modo gráfico, como mostra a figura 15.

Figura 12 - Interface gráfica no flop PC vs User

Figura 13 - Interface gráfica no flop PC vs PC

```
*************** Simulador Head's Up Texas Hold'em *****************

                1- Jogar em modo de texto.

                2- Jogar em modo gráfico.

                Introduza a sua opção:
```

Figura 14 - Menu inicial

Nesta classe para o modo gráfico são criadas duas janelas distintas através dos dois métodos principais desta classe,

- "paginaInicial()": a função deste método é criar a página inicial da interface gráfica do jogo de *poker*. Esta janela pede ao utilizador para introduzir o seu nome e entrar no jogo, como mostra a figura 16.
- "paginaMenus()": o objetivo deste método é criar a página em modo gráfico que contem todas as opções de jogo disponíveis ao utilizador, como ilustra a figura 17.

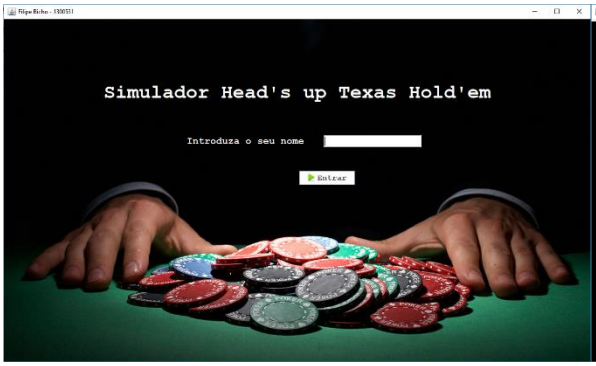


Figura 15 - Página inicial

Figura 16 - Página dos menus

4. Simulador de jogador de Poker

Nesta seção descrevem-se os algoritmos desenvolvidos para simular um jogador de poker. Para tal foi necessário calcular um conjunto de probabilidades associadas às várias rondas do jogo, coligidas na classe "Probabilidade", e desenvolver conjuntos de regras de decisão das apostas a efetuar a partir das probabilidades calculadas, coligidas na classe "Simulador".

4.1. Classe Probabilidade

Esta classe tem o objetivo de calcular as probabilidades no *poker*, sendo calculadas em quatro formas diferentes:

- Jogador contra Oponente: neste modo são conhecidas as cartas do jogador e do oponente, sendo calculadas as probabilidades de ambos ganharem o jogo. As probabilidades são calculadas no *flop* e no *turn*.

- Jogador contra Mão aleatória: neste modo apenas são conhecidas as cartas do jogador, sendo calculadas as probabilidades do jogador ganhar o jogo. As probabilidades são calculadas no *flop*, no *turn* e no *river*.
- Probabilidade de sair uma certa mão: neste modo apenas são conhecidas as cartas do jogador e é calculada a probabilidade de sair uma certa mão (*ranking*) final de *poker* para o jogador no *river*. As probabilidades são calculadas no *flop* e no *turn*.
- Probabilidade de o oponente ter uma certa mão: neste modo apenas são conhecidas as cartas da mesa e é calculada a probabilidade de o oponente ter certa mão (*ranking*) no *flop* e no *turn*. As probabilidades são calculadas no *flop* e no *turn*.

Jogador contra Oponente

Para calcular as probabilidades são simuladas todas as possibilidades de cartas por sair e a cada possibilidade verificado qual o jogador vencedor, criando assim as probabilidades de vitoria / derrota de cada jogada. Sabendo as cartas dos dois jogadores, este tipo de probabilidade é apenas possível de calcular no *flop* e no *turn*, porque, como por exemplo, para calcular estas probabilidades antes do *flop* seria necessário considerar $\binom{52-2-2}{5} = \binom{48}{5} = 1712304$ combinações de 5 cartas da mesa por sair, o que seria um valor demasiado alto para efetuar os cálculos. Para o caso antes do flop foi utilizado o método "probabilidadesPreFlop()" recorrendo-se à tabela que se encontra em http://www.natesholdem.com/pre-flop-odds.php onde foi extraída a primeira coluna referente a dois jogadores. As restantes probabilidades são calculadas pelos seguintes métodos:

