Instituto Superior Técnico

Programação Orientada por Objetos

BlackJack

Relatório

Grupo 16:
Inês Ferreira 90395
Ricardo Santos 90178
Tomás Bessa 90200

2020/2021 - 2º Semestre
24.05.2021

Introdução

O BlackJack é um jogo que usualmente se joga no casino, sendo particularmente reconhecido como um dos seus principais jogos. O objetivo deste jogo é derrotar a mesa e o seu dealer num "frente a frente"entre duas mãos de cartas de modo a perfazer o 21 ou o mais próximo dos valores inferiores a 21, sendo que se verifique maior que o resultado obtido pela mesa.

O objetivo do projeto realizado no âmbito da disciplina de Programação Orientada por Objetos é desenvolver um programa que simule partidas de BlackJack em 3 modos diferentes - *Interactive*, *Debug*, e *Simulation*.

O seguinte relatório aborda, primeiramente, aspetos importantes relativamente ao UML do projeto, uma ferramenta que permitiu esquematizar, *a posteriori*, toda a implementação do código. Seguidamente, são cobertos alguns exemplos ilustrativos de casos relevantes que devem serem analisados. Em terceiro lugar, são analisados os resultados obtidos através do modo *Simulation* e, por último, são apresentadas algumas conclusões e possíveis melhorias na implementação.

Notas sobre o UML e Implementação

O UML é o esqueleto de todo o projeto, traduzindo a sua organização e a forma como o trabalho é implementado. Seguem-se algumas notas relevantes sobre o UML desenvolvido:

- O atributo *isSoft* foi adicionado à classe *Card* por duas razões. Primeiro, facilita o cálculo do valor de uma mão sempre que se adicionam cartas. Segundo, permite saber, ao usar a estratégia de contagem de cartas *Basic Strategy*, se devemos utilizar a tabela *soft* ou a tabela *hard*.

- Decidiu não se utilizar uma classe *Deck* porque, nesse caso, a classe *Shoe*, sendo associada a vários *Deck*, teria como atributo um *ArrayList* de *Deck*. Para podermos utilizar o método *shuffle* do *Java Collections* para baralhar as cartas de todos os baralhos de forma conjunta, é necessário que o *Shoe* contenha um *ArrayList* de *Card* e não de *Deck*.

- A subclasse *PlayerHand* da classe *Hand* foi criada, dado que existem algumas especificidades relativas à mão do jogador que não existem na mão do *dealer* devido, entre outras razões, às *side rules*.

- De forma a tornar o código extensível a futuras mudanças, criaram-se duas classes abstratas correspondentes às estratégias das apostas e às estratégias das jogadas. Assim, é possível adicionar novas estratégias para além daquelas que estão implementadas.

