Instituto Superior Técnico

Programação Orientada por Objectos

Relatório - Blackjack

Autores

Nome: Carlos Cardoso **Número:** 87161
Nome: Luís Coelho **Número:** 90127

Grupo 25
Prof. Alexandra de Carvalho

2020/2021 - 2º Semestre
24 de Maio 2021

1 Introdução

O Blackjack é um jogo de casino bastante popular internacionalmente. Este trabalho tem como objetivo implementar a simulação deste mesmo jogo. Esta simulação permite a um utilizador deste programa que jogue o jogo normalmente, com ou sem sugestões do computador, através do modo de Interface. A mesma permite também realizar a análise de determinadas estratégias de jogo tanto a nível de estratégias de betting, nomeadamente a "Standard Betting Strategy"e a "Ace-Five Strategy", como a nível de escolha de ações, nomeadamente a "Basic Strategy"e a "Hi-Lo Strategy". Esta análise de estratégias é possível graças ao modo de Simulação implementado. Também no âmbito deste trabalho foi implementado um modo de Debugging que é utilizado para essa mesma função. De modo a realizar este projeto foi então utilizada a linguagem de programação orientada por objetos *Java* em conjunto com o planeamento do mesmo através do desenvolvimento de diagramas UML (Unified Modelling Language).

2 State Design Pattern

Durante o jogo de Blackjack o jogador tem várias ações possíveis, sendo que algumas destas só estão disponíveis em diferentes fases de cada jogada. Isto torna uma implementação através de uma máquina de estados pouco prática pois a existência de vários estados dependentes de diferentes condições levaria a métodos repletos de "if-else statements"o que levaria a um código difícil de manter.

Por forma a contornar estas adversidades este projeto foi implementado com base no "State Pattern Design".

O State é um padrão de projeto de software maioritariamente utilizado quando, dependendo do seu estado, o comportamento de um objeto muda, o que é ideal para este projeto.

Foram então implementados os seguintes estados: BettingState; DealingState; SidesState; SplittingState e PlayingState.

Todos estes diferentes estados correspondem a diferentes implementações da Interface GameState e seus respetivos métodos: bet; balance; deal; hit; stand; insurance; surrender; splitting; doubledown; advice e statistics.

Cada um destes métodos corresponde à implementação das diferentes ações possíveis para o jogador de Blackjack. No entanto, algumas destas ações só devem estar disponíveis em certas alturas do jogo o que realça a importância da utilização deste padrão de projeto pois em diferentes alturas do programa é necessário que os mesmos métodos apresentem comportamentos diferentes.

O **BettingState** corresponde à altura do jogo em que o jogador ainda não colocou uma aposta em jogo. Como tal, os métodos disponíveis devem ser apenas: o **advice**, para aconselhar ao utilizador um valor de aposta conforme as diferentes estratégias de betting implementadas; o **statistics** para informar o utilizador acerca das estatísticas do jogo até ao momento corrente; o balance para informar o jogador do balance disponível; e o **bet** para que o utilizador coloque uma aposta em jogo. O resto dos métodos neste estado encontram-se indisponíveis. O jogo transita do estado **BettingState** para o estado **DealingState** após o jogador apostar uma quantia válida.

O **DealingState** corresponde à altura do jogo em que o utilizador pode apenas pedir ao dealer que o mesmo distribua as cartas através do método **deal**. Neste estado o utilizador pode ainda consultar o seu balanço através do método **balance** e as estatísticas do jogo através do método **statistics**. Os restantes métodos encontram-se indisponíveis nesta fase. Após a ação de deal por parte do jogador o jogo transita do estado DealingState para o estado SidesState.

O **SidesState** corresponde à altura do jogo em que estão disponíveis as diferentes **side rules** implementadas pelos métodos: **insurance**, **surrender**, **splitting** e **doubledown**. Estes métodos estão disponíveis caso se verifiquem as condições para os mesmos. Nesta fase do jogo passam também a estar disponíveis os métodos: **hit** e **stand**. Estes implementam as ações de pedir uma carta (hit) ou terminar a jogada (stand) Ainda nesta fase estão também disponíveis os métodos: **balance**; **advice**; **statistics**.

Deste último estado o jogo pode transitar para vários estados diferentes. Pode transitar para o estado **SplittingState** caso o utilizador tome a ação de **split**. Pode também transitar para o estado **PlayingState** caso o jogador dê um **hit que não leve a um bust**. Pode ainda transitar para o estado de **BettingState** caso o jogador dê um **hit que leve a bust**, caso o jogador faça **stand** ou **surrender** terminando assim a sua jogada ou caso o mesmo faça **doubledown**. Nestes últimos quatro casos a jogada encontra-se terminada por parte do jogador e portanto o estado pode transitar para o estado inicial (**BettingState**).