- "probabilidadeJogadorvsJogadorFlop()": Para calcular as probabilidades dos dois jogadores ganharem o jogo no *flop* é necessário considerar $\binom{52-4-3}{2} = \binom{45}{2} = 990$ combinações de 2 cartas da mesa por sair e determinar para cada uma qual o jogador vencedor.
- "probabilidadeJogadorvsJogadorTurn()": é um método semelhante ao anterior com a pequena diferença que agora a mesa tem 4 cartas, assim existem apenas $\binom{44}{1} = 44$ combinações de 1 carta da mesa por sair.

Estas probabilidades são calculadas em situações de all-in para serem mostradas a título informativo para os jogadores.

Jogador contra Mão aleatória

No cálculo destas probabilidades não se sabe as cartas do oponente. Estas probabilidades são apenas necessárias para a construção de um simulador de jogador de *poker*, que baseia as suas apostas nas probabilidades. Tal como no calculo anterior não é possível calcular estas probabilidades antes do *flop* porque o numero de combinações é muito alto. As probabilidades são calculadas pelos seguintes métodos:

- "probabilidadesFlop()": Para calcular as probabilidades do jogador ganhar o jogo no *flop* é necessário considerar $\binom{52-2-3}{2} = \binom{47}{2} = 1081$ combinações de 2

cartas do oponente vezes $\binom{52-2-2-3}{2} = \binom{45}{2} = 990$ combinações de 2 cartas da mesa por sair, num total de 1070190 combinações de 4 cartas, e determinar para cada uma qual o jogador vencedor. Este cálculo demorou 14 segundos a executar, sendo considerado um tempo exagerado. Para contornar este problema, as iterações dos ciclos *for* incrementarem-se de 10 em 10 valores, diminuindo assim para $\frac{\binom{47}{2}}{10}\frac{\binom{45}{2}}{10}$ combinações, ou seja, 10701 combinações, demorando assim apenas 0,6 segundos a executar, o que é um tempo aceitável, apenas com a contrapartida que as probabilidades têm uma margem de erro de no máximo 10%, tendo-se chegado a este valor após várias verificações, e a margem de erro nunca excedeu esse valor.

- "probabilidadesTurn()": este método é semelhante ao método anterior, apenas com a diferença que agora a mesa tem quatro cartas em vez de três, sendo necessário considerar $\binom{52-2-4}{2} = \binom{46}{2} = 1035$ combinações de 2 cartas do oponente vezes $\binom{44}{1} = 44$ combinações de 1 carta da mesa por sair, num total de 45540 combinações de 3 cartas.
- "probabilidadesRiver()": neste método já se sabe todas as cartas da mesa, então apenas é necessário atribuir cartas ao oponente, o que dá um total de, $\binom{52-2-5}{2} = \binom{45}{2}$ = 990 combinações possíveis.

Probabilidade de sair uma certa mão (*ranking*)

Estas probabilidades também são apenas necessárias para a construção de um simulador de jogador de *poker*. As probabilidades são calculadas pelos seguintes métodos:

- "potencialFlop()": Para no *flop* calcular as probabilidades do jogador ter uma certa mão (*ranking*) no *river*, é necessário considerar $\binom{52-2-3}{2} = \binom{47}{2} = 1081$ combinações de 2 cartas da mesa por sair e determinar para cada uma qual o seu *ranking*.
- "potencialTurn()": este método é igual ao método anterior apenas com mais uma carta na mesa recebida, ou seja, assim existem 46 cartas no baralho (52-2-4), dando um total de $\binom{52-2-4}{1} = \binom{46}{1}$ = 46 combinações de 1 carta da mesa por sair.

