Programação Avançada

Quatro Em Linha

2020 - 2021

José Almeida - 2019129077

Licenciatura de Engenharia Informática
23 de maio de 2021

Conteúdo

1 Introdução

Este trabalho prático consiste na implementação de uma variante do conhecido jogo "Quatro em Linha", com algumas alterações significativas que incluem, por exemplo, uma funcionalidade de undo, e a existência de peças especiais que, depois de obtidas através de minijogos, permitem eliminar toda uma coluna. Os jogos são constituídos por 2 jogadores, que podem ser humanos ou computadores, ou um de cada. É possível guardar e continuar jogos, assim como ver o replay dos 5 últimos jogos.

2 Organização Lógica

Seguindo as indicações sobre arquitetura do projeto, foram adicionados diversos packages de modo a estruturar logicamente o código. Estão organizados da seguinte forma:

- **iu** - Interfaces de utilizador implementadas. De momento, apenas inclui a interface de texto, no package **texto**;
- **utils** - Classes de utilização geral ao programa. Inclui as classes de utils;
- **logica** - Classes que realmente implementam o jogo e toda a lógica associada ao mesmo;
  - **command** - Classes associadas à implementação do padrão Command;
  - **dados** - Classes que constituem o jogo e todas as suas partes;
    - **jogadores** - Classes que implementam a lógica de jogadores;
    - **minijogos** - Classes que implementam a lógica dos minijogos;
    - **tabuleiro** - Classes que implementam o tabuleiro de jogo.
  - **estados** - Classes associadas à implementação do padrão Máquina de Estados.

3 Padrões de Programação

3.1 Máquina de Estados

Em termos de implementação, a interação UI - Jogo é feita através do design pattern Máquina de Estados. Este permite aceder a informação essencial à IU e mudar o estado do programa, sem quebrar princípios da programação orientada a objetos, como o encapsulamento.

Abaixo encontra-se o diagrama de estados que serve de base à implementação. Os estados incluídos foram definidos por se tratarem de momentos em que o input do utilizador é obrigatório para a continuação do funcionamento, ou apenas por uma questão de compreensão de acontecimentos por parte