O **SplittingState** corresponde à fase do jogo imediatamente a seguir à tomada de decisão por parte do jogador de fazer **split**. Neste fase é ainda possível aplicar **algumas das side rules** como o **doubledown** ou novamente o **splitting**, no caso de uma das mãos resultantes do splitting anterior constituir um par. Estão então neste estado disponíveis os métodos de doubledown, splitting, balance, advice e statistics.

Neste caso o jogo pode transitar deste estado para o **PlayingState** caso o jogador dê **hit sem dar bust**. Pode ainda transitar para o estado inicial de **BettingState** caso o jogador tenha terminado de jogar as suas mãos.

O **PlayingState** corresponde a uma fase do jogo em que para além dos comandos que retornam informação (balance, advice, statistics) estão apenas disponíveis as ações de **hit** e **stand**

O jogo transita então deste estado naturalmente para o estado inicial **BettingState** caso o jogador tenha terminado de jogar todas as suas mãos ou então transita para o estado de **SplittingState** caso o jogador tenha tido a oportunidade de fazer **split** e ainda não tenha terminado de jogar todas as suas mãos.

Uma ilustração das diferentes transições entre estados pode ser observada na Figura 1.

Por uma questão de simplificação foram omitidas as transições dos métodos para eles mesmos na utilização de comandos como advice, balance, statistics os quais nunca resultam em transições entre estados.