Probabilidade de o Oponente ter uma certa mão (*ranking*)

Estas probabilidades a ser calculadas têm o mesmo objetivo que as anteriores, auxiliar na construção de um simulador de *poker*. É calculada a probabilidade de o oponente ter uma determinada mão (*ranking*) apenas sabendo as cartas da mesa. Os cálculos são efetuados para o *flop* e para o *turn* pelos seguintes métodos:

- "potencialMaoFlop()": o objetivo neste método é calcular qual a mão (*ranking*) mais provável que o adversário pode ter no *flop*, para isso é apenas necessário as cartas do *flop* que são recebidas em argumento. Para construir o algoritmo é necessário ter em conta que sobram 49 cartas no baralho (52 – 3 (*flop*)), existindo

assim $\binom{49}{2} = 1176$ combinações de 2 cartas que o oponente pode ter e determinar para cada uma qual o seu *ranking*.

- "potencialMaoTurn()": estas probabilidades são semelhantes às probabilidades de o oponente ter uma certa mão no *flop*, apenas com a diferença que agora existem 4 cartas na mesa, e assim existem $\binom{48}{2} = 1128$ combinações de 2 cartas que o oponente pode ter.

4.2. Classe Simulador

Esta classe tem como objetivo desenvolver algoritmos que consigam simular um jogador de *poker*, sendo o funcionamento de cada um desses algoritmos explicados em detalhe.

4.2.1. Mathematically Fair Strategy

"MFS()": o primeiro simulador é desenvolvido baseado na formula "Mathematically Fair Strategy – MFS" , ligeiramente alterada para um funcionamento eficiente apenas com dois jogadores (https://projetos.inf.ufsc.br/arquivos_projetos/projeto_773/tcc%20final%20e%20artigo.pdf),

$$V = probabilidade_{vitoria} \times aposta_{oponente} - probabilidade_{derrota} \times (aposta_{oponente} - aposta_{jogador})$$

depois com o valor retornado é determinada a melhor decisão. O simulador que usa esta formula joga de maneira justa em relação à probabilidade de vitoria, as apostas são feitas de maneira proporcional às probabilidades e ás apostas dos dois jogadores. Este método recebe como argumentos, o jogador, as fichas dos jogadores e do pote, as probabilidades, as apostas dos jogadores e qual o menu em que o jogo se encontra no momento e retorna a melhor opção consoante o resultado da fórmula. Por exemplo, se o jogador estiver no menu "Fold_Call" tem as opções *fold*, *call*, aposta *blind*, aposta valor e *allin* e assim retorna uma dessas opções. Neste método á definida também uma constante T com o valor de 20. Se o jogador estiver no menu "Check" significa que não existe apostas e assim não é necessária a formula MFS, a escolha da melhor opção é realizada apenas com as probabilidades de vitória.

Se a probabilidade de vitoria for menor que 55% a opção será *check*, se a probabilidade de vitória for entre 55% e 65% efetua uma aposta de uma *blind.* se a probabilidade de vitória for entre 65% e 75% efetua uma aposta no valor do pote, se a probabilidade de vitória for entre 75% e 85% efetua uma aposta no valor do pote + 10 *blinds* e por fim se a probabilidade de vitória for maior que 85% efetua *allin*.

Se já existir uma aposta então é necessário calcular a decisão através da fórmula, e com o valor retornado é verificada qual a melhor opção a ser tomada, se o jogo se encontrar no menu "Fold_Allin" então apenas é verificado se o valor retornado é menor ou igual a 5T, se sim então retorna a opção de *fold* senão faz *allin*.

Se o jogo se encontrar no menu "Fold_Call" então começa-se por verificar se V é um valor negativo, se sim faz *fold*, se V estiver entre 0 e T faz *check*, se tiver entre T e 2T aposta

uma *blind*, se tiver entre 2T e 5T aposta o valor do pote, se tiver entre 5T e 10T aposta o valor do pote + 10 *blinds* e por fim se tiver mais de 10T aposta tudo.

```
T= 20
W = Probabilidade vitoria
L = Probabilidade derrota
Se Menu = "Menu_Check"
        Se W <= 55
                Fold
        Se W > 55 e W <= 65
                Aposta blind
        Se W > 65 e W <= 75
                Aposta pote
        Se W > 75 e W <= 85
                Aposta Pote+10*blind
        Se W > 85
                Aposta tudo
Senão
        V = (W x Aposta_oponente - L * (Aposta_oponente - Aposta_jogador) )/10
        Se Menu = "Fold_Allin
                Se V <= 5T
                        Fold
                Senao
                        Allin
        Se Menu = "Fold_Call"
                Se V <= 0
                        Fold
                Se V > 0 e V <= T
                        Call
                Se V > T e V <= 2T
                        Aposta blind
                Se V > 2T e V <= 5T
                        Aposta pote
                Se V > 5T e V <= 10T
                        Aposta pote+10*blinds
                Se V > 10T
                        Aposta tudo
```

Figura 28 - MFS pseudocódigo

Os próximos algoritmos para simular um jogador de poker já são mais elaborados, sendo constituídos por vários métodos.