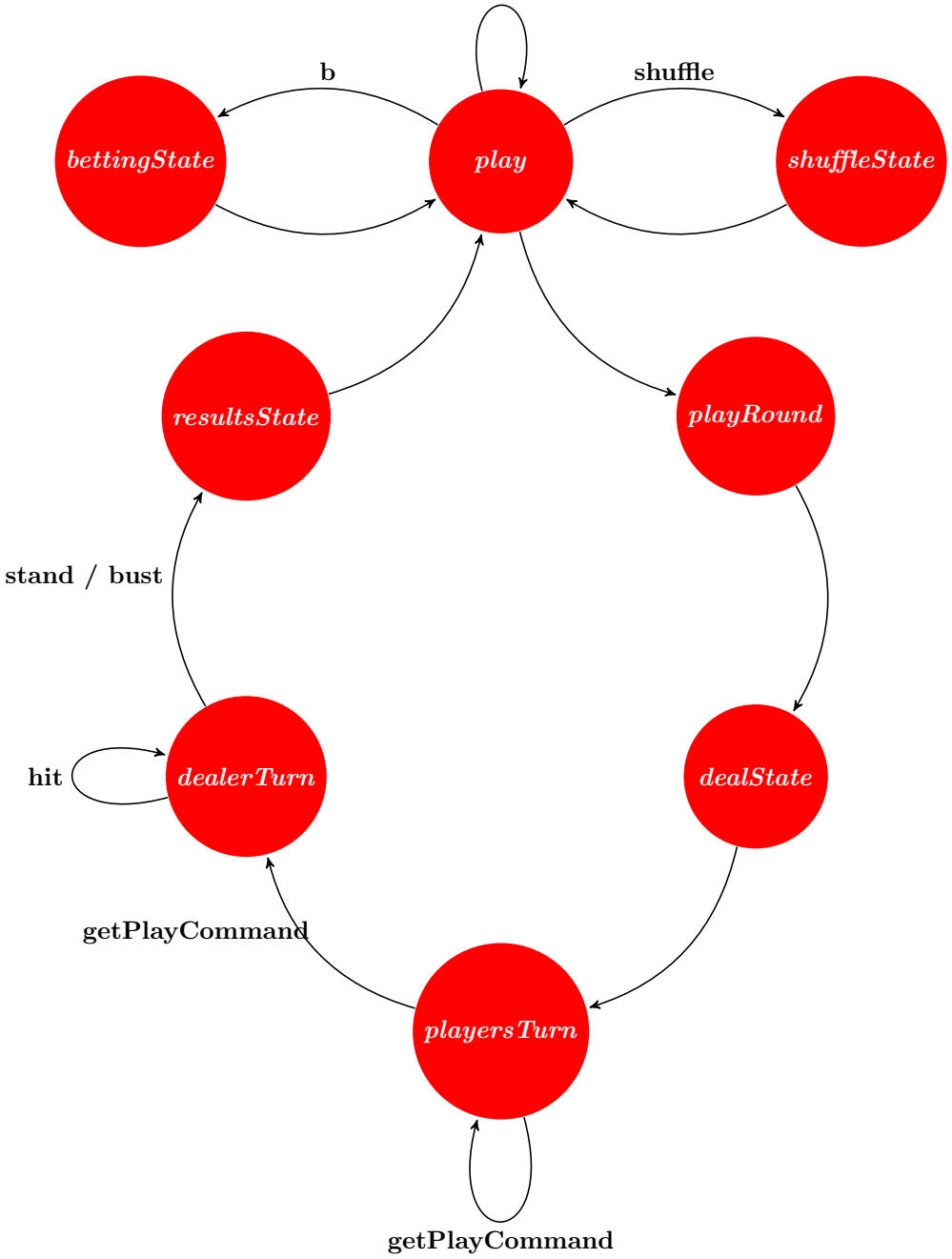
- Na tabela *soft* da estratégia *Basic*, note-se que existem vários valores da mão do jogador para os quais é recomendado fazer *Double Down*. No entanto, na implementação que nos foi pedida, apenas jogadores com mãos num determinado intervalo (9 a 11 neste caso), podem realizar este comando. Assim, de forma a que se possa alterar este intervalo no futuro dependendo das regras de cada casino, adicionaram-se dois atributos, o *DDmin* e *DDmax*. Estes definem quando é possível realizar *Double Down*.

- A existência dos três modos de jogo requer que os comandos correspondentes às apostas e jogadas sejam recebidos / calculados de forma diferente: através das estratégias, vindo do terminal ou de ficheiros. Decidiu então implementar-se uma interface com três métodos diferentes: dois referentes a receber o comando com a próxima jogada e um terceiro que recebe o comando respetivo à aposta. A criação desta interface também permite que se implementem novos jogos mais facilmente.

- Sobre o modo de jogo *Simulation* é importante referir que, para retornar os comandos de forma automática, é preciso que ele tenha acesso às estratégias de apostas e de contagem de cartas, sendo necessária uma associação entre esta classe e as estratégias. No entanto, como queremos manter a atualização das contagens e das apostas na classe *Game* (tal como para os outros modos de jogo), passaram-se por referência as estratégias que estão associadas ao *Game* para o modo *Simulation*. Desta forma, alterações nos atributos das estratégias associadas ao *Game* vão ser replicadas na simulação.

Máquina de estados

Na classe *Game* temos o método *play* que representa o jogo em si. Abaixo está apresentada a máquina de estados que é implementada. Note-se que os estados correspondem a vários métodos desta classe.



Exemplos de Interesse

Exemplo 1

Os ficheiros de exemplo "cmd-file1.txt"e "shoe-file1.txt"ilustram o comportamento da mistura entre as *side rules* "*split*"e "*double down*".

```
# b
player is betting 5
# d
dealer's hand 4S X
player's hand 2D 2H (4)
# p
player is splitting
playing 1st hand...
player's hand [1] 2D 7D (9)
# 2
player's hand [1] 2D 7D QC (19)
playing 2nd hand...
player's hand [2] 2H 5S (7)
# 2
2: illegal command
# q
bye
```

Como podemos ver, é possível fazer *double down* de uma mão que vale 9 e automaticamente o jogador recebe uma e uma só carta e avança para a mão seguinte. Assim, esta operação é equivalente a um *hit* e um *stand*. Se a mão for diferente de 9, 10 ou 11 (neste caso a segunda mão é 7), não é possível realizar o comando.