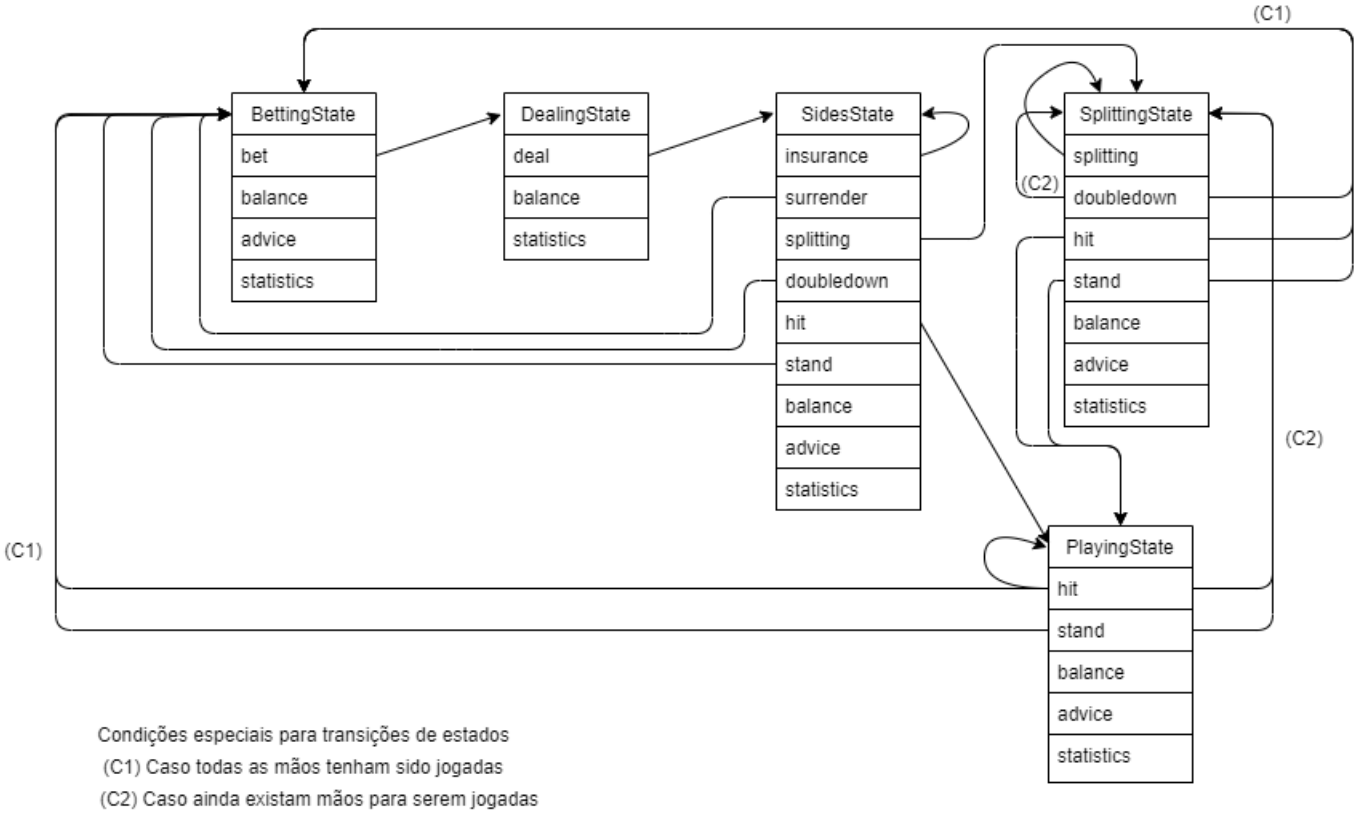

Figura 1: Ilustração de transições de estados.

3 Análise Crítica do Design

Em termos de qualidades o design deste projeto tem a seu favor o facto de ter sido implementado com base no State Design Pattern, que será muito provavelmente a forma mais viável de implementar este tipo de projeto derivado das suas características.

O mesmo também tem como qualidade a divisão por modos de jogo o que leva a facilidade de compreensão do funcionamento do projeto.

No entanto, este projeto tem alguns bugs por resolver nomeadamente a nível de verificações da possibilidade de utilização dos métodos que implementam as side rules o que origina problemas de maiores dimensões no modo de Simulação pois pode originar um ciclo infinito de tentativas de executar um comando ilegal.

Este projeto apresenta também problemas a nível de divisibilidade. Apesar de as classes que implementam a funcionalidade das cartas serem separáveis por exemplo da restante funcionalidade do jogo, criando um package para as mesmas, o mesmo não foi feito.

4 Avaliação Crítica dos Resultados

Segundo os dados recolhidos, disponíveis no ficheiro dadosBlackjack.xlsx é possível observar, apesar do número baixo de amostras, que em geral existe em média um decréscimo no balanço indicando assim a vantagem estatística para o casino quando utilizadas as estratégias de Hi-Lo e Basic por forma a escolher as ações do jogador e utilizadas as estratégias Standard e Ace-Five para a decisão ao nível de dimensão da aposta.

Seria de esperar que uma estratégia de aposta como a estratégia standard obtivesse resultados mais polarizados (ou muito negativos ou muito positivos) do que uma estratégia mais ponderada como a Ace-Five, o que acaba por se verificar através de casos isolados em que o balanço é bastante positivo e noutros em que o balanço é bastante negativo quando utilizada a estratégia Standard. Enquanto isso a estratégia Ace-Five tem resultados que embora negativos em média, o jogador não se arrisca a perder grandes quantias de uma vez só. A estratégia Standard é então caracterizada como mais agressiva do que a Ace-Five.

Em termos de estratégias de decisão das diferentes ações, o espectável seria que a estratégia Hi-Lo obtivesse melhores resultados do que a estratégia Basic, isto porque a primeira tem em conta as cartas que já sairam enquanto que a última segue uma tabela rígida.

Seria também de esperar que quanto mais "demorado"fosse o shuffle maior seria a vantagem de utilizar uma estratégia como a Hi-Lo em contraste com a estratégia Basic.

Não se verificaram no entanto grandes diferenças entre as mesmas e o mesmo deve-se ao baixo número de amostras que acaba por tornar-se inconclusivo, no entanto ao analisarmos e implementarmos as diferentes estratégias tornaram-se vísiveis as vantagens de cada uma delas e as situações onde as mesmas seriam favorecidas.

Notou-se no entanto um padrão que já era de esperar, que era o facto de que, qualquer que seja a estratégia utilizada, em média o resultado será negativo para o balanço do jogador.

5 Conclusão

Com este projeto foi possível adquirir conhecimentos acerca da utilização de ferramentas como o Visual Paradigm, de modo a realizar diagramas UML (Unified Modeling Language). O que constitui um tema bastante relevante no âmbito desta unidade curricular.

Foi também possível aprofundar conhecimentos acerca da implementação de interfaces, classes abstratas e não abstratas e respetivos métodos aumentando assim a nossa capacidade de programação orientada por objetos.

A nível mais geral em termos de desenvolvimento de software foi também positivo implementar o state pattern design.

Foi também interessante poder analisar diferentes tipos de estratégias num jogo de azar como o Blackjack o que tornou o projeto mais interessante.

Em termos de aspetos a melhorar temos o facto de não termos dividido o nosso projeto em diferentes packages e de termos ainda alguns bugs presentes no nosso código. Embora possam ser

raros e bastante específicos deveriam ter sido resolvidos a tempo.