4.8.2. Bot_certo

Este algoritmo simula um jogador de *poker* que efetua apostas apenas quando as probabilidades lhe são favoráveis. Este tipo de jogador é constituído pelos métodos, "preFlop_certo" que retorna a melhor opção quando o jogo se encontra no pre *flop*, "flop_certo" que retorna a melhor opção quando o jogo se encontra no *flop* ou no *turn* e "river_certo" que retorna a melhor opção quando o jogador se encontra no *river*. Para construir o algoritmo do "Bot_certo" são necessárias as seguintes informações, se o jogador está na posição de *dealer*, as fichas dos jogadores e do pote, as apostas dos dois jogadores, em que ronda se encontra o jogo, qual o menu em que se está a escolher a opção de jogo, as cartas dos jogadores, as cartas da mesa e as probabilidades do jogador.

- "preFlop_certo()": o objetivo deste método é retornar a melhor opção de jogo quando o jogo se encontra no inicio, ou seja, antes do *flop*. O método começa por converter as fichas dos jogadores para *stacks*, cada *stack* é equivalente a fichas/*blind*, como por exemplo, se o jogador tiver 1200 fichas e a *blind* for igual a 20 fichas então a *stack* do jogador é de 60 *blinds*, depois verifica em que grupo se encontra as cartas do jogador, para isso recorro ao método "grupo()" que é apenas um método que coloca as cartas do jogador num determinado grupo de acordo com https://www.intelipoker.com/articles/Grupos-e-Categorias-de-Maos. Após estes dois passos, chegou a altura de verificar qual a melhor opção de jogo tendo em conta os argumentos recebidos, como esta explicação é longa e repetitiva é apresentado na figura 15 o seu pseudocódigo para melhor compreensão

```
call = aposta_oponente - aposta_jogador
stack = fichas / blind
grupo = grupo em que as cartas se encontram

Se call <= blind
        Se stack_jogador < 8 ou (stack_oponente+call) < 8
                Apostar tudo
        Se (stack_jogador >= 8 e stack_jogador < 12) ou (stack_jogador >= 8 e stack_jogador < 12)
                Se o jogador é dealer
                        Se grupo <= 8
                                Apostar tudo
                        Senão
                                Passar
                Senão
                        Se grupo <= 7
                                Apostar tudo
                        Senão
                                Passar
        Se stack_jogador > 12 e stack_oponente > 12
                Se o jogador é o dealer
                        Se grupo <= 8
                                Se grupo <= 5
                                        Apostar 6Blinds
                                Senão
                                        Apostar 3Blinds
                        Senão
                                Passar
                Senão
                        Se grupo <= 7
                                Se grupo <= 4
                                        Apostar 8Blinds
                                Senão
```

```
                                        Apostar 4Blinds
                        Senão
                                Passar
Se call > blind
        Se stack_jogador < 2 ou (stack_oponente+call) < 2
                Apostar tudo
        Se (stack_jogador >= 2 e stack_jogador < 4) ou (stack_jogador >= 2 e stack_jogador < 4)
                Se grupo <= 7
                        Apostar tudo
                Senão
                        Desistir
        Se (stack_jogador >= 4 e stack_jogador < 12) ou (stack_jogador >= 4 e stack_jogador < 12)
                Se grupo <= 6
                        Apostar tudo
                Senão
                        Desistir
      Se stack_jogador > 12 ou stack_oponente > 12
              Se call <= 3Blinds
                      Se grupo <= 7
                              Se grupo <= 5
                                      Apostar 4Call
                              Senão
                                      Passar
                      Senão
                              Desistir
              Se call > 3Blinds e call <= 6Blinds
                      Se grupo <= 6
                              Se grupo <= 4
                                      Se grupo <= 2
                                              Apostar 4Call
                                      Senão
                                              Apostar 2Call
                              Senão
                                      Igualar Aposta
                      Senão
                              Desistir
              Se call > 6Blinds e call <= 12Blinds
                      Se grupo <= 5
                              Se grupo <= 3
                                      Se grupo <= 2
                                              Apostar 4Call
                                      Senão
                                              Apostar 2Call
                              Senão
                                      Igualar Aposta
                      Senão
                              Desistir
              Se Call > 12Blinds
                      Se grupo <= 4
                              Se grupo <= 3
                                      Se grupo <= 2
                                              Apostar 4Call
                                      Senão
                                              Apostar 2Call
                              Senão
                                      Igualar Aposta
                      Senão
                              Desistir
```