Exemplo 2

Os ficheiros de exemplo "cmd-file2.txt"e "shoe-file2.txt"ilustram um caso em que, como o jogador perdeu em todas as suas mãos, não vale a pena o *dealer* fazer a sua jogada e a rodada acaba automaticamente.

```
# b
player is betting 5
# d
dealer's hand 2S X
player's hand 8D 8C (16)
# p
player is splitting
playing 1st hand...
player's hand [1] 8D 8H (16)
# p
player is splitting
playing 1st hand...
player's hand [1] 8D 9H (17)
# h
player hits
player's hand [1] 8D 9H JD (27)
player busts [1]
playing 2nd hand...
player's hand [2] 8H 5S (13)
# h
player hits
player's hand [2] 8H 5S QC (23)
player busts [2]
playing 3rd hand...
```

```
player's hand [3] 8C 4S (12)
# h
player hits
player's hand [3] 8C 4S KC (22)
player busts [3]
dealer's hand 2S 7H (9)
dealer stands
Player loses [1] and his current balance is 485.0
Player loses [2] and his current balance is 485.0
Player loses [3] and his current balance is 485.0

# q
bye
```

Exemplo 3

Os ficheiros de exemplo "cmd-file3.txt"e "shoe-file3.txt"ilustram um caso em que é ilegal realizar *split*.

```
# b
player is betting 5
# d
dealer's hand JS X
player's hand JD QC (20)
# p
player is splitting
playing 1st hand...
player's hand [1] JD 5C (15)
# s
player stands [1]
playing 2nd hand...
player's hand [2] QC KS (20)
# p
player is splitting
playing 2nd hand...
player's hand [2] QC 8D (18)
# s
player stands [2]
playing 3rd hand...
player's hand [3] KS 10H (20)
# p
player is splitting
playing 3rd hand...
player's hand [3] KS JC (20)
# p
p: illegal command
# q
bye
```

Ao jogar a terceira de quatro mãos, não é possível fazer *split* como esperado, mesmo que todas as outras mãos já tenham sido jogadas.

Exemplo 4

Os ficheiros de exemplo "cmd-file4.txt"e "shoe-file4.txt"ilustram um caso onde se realiza *split* de ases.

```
# b
player is betting 5
# d
```

```
dealer's hand 2D X
player's hand AC AD (12)
# p
player is splitting
playing 1st hand...
player's hand [1] AC 5C (16)
playing 2nd hand...
player's hand [2] AD AS (12)
# h
h: illegal command
# p
player is splitting
playing 2nd hand...
player's hand [2] AD 4D (15)
playing 3rd hand...
player's hand [3] AS 5H (16)
dealer's hand 2D JH (12)
dealer hits
dealer's hand 2D JH 3S (15)
dealer hits
dealer's hand 2D JH 3S QC (25)
dealer busts
Player wins [1] and his current balance is 495.0
Player wins [2] and his current balance is 505.0
Player wins [3] and his current balance is 515.0

# q
bye
```

Quando se executa um *split* de ases podem ocorrer dois casos: ou a segunda carta da mão resultante é um ás e o jogador pode fazer *stand* ou *split*, ou a segunda carta não é um ás, sendo a única opção do jogador fazer *stand*. Assim, decidimos implementar esta opção automaticamente para este último caso. Neste exemplo começamos então por realizar um *split* de ases, após o qual a primeira mão resultante não é um par de ases, sendo o jogador obrigado a fazer *stand*. Já na segunda mão mostramos que é necessário obter um *input* do jogador, mas que a opção *hit* não é válida já que, após um *split* de ases, só se pode obter mais uma carta.

Exemplo 5

Os ficheiros de exemplo "cmd-file5.txt"e "shoe-file5.txt"ilustram momentos em que o jogador pode ou não fazer *insurance*.

```
# b
player is betting 5
# d
dealer's hand AS X
player's hand 2C 2D (4)
# i
player is insuring
# p
player is splitting
playing 1st hand...
player's hand [1] 2C 5H (7)
# i
i: illegal command
# u
player is surrendering [1]
playing 2nd hand...
```

```
player's hand [2] 2D 8C (10)
# h
player hits
player's hand [2] 2D 8C 7C (17)
# u
u: illegal command
# q
bye
```

O jogador começa por realizar *insurance* na *opening hand*. No entanto, depois de realizar *split* já não é possível fazer o mesmo, ainda que a mão apenas tenha duas cartas. Pode-se no entanto realizar *surrender*. Após acabar as jogadas desta mão e avançar para a seguinte, note-se que já não é possível fazer *surrender* após um *hit*.

