Instituto Superior Técnico

MEEC

Programação Orientada por Objetos

Math in Casinos - Video Poker

Alunos:	*Número*
Pedro Martinho	78141
Rui Figueiredo	78247
Alexandre Candeias	78599

13 de Maio de 2017

1 Introdução

O presente relatório tem como objetivo descrever os detalhes de implementação do projeto *Math in Casinos - Video Poker* realizado no âmbito da cadeira de Programação Orientada a Objetos. Tentou tornar-se o código o mais extensível e aberto possível e usar o máximo das primitivas da linguagem Java.

Todos os *packets* e respetivas classes encontram-se devidamente comentados tendo sido gerado o *javadoc*. Além disso, efetuamos 5 testes de Debug diferentes para verificar o correto funcionamento do jogo.

Por fim, efetuámos alguns jogos exaustivos de forma a podermos obter resultados que nos permitam concluir o correto funcionamento do jogo e do respetivo *advice*.

2 Organização Geral

O problema foi separado em quatro componentes principais, dando lugar aos quatro *packages* disponibilizados. Temos um *package* que implementa classes relacionados com as cartas, onde temos uma classe *Cards* e *Deck* além de muitas outras. Este package pode, por exemplo, ser utilizado num outro jogo também sejam utilizadas cartas.

Além disso, disponibilizamos também um *package* onde temos as interfaces de uma máquina de Videopoker, bem como a interface da variante pretendida *VideoPoker-Variation*. Disponibilizamos também classes que servem para dar suporte á comunicação entre a maquina de videopoker que implementa as interfaces anteriores, e o próprio *player*.

Temos o *package player* onde são disponibilizados vários tipos de jogador, que utilizam um videopooker implementado pela classe apresentada anteriormente. Implementámos os jogadores: automáticos, iterativo e de debug a partir de um ficheiro de comandos e outro de cartas. Considerando o jogador iterativo, este pode interagir com o jogo a partir de uma *GUI* ou da própria linha de comandos.

Por fim, através do *package ourvideopoker* implementámos uma máquina de videopoker, que respeita a interface Videopoker, e também uma variante do videopoker chamada *Double Bonus 10/7*.

Apresentamos na figura 1 a máquina de estados da nossa implementação do videopoker presente na classe OurVideoPoker de modo a entender melhor o funcionamento da mesma. É de salientar que o que possibilita a transição de um estado para outro são os diferentes métodos da classe.

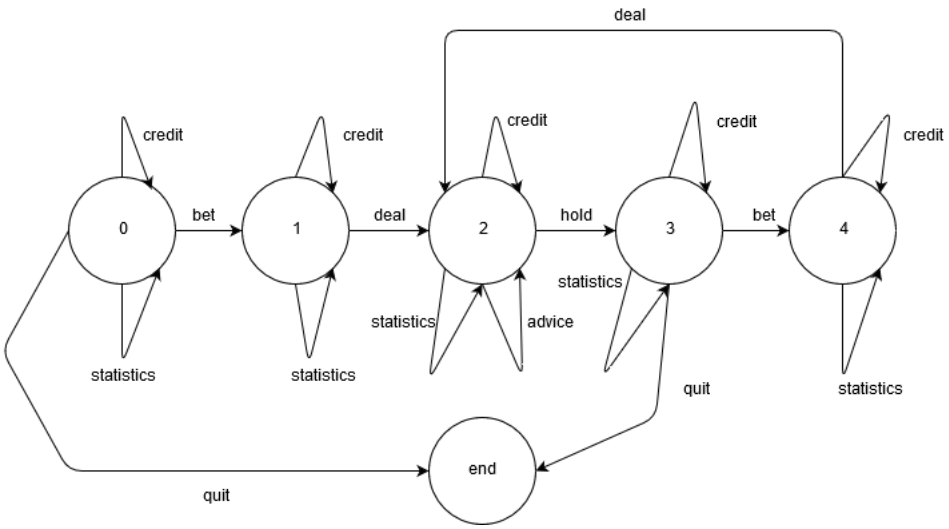

Figura 1: Máquina estados classe OurVideoPoker

Deste modo o nosso programa tem muitos pontos em que pode ser estendido. Por exemplo, se quisermos implementar um outro modo de jogo basta construir outra classe que represente esse modo de jogo que implemente a interface VideoPokerVariation e podemos assim utilizar a nossa implementação da interface *VideoPoker* para jogar esta variante.

Se quisermos uma nova interface do jogo com o jogador, onde pretendemos mudar as mensagens que são apresentadas ao jogador, basta estender a classe *Player* ou uma das suas subclasses e realizar as alterações necessárias.