Figura 19 - Pseudocódigo preFlop_certo

Como visto no pseudocódigo o método resume-se a condições *if* que verifica o estado do jogo e calcula qual a melhor opção.

- "flop_certo()": este método é um pouco mais completo que o anterior porque calcula a melhor opção nas rondas do *flop* e do *turn*, e para isso é necessário toda a informação disponível sobre a jogada. Este método começa por verificar em que ronda se encontra, e define o seguinte:
    - "percentagemMaoOponente": fica com as probabilidades de o oponente ter uma certa mão no *flop* ou no *turn* através do método "potencialMaoFlop()" explicado em probabilidades.
    - "percentagemMaoPotencial": fica com as probabilidades de sair uma certa mão ao jogador através do método "potencialFlop()".
    - "probabilidadePiorJogo": fica com a probabilidades de o jogador ter uma mão pior que a mão do oponente.
    - "probabilidadeMelhorarMao": fica com a probabilidade de a mão do jogador ser melhorada

Após reunir toda a informação disponível chegou a altura de verificar qual a melhor opção de jogo tendo em conta todas as variáveis, para isso foram definidas as várias condições *if* que podem ser verificadas no pseudocódigo da figura 20.

```
Stack = fichas / Blind
Call = Apostas_oponente - Apostas_jogador
probabilidades = probabilidades de o jogador ganhar a jogada
probabilidadePiorJogo = probabilidades de o jogador ter uma mão pior que a mão do oponente
probabilidadeMelhorarMao =  probabilidade de sair uma certa mão ao jogador

Se Call = 0
        Se probabilidades < 50
                Passar
        Senão
                Se stack_jogador < 8 ou stack_oponente < 8
                        Apostar tudo
                Se (stack_jogador >= 8 e stack_jogador < 12) ou (stack_jogador >= 8 e stack_jogador < 12)
                        Se probabilidadeMelhorarMao  > 70
                                Apostar 5Blinds
                        Senão Se probabilidadePiorJogo < 40
                                Apostar tudo
                        Senão
                                Passar
                Se stack_jogador > 12 ou stack_oponente > 12
                        Se probabilidadePiorJogo < 40
                                Se probabilidadePiorJogo < 20
                                        Apostar 12Blinds
                                Senão
                                        Apostar 8Blinds
                        Senão
                                Se probabilidadeMelhorarMao  > 80
                                        Apostar 8Blinds
                                Senão
                                        Passar
```

```
Senão
        potOdds = call / (pote+call)
        Se potOdds > probabilidades ou probabilidaes < 50
                Desistir
        Senão
                Se stack_jogador < 4 ou (stack_oponente+call) < 4
                        Se probabilidadePiorJogo < 50 ou probabilidadeMelhorarMao > 70
                                Apostar tudo
                        Senão
                                Igualar aposta
                Se (stack_jogador >= 4 e stack_jogador < 8) ou (stack_jogador >= 4 e stack_jogador < 8)
                        Se probabilidades > 60 ou probabilidadePiorJogo < 40 ou probabilidadeMelhorarMao > 60
                                Apostar tudo
                        Senão
                                Igualar aposta
                Se (stack_jogador >= 8 e stack_jogador < 15) ou (stack_jogador >= 8 e stack_jogador < 15)
                         Se probabilidades > 75 ou probabilidadePiorJogo < 35 ou probabilidadeMelhorarMao > 75
                                Apostar tudo
                         Senão Se probabilidaes > 65 ou probabilidadePiorJogo < 40 ou probabilidadeMelhorarMao > 70
                                Apostar 2Call
                        Senão
                                Igualar aposta
                Se stack_jogador > 15 ou stack_oponente > 15
                        Se call <= 5Blinds
                                Se o jogador é o dealer
                                         Se probabilidaes > 75 ou probabilidadePiorJogo < 35
                                                 Apostar 4Call
                                         Senão Se probabilidades > 60
                                                 Apostar 2Call
                                        Senão se probabilidadeMelhorarMao > 60
                                                Igualar aposta
                                        Senão
                                                Desiste
                                Senão
                                        Se probabilidaes > 80 ou probabilidadePiorJogo < 30
                                                Apostar 4Call
                                        Senão Se probabilidades > 65
                                                Apostar 2Call
                                        Senão se probabilidadeMelhorarMao > 65
                                                Igualar aposta
                                        Senão
                                                Desiste
                        Se call > 5Blinds
                                Se o jogador é o dealer
                                        Se probabilidaes > 75 ou probabilidadePiorJogo < 35
                                                Apostar 4Call
                                        Senão Se probabilidades > 65
                                                Apostar 2Call
                                        Senão se probabilidadeMelhorarMao > 80
                                                Igualar aposta
                                        Senão
                                                Desiste
                                Senão
                                        Se probabilidaes > 80 ou probabilidadePiorJogo < 30
                                                Apostar 4Call
                                        Senão Se probabilidades > 75
                                                Apostar 2Call
                                        Senão se probabilidadeMelhorarMao > 85
                                                Igualar aposta
                                        Senão
                                                Desiste
```

Figura 20 - Pseudocódigo flop_certo

- "river_certo()": este é o método responsável por escolher a melhor a opção de jogo quando o jogo se encontra no *river*. Para calcular a melhor opção é necessário saber

a *stack* do jogador, qual o valor de *call* e as probabilidades de ganhar a jogada e para definir qual a melhor a melhor opção foi usada a mesma estratégia dos métodos explicados anteriormente, como mostra o pseudocódigo da figura 21.

```
Stack = fichas / Blind
Call = Apostas_oponente - Apostas_jogador
probabilidades = probabilidades de o jogador ganhar a jogada

Se call = 0
        Se probabilidades < 50
                Passar
        Senão
                Se stack_jogador < 8 ou stack_oponente < 8
                        Apostar tudo
                Se (stack_jogador >= 8 e stack_jogador < 12) ou (stack_jogador >= 8 e stack_jogador < 12)
                        Se probabilidades > 70
                                Apostar tudo
                        Senão
                                Passar
                Se stack_jogador > 12 ou stack_oponente > 12
                        Se probabilidades > 70
                                Se probabilidades > 80
                                        Apostar 12Blinds
                                Senão
                                        Apostar 8Blinds
                        Senão
                                Passar
Senão
        Se potOdds > probabilidades ou probabilidaes < 50
                Desistir
        Senão
                Se stack_jogador < 4 ou (stack_oponente+call) < 4
                        Apostar tudo
                Se (stack_jogador >= 4 e stack_jogador < 8) ou (stack_jogador >= 4 e stack_jogador < 8)
                        Se probabilidades > 70
                                Apostar tudo
                        Senão
                                Igualar aposta
                Se (stack_jogador >= 8 e stack_jogador < 15) ou (stack_jogador >= 8 e stack_jogador < 15)
                         Se probabilidades > 75
                                Apostar tudo
                         Senão
                                Igualar aposta
                Se stack_jogador > 15 ou stack_oponente > 15
                        Se call <= 5Blinds
                                Se probabilidaes > 85
                                        Apostar 4Call
                                Senão
                                        Igualar aposta
                        Se call > 5Blinds
                                Se probabilidaes > 95
                                        Apostar tudo
                                Senão Se probabilidaes > 90
                                        Apostar 4Call
                                Senão Se probabilidaes > 80
                                        Apostar 3Call
                                Senão Se probabilidaes > 70
                                        Igualar aposta
                                Senão
                                        Desistir
```