Resultados

% até Shuffle	Estratégias	% BlackJack - Jogador	% BlackJack - Dealer	% Vitórias	% Derrotas	% Empates	Saldo Final	Saldo Final / Saldo inicial
20	BS	5	6	41	54	5	518	1.03
		3	9	48	47	5	480	0.96
		4	3	45	51	4	440	0.88
		1	5	26	66	7	208	0.41
		10	10	48	43	9	813	1.62
	BS-AF	4	4	56	38	5	1063	2.12
		7	2	41	51	7	935	1.87
		2	9	43	50	7	935	1.87
		4	3	33	53	14	890	1.78
		5	5	37	57	6	875	1.75
	HL	7	3	42	48	10	255	0.51
		4	3	36	57	7	255	0.51
		3	2	51	43	6	658	1.31
		5	3	43	48	9	443	0.89
		3	4	40	41	18	413	0.82
	HL-AF	3	4	51	44	6	615	1.23
		4	5	39	54	7	340	0.68
		7	5	54	42	4	500	1
		11	7	46	47	8	693	1.39
		8	7	47	45	7	550	1.1

Tabela 1: Resultados para cinco iterações de cada estratégia no modo *Simulation* utilizando 5 como aposta mínima, 50 como aposta máxima, 500 como saldo inicial, 4 como número de baralhos, 20 como percentagem do *shoe* que sai antes de fazer *shuffle* e 10 como o número de baralhos que sai antes da simulação acabar.

Na tabela acima temos cinco iterações do modo *Simulation* com cada uma das combinações de estratégias. É fácil de notar que a estratégia *Basic + Ace5* é a única onde o rácio entre o saldo final e o saldo inicial é sempre superior a 1. A estratégia com o rácio mais baixo é a estratégia *HiLo*.

% até Shuffle	Estratégias	% BlackJack - Jogador	% BlackJack - Dealer	% Vitórias	% Derrotas	% Empates	Saldo Final	Saldo Final / Saldo inicial
80	BS	5	6	44	49	8	100	0.2
		5	3	41	51	8	62.5	0.12
		5	5	40	52	8	-220	0.44
		5	6	41	51	8	403	0.81
		6	4	44	48	8	438	0.88
	BS-AF	5	5	45	48	6	2335	4.67
		6	6	46	45	9	2355	4.71
		6	6	41	52	7	2173	4.34
		7	3	42	48	1	2260	4.52
		3	5	39	54	8	2095	4.19
	HL	6	6	43	50	7	2200	4.4
		5	6	43	49	9	195	0.39
		6	3	46	47	7	1240	2.48
		5	4	46	48	7	168	0.34
		5	6	43	50	7	-85	-0.17
	HL-AF	3	5	48	44	8	1220	2.44
		4	4	45	48	7	613	1.23
		4	4	44	49	8	443	0.89
		7	6	46	46	9	1230	2.46
		3	6	40	53	7	-170	-0.34

Tabela 2: Resultados para cinco iterações de cada estratégia no modo *Simulation* utilizando 5 como aposta mínima, 50 como aposta máxima, 500 como saldo inicial, 4 como número de baralhos, 80 como percentagem do *shoe* que sai antes de fazer *shuffle* e 10 como o número de baralhos que sai antes da simulação acabar.

Na tabela acima temos uma nova versão com uma maior percentagem do *shoe* antes de fazer *shuffle*. Note-se que todos os valores do rácio da estratégia *Basic* são inferiores a 1. Para as duas estratégias *HiLo*, os rácios são bastante díspares, sendo até alguns deles negativos. A estratégia *Basic + Ace5* continua a ser a melhor, sendo os rácios, no mínimo, duas vezes mais elevados do que na tabela anterior. Isto era esperado visto que, com uma maior percentagem do *shoe* antes de fazer *shuffle*, é mais fácil prever que cartas faltam sair. Conclui-se então que a melhor estratégia é a combinação de *Basic* com *Ace5*.

Conclusão e Melhorias Futuras

A implementação do projeto foi bem sucedida mas existe espaço para certas melhorias. Nomeadamente seria interessante estender o jogo a vários jogadores, ter uma versão sem *side rules* e implementar mais estratégias de contagens de cartas, quer para calcular jogadas, quer para calcular o valor ótimo das apostas seguintes. A parte correspondente à máquina de estados pode sem dúvida ser melhorada, especialmente se quisermos adicionar mais jogadores.