3 Testes

3.1 Ficheiros de Teste

3.1.1 Teste 1

O primeiro teste tem como intuito testar os *advices*. Para tal, construimos um ficheiro em que acontecem todos os casos de exemplo descritos na secção 2 do enunciado. Verificou-se que todos os conselhos dados pelo programa neste modo vão de acordo com o resultado esperado, por isso conclui-se que o *advice* tem o comportamento desejado.

3.1.2 Teste 2

O segundo teste tem como objetivo testar a mão final do jogador. Ou seja, pretendemos verificar se o programa classifica com sucesso ou não uma mão e se o seu pagamento é o correto. É efetuado um teste para cada uma das mãos principais para cada um dos valores de *bet* possíveis. Foi verificado que os *payouts* vão de acordo com os previstos na tabela apresentada na secção 1.3 do enunciado e que a mão é corretamente classificada.

3.1.3 Teste 3

O terceiro teste serviu para testar o funcionamento do Modo de Debug quando não existam cartas suficientes para complementar todos os comandos desejados. Foi verificado que o programa termina subtilmente, informando o utilizador que não existem mais cardas disponíveis para serem jogadas no ficheiro.

3.1.4 Teste 4

O quarto teste serviu para testar a *máquina de estados*. Com isto pretendemos verificar se o programa apenas aceita os comandos permitidos num determinado ponto do jogo. Por exemplo, verificar que o jogador só pode fazer *advice* depois de se ter feito *deal*, entre outras situações. Verificamos que de facto os comandos aceites vão de acordo com a *máquina de estados* apresentada na figura 1.

3.1.5 Teste 5

O quinto teste teve como intuito testar o comportamento do programa quando o jogador fica sem créditos. Para isso criámos um ficheiro de comandos com várias jogadas em que faz *hold* de todas as cartas e um ficheiro de cartas com várias mão vazias. Após várias iterações o jogador eventualmente perde todo o crédito e verificámos que o programa é terminado subtilmente informando o jogador que não tem mais créditos para jogar.

3.2 Retorno Real

De modo a testar a qualidade do *advice* desenvolvido construiu-se um *script* que executava o jogo cem vezes realizando em cada vez cem mil jogadas e calculava-se o retorno médio dessas experiências os resultados obtidos encontram-se na tabela (1). Os resultados são um pouco a baixo nos apresentados no enunciado do projeto mas isto deve-se ao facto de a estratégia implementada não ser exatamente a melhor estratégia mas sim uma perto da melhor estratégia.

Bet	Pratical Return
1	98.69
2	97.30
3	96.13
4	94.50
5	98.72

Tabela 1: Retornos médios no modo de simulação

Foi também efetuada uma comparação entre o número de mãos de cada um dos tipos resultante do nosso programa e a informação presente no site ***wizardofodds.com***. Para isso, realizamos um jogo em que foram efetuadas um milhão de jogadas com a melhor estratégia possível. Como podemos verificar, os resultados são muito próximos.

Hand	Nb
Jacks or Better	190214
Two Pair	125260
Three of a Kind	72979
Straight	13920
Flush	15182
Full House	11124
Four of a Kind	2358
Straight Flush	97
Royal Flush	22
Other	568844
Total	1000000
Credit	1008575 (100.86)

Figura 2: Tabela de estatísticas para uma simulação com um milhão de jogadas

4 Conclusão

Existe uma parte da nossa implementação do Videopoker, presente no *package* ourvideopoker que poderia ser mais genérica, a parte referente ás estatísticas recolhidas na máquina ao longo do jogo. O principal problema é que estas estatísticas deveriam ser genéricas, isto é deviam de ser de acordo com a *payout table* da variação que estamos a jogar. Isto seria fácil de fazer usando *Hash Maps* da linguagem java de acordo com os nomes das mãos provenientes da class que implementa a interface VidoPokerVariation. Tal não foi feito pois no enunciado era pedido explicitamente uma tabela de mãos que não era a de acordo com a *payout table* da variação *Double Bonus 10/7*.

Uma outra coisa que teria sido interessante de implementar era outra estratégia de conselhos a dar ao utilizador do videopoker. Uma estratégia probabilística com base na maximização do valor esperado de ganho dada a mão do jogador. Com o *software* implementado para este projeto era facilmente implementada uma estratégia deste tipo, para isso poderíamos estender a classe que implementa a variação *Double Bonus 10/7* e redefinir apenas o método *advice*. Depois bastaria em vez de passarmos como argumento um objeto do tipo anterior e iríamos ter um videopoker que dava concelhos ao jogador de acordo com esta nova estratégia.

Com a realização deste trabalho foi possível aumentar a compreensão da importância de deixar o código aberto e possível de ser estendido e as formas mais adequadas de permitir esta extensão.