Figura 21 - Pseudocódigo river_certo

- "bot_certo()": este é o método central do simulador de apostas certas, é por aqui que a informação chega aos outros métodos, Este método começa por criar um *array* com o nome "opcao" de tamanho 2 em que na primeira posição fica a opção de jogo

e na segunda posição o valor da aposta. Após declarado o *array* é verificado em que ronda o jogo se encontra, se o jogo estiver no pre *flop* "opcao" fica com o valor retornado por "preFlop_certo()", se estiver no *flop* ou no *turn* fica com o valor retornado por "flop_certo()" e por fim se a ronda estiver no river "opcao" fica com o valor retornado por "river_certo();" ficando assim concluído o simulador de apostas certas.

4.8.3. Bot_agressivo

Esta versão que simula um jogador de *poker* é semelhante à versão anterior, apenas com alguns ajustes para tornar o modo de apostar mais agressivo, ou seja, aposta valores mais altos com probabilidades de vitoria mais baixos.

4.8.4. Bot_misto

Este tipo de jogador é um misto entre o jogador "Bot_certo" e "Bot_agressivo", ou seja, usa os dois algoritmos para efetuar apostas, para isso é usado um número aleatório entre 0 e 1, em que se sair o número 0 é chamado o "Bot_certo" e se sair o número 1 é chamado o "Bot_agressivo", ficando assim desenvolvido um algoritmo que é uma mistura dos dois algoritmos anteriores.

As figuras 22 a 27 mostram um dos aspetos mais interessantes deste projeto, a possibilidade de confronto entre os vários simuladores de jogadores de poker. Dos algoritmos desenvolvidos, nenhum parece ter uma clara vantagem sobre todos os outros.

