

Mestrado Integrado em Engenharia Electrotécnica e de Computadores

**Programação Orientada por Objectos**

2º Semestre – 2015/2016

**Relatório do Projeto**

Math in Casinos: Blackjack

Grupo 2

73065 – João Francisco Ribeiro Silva

73177 – Ana Catarina Rosa Gonçalves

74319 – Luís Eduardo Gomes Sobral Gonçalves

Prof. Alexandra Carvalho

**Data de Entrega:** 15/05/2016

**Índice**

|  |  |  |  |
| --- | --- | --- | --- |
| 1. Objetivos |  |  | 1 |
| Manual de utilizador |  |  | 2 |
| 1. Classes e Enumerados e Interfaces |  |  | 3 |
| 1. Modo de Funcionamento |  |  | 5 |
| 1. Interface Gráfica |  |  | 5 |
| 1. Conclusões |  |  | 7 |

1. **Objetivos**

Este projeto tem como principal objetivo o desenvolvimento de um jogo de Blackjack de acordo com as regras estipuladas, com side-rules implementadas e com possibilidade de contagem de cartas.

Para tal existem três métodos de jogo, *Debug*, *Simulation* e *Interactive*. Tal como os nomes indicam, o modo de *Debug* é utilizado para o teste dos comandos possíveis do jogo através de dois ficheiros dados, um com a ordem do baralho e o segundo com a ordem dos comandos a realizar. Esse serve para testar os casos limite da execução do programa. O modo *Simulation* permite escolher a combinação de métodos de contagem, um que influencia a aposta e o outro a ação tomada de modo a testar qual a combinação mais eficaz e quantificar a vantagem na utilização de métodos de contagem de cartas no caso de existir. Por fim, o modo *Interactive* permite jogar blackjack sendo que apenas existe um player que pode fazer até três splits logo pode jogar com quatro mãos. Aqui são dados advices ao jogador de acordo com os dois métodos de contagem definidos quando são pedidos através do comando ad. Para os últimos dois métodos são feitas estatísticas. Estas são apresentadas no fim do jogo/simulação ou no caso de ser pedido com o comando st.

1. **Manual de utilizador**

No caso de se querer executar o *Interactive mode* o comando a utilizar no terminal é o seguinte:

java -jar <<Grupo\_2>>.jar -i min-bet max-bet balance shoe shuffle

* **-i** indica que será executado o modo Interactiv.
* **min-bet** é a aposta mínima, tem de ser superior a 1.
* **max-bet** é a aposta máxima, tem de ser superior a 10\*min-bet e inferior a 20\*min-bet.
* **balance** é o valor de dinheiro inicial, tem de ser superior a 50\*min-bet.
* **shoe** é o número de baralhos de 52 cartas utilizados, têm de ser entre quatro e oito.
* **shuffle** é a percentagem do baralho utilizado a partir da qual se volta a baralhar as cartas.

Para o *Debug mode* o comando a utilizar no terminal é o seguinte:

java -jar <<Grupo\_2>>.jar -d min-bet max-bet balance shoe-file cmf-file

* **-d** indica que será executado o modo de Debug.
* **min-bet** é a aposta mínima, tem de ser superior a 1.
* **max-bet** é a aposta máxima, tem de ser superior a 10\*min-bet e inferior a 20\*min-bet.
* **balance** é o valor de dinheiro inicial, tem de ser superior a 50\*min-bet.
* **shoe** é um ficheiro que contém uma sequência de cartas que serão utilizadas sem ser baralhadas e como um baralho.
* **cmf-file** é um ficheiro que contém uma sequência de comandos efetuados pelo jogador.

Por fim, quando se pretende executar o *Simulation mode* utiliza-se no terminal o seguinte comando:

java -jar <<Grupo\_2>>.jar -s min-bet max-bet balance shoe shuffle s-number strategy

* **-s** indica que será executado o modo Simulation.
* **min-bet** é a aposta mínima, tem de ser superior a 1.
* **max-bet** é a aposta máxima, tem de ser superior a 10\*min-bet e inferior a 20\*min-bet.
* **balance** é o valor de dinheiro inicial, tem de ser superior a 50\*min-bet.
* **shoe** é o número de baralhos de 52 cartas utilizados, têm de ser entre quatro e oito.
* **shuffle** é a percentagem do baralho utilizado a partir da qual se volta a baralhar as cartas.
* **s-number** é o número de vezes que se baralha o conteúdo do shoe.
* **strategy** é a combinação de estratégias a utilizar.

Neste projeto não foi implementada a parte gráfica, logo não se justifica inserir o comando para tal visto que não irá funcionar.

1. **Classes e Enumerados**
2. *GameState*

É um enumerado que guarda cinco estados do jogo definidos. Estes são, Betting, que é indica a espera por uma aposta, waitingTheD que representa a espera pela distribuição das cartas. Dealing, o estado em que são dadas as primeiras cartas, Opening\_Hand que indica que a mão do jogador é constituída por apenas duas cartas. Por fim, o estado MORE\_CARDS que é o estado representante do jogo em si, fazer hit ou stand.