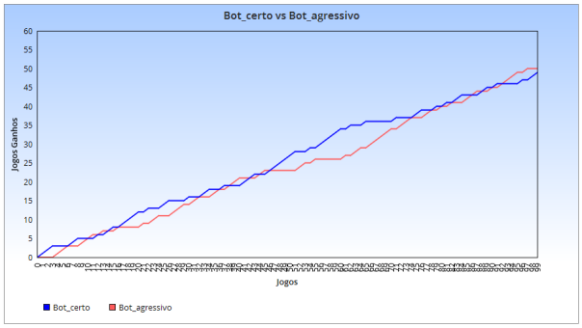


Figura 22 - Bot_certo vs Bot_agressivo

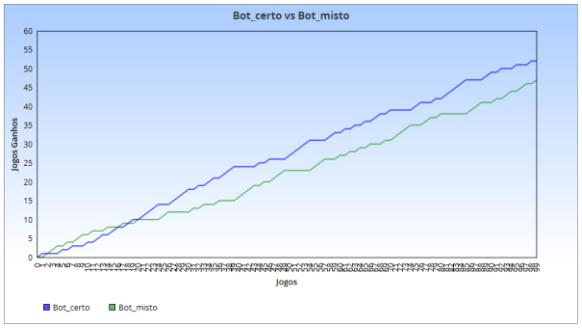


Figura 23 - Bot_certo vs Bot_misto

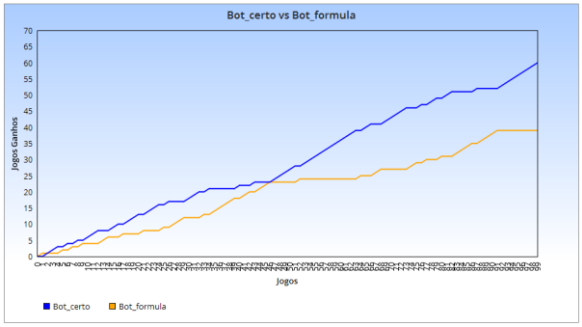


Figura 24 - Bot_certo vs Bot_fórmula MFS

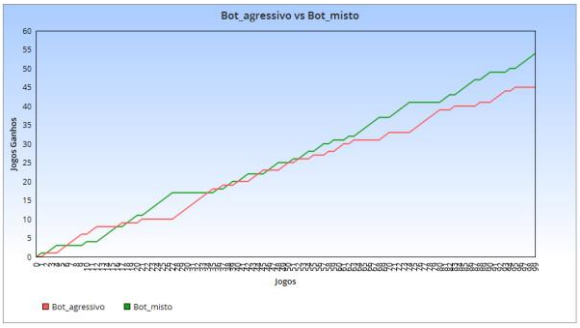
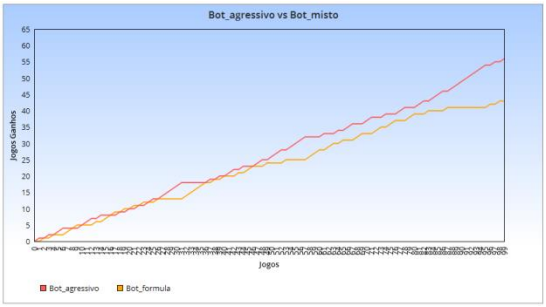


Figura 23 - Bot_certo vs Bot_misto



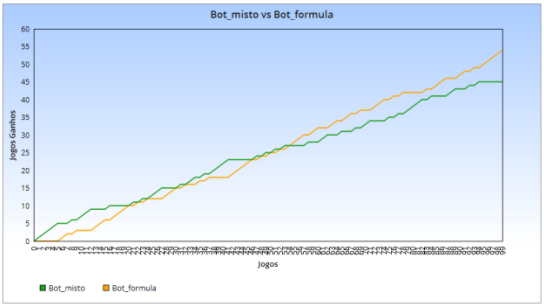


Figura 26 - Bot_agressivo vs Bot_fórmula MFS Figura 27 - Bot_misto vs Bot_fórmula MFS

5. Conclusões e Trabalho Futuro

A realização deste trabalho foi um desafio muito interessante apesar das dificuldades encontradas ao longo do seu desenvolvimento, nomeadamente a classe das apostas e das probabilidades. A parte que gostei mais de desenvolver foi a classe simulador em que tinha de tomar decisões consoante as probabilidades, a ronda do jogo, se o jogador é o *dealer* e as apostas efetuadas, esta classe faz uso de todas as classes anteriormente desenvolvidas exceto as classes das estatísticas e da interface gráfica.

Os próximos passos a efetuar neste projeto serão o desenvolvimento de uma aplicação *android*, melhorar a eficácia dos algoritmos para calcular as probabilidades, com especial foco nos algoritmos que calcula as probabilidades sem saber as cartas do adversário e dar uma maior dinâmica aos algoritmos que simulam um jogador de *poker*.

Referências

Fisher, Ronald A.; Yates, Frank (1948). Statistical tables for biological, agricultural and medical research (3rd ed.). London: Oliver & Boyd. pp. 26–27. OCLC 14222135.

Jogo Poker (2012). História do poker. Site Jogo Poker http://www.jogopoker.com/historia-poker

Filipe Bicho é licenciado em Informática pela Universidade Aberta. Atualmente é Software Developer na empresa REFUSiON GmbH. Tem conhecimentos de HTML, CSS, JavaScript, Java, angularJS, jQuery, MySQL e C/C++. Fluente em português, inglês e alemão.

Paulo Shirley, Professor Auxiliar no Departamento de Ciências e Tecnologia (DCeT). Coordenador da Licenciatura em Informática no triénio 2014–2016 e Vice-Coordenador do Mestrado Tecnologias e Sistemas Informáticos Web no biénio 2014–2015. Licenciado em Engenharia Electrotécnica e de Computadores em 1988 pelo IST-UTL. Obteve os graus de Mestre (perfil de Controlo e Robótica) e de Doutor em Eng. Electrotécnica e de Computadores, pelo IST-UTL em 1993 e 2003 respetivamente. Tem como áreas de interesse, a intersecção da Informática com a área do Controlo Automático, nomeadamente a área da "Computação de Alto Desempenho".