1. *GameMode*

É também um enumerado. Contém os três modos de jogo, Interactive, Debug e Simulation.

1. *GameRunner*

Esta classe contém o main e é responsável pela leitura dos argumentos do terminal para os três métodos de funcionamento de jogo, (GameMode). É aqui que são também criadas a table e o player.

1. *Player*

Para esta classe foram criados dois construtores, um utilizado no método Debug e o outro para o Simulation. Sendo que no primeiro o player é constituído pelo nome, balance, e dois ArrayList, um que contém as mãos do jogador e o outro a lista de comandos. É composta pelos métodos addMoney, getCurrantBalance, getStrategy, getBettingStratagy, getPoor, defineStrategy.

1. *Hand*

A Hand é criada tendo como base um ArrayList de cartas e um boleano que representa se a mão já foi jogada. Tem três construtores, que criam a mão com zero cartas, uma carta ou duas cartas. Contém os métodos getHandSum, addCard, getNumCards, setNumCards, getCards, isBusted, isBlackjack, isSoft, isFinished, printHand e emptyHand.

1. *PlayerHand*

Esta classe faz extend da Hand acrescentando aos seus construtores um inteiro, betAmount, dois boleanos, beenSplit e doubleDown e um Player. Para além dos métodos definidos na Hand contém isPair, beenSplit, beenDoubled, getName, getBetAmount e makeSplit.

1. *Suit*

Enumerado que contem os quatro naipes de cartas possíveis.

1. *Card*

A Card possui um construtor que cria um objeto composto por um naipe (Suit) e por um inteiro, que representa o número da carta, servindo o número um para o Ás e os números onze, doze e treze para o Valete, Dama e Reis respetivamente. Contém os métodos: getNumber, getValue, toString e isAce.

1. *Deck*

O *Deck* consiste num *ArrayList* com as cartas que serão utilizadas no decorrer do jogo, como tal, possui 3 construtores, um que cria um Deck com apenas um conjunto de 54 cartas, um segundo que cria o mesmo Deck, mas com um numero pre-recebido de baralhos, e por ultimo um construtor que recebe um baralho pre-ordenado, com as cartas em posições estratégicas para testar o jogo.

1. *Chips*

É um enumerado que contém os quatro tipos de fichas de jogo possíveis. Apenas seria utilizado se fosse implementada uma interface gráfica definida para receber a aposta em *chips* e não em *amout*.

1. *Table*

A classe *Table* é composta por três construtores, cada um deles referente a um dos diferentes métodos de jogo, e pelos métodos *getMinimumBet*, *getMaximumBet*, *playRounds*, *bettingProcess*, *dealHouse*, *dealPlayers*, *playerTurn*, *dealerTurn*, *checkWinner*, *clearTable*, *placeBets*, *readCommand*, *convertActionToCommand*, *handleSplit*, *handleInsurance*, *handleSurrender*, *showStatsStrategy*, *BasicStrategy*, *HiLoStrategy*, *StandardBettingStrategy*, *AceFiveStrategy*.

1. *Strategy*

A *Strategy* é uma interface que possui cinco métodos implementados pelas classes: *BasicStrategy*, *HiLoStrategy*, *StandardBettingStrategy*, *AceFiveStrategy*. Os métodos definidos são: *actionToTake*, *nextBet*, *countCard*, *resetCounters*, *getWhich*.

1. *BasicStrategy* e *HiLoStrategy*

Estas classes implementam os métodos da interface *Strategy*, utilizando o método actionToTake para proceder a contagem das cartas.

1. *StandardBettingStrategy* e *AceFiveStrategy*

Estas classes implementam os métodos da interface *Strategy*, utilizando o método *nextBet*, de modo a calcular o montante da aposta seguinte com base nos resultados do jogo.

1. **Modo de Funcionamento**

Após o comando para o modo interativo são lidos dos argumentos, o valor das apostas mínima e máxima, o valor inicial do *balance*, o número de baralhos e a percentagem de baralho utilizada até ao *shuffle* seguinte. Estes valores são inseridos nos construtores do *Player* e da *Table*, e dá-se início ao jogo. As cartas são baralhadas através do construtor do *Deck* e a partir daí as ações tomadas são lidas através dos comandos dados pelo utilizador através de teclado.

A partir da classe *Table* é feita a aposta seguida da distribuição de cartas, primeiras duas para o *dealer* e duas seguintes para o *player*. Para o decorrer do jogo também é a classe *Table* a utilizada e no fim do jogo, o método *checkWiner* indica quem foi o vencedor. O jogo pode recomeçar enquanto o *player* tiver dinheiro ou enquanto não for feito o comando quit “q”.

Ao atingir a percentagem do *shoe* que é necessária para voltar a fazer *shuffle* é chamado novamente o construtor do *Deck* para voltar a baralhar as cartas.

No modo debug, são lidos a partir dos argumentos do comando dois ficheiros, um que contém uma sequência de cartas e o outro, os comandos a executar.

Ao criar o jogador, através do construtor do *Player*, é enviado um *ArrayList*, contendo todos os comandos lidos num dos ficheiros recebidos. Por sua vez, o construtor da *Table*, recebe uma string que contem o baralho, lido no outro ficheiro recebido. Esta string será lida e guardada no *shoe*, com recurso ao construtor do *Deck*.

O jogo é iniciado e jogado da mesma forma que no modo interativo com a principal diferença de que as cartas distribuídas são as contidas no ficheiro shoe, (sem que sejam baralhadas) e os comandos lidos a partir do ArrayList.

No modo de simulação, o programa testa a estratégia, sendo lhe dado o número de baralhos, a percentagem de *shuffle*, o número de *shuffles* e a ou as estratégias a utilizar através do comando lido no terminal. Neste caso o programa funciona apenas com recurso às estratégias implementadas nas respetivas classes e geram os comandos que serão utilizados na resolução do jogo.

1. **Interface Gráfica**

No modo de jogo interativo, seria criada uma interface gráfica para o utilizador, ou seja, aqui seriam dadas as informações da aposta, mostradas as cartas de forma escrita e dados os comandos de jogo pretendidos. Esta interface foi criada e é apresentada na *figura 1*, no entanto não foi junta ao código e implementada, acima de tudo por falta de tempo.

Esta teria um cabeçalho com o nome *Blackjack* de modo a indicar o jogo, uma *JTextArea* que recebe o valor da aposta e dois *JLabel*, um com o nome *My Bet* que é atualizado aquando uma nova aposta é feita e o outro com o nome *Balance* que tem o dinheiro do utilizador e que é atualizado durante o jogo. Contém também oito *JButton*, o primeiro para carregar a aposta chamado *Bet*, os três seguintes, *Hit*, *Stand* e *Quit* que realizam as ações definidas pelo utilizador de pedir mais uma carta, parar de pedir cartas ou fechar o programa respetivamente. Neste último caso são apresentadas as estatísticas. Os últimos quatro são utilizados para a implementação das *side rules* possíveis, *Double*, *Surrender*, *Insurance* e *Split*. De modo a ver o jogo existem duas *JLabel* que são criadas com os nomes, *Dealer*, *Player.* Seria suposto que ao iniciar o jogo, a *JLabel* do *Player* fosse editada para passar a mostrar o nome do jogador e após a aposta de modo a visualizar as cartas entregues, seriam criadas outras duas *JLabel* que iriam conter respetivamente, as cartas do *dealer*, as cartas do *player*, (utilizador). Os *Advices* e as estatísticas podem ser pedidas através de outros dois *JButton* criados, Advices e *Statistics*, ou no caso das estatísticas, estas aparecem no fim do jogo numa *JLabel* criada nessa altura para esse efeito.

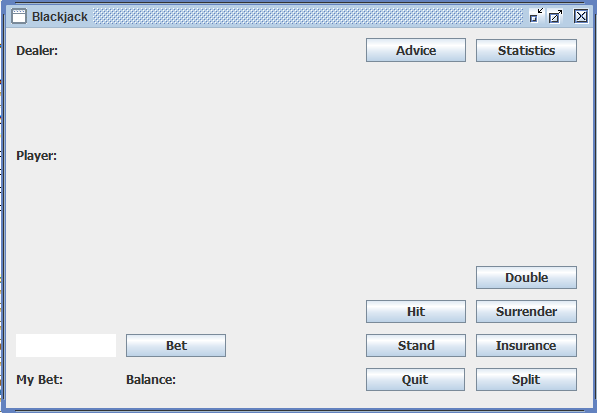


Figura 1 – Desenho da Interface Gráfica

1. **Conclusões**

Por fim, é possível verificar que foi conseguida a implementação de particamente todas as funcionalidades do programa. Foi observado apenas um erro no modo de jogo *Simulation* no caso de existirem já quatro mãos criadas sendo que uma delas tem duas cartas com o mesmo valor, seguindo os métodos de contagem deveria ser feito *Split*. No entanto, pelas regras do projeto a execução do *Split* numa situação destas, (existência de quatro mãos), não é possível o que faz com que o programa bloqueie.

Para testar todo o programa foram testados os exemplos fornecidos através da página da disciplina na secção Input/output exemples através dos quais é possível concluir a boa resolução do problema no sentido em que foram verificados resultados coerentes com os dados apresentados.

No teste do modo Debug com a utilização do exemplo 2, os resultados obtidos foram iguais aos indicados. No entanto não foi criada uma mensagem de erro quando o ficheiro de comandos deixa de ter comandos para ler. Assim o eclipse indica um erro por não encontrar o comando seguinte.

A partir do modo *Simulation* foi possível analisar os diferentes métodos de contagem, podendo concluir-se a partir dos testes feitos que a mais eficaz é a *Hi-lo* juntamente com a *Ace-five*.

No modo *Interactiv*, apesar de todas as funcionalidades estarem funcionais, os *advices* foram apenas definidos para serem dados de acordo com a *Basic Strategy*, portanto, apesar de funcionarem é apenas dado o *advice* segundo esta estratégia.

Como foi referido anteriormente, apesar de ter sido desenvolvida uma interface gráfica simples, esta não foi implementada juntamente com o código do programa, logo não é possível que seja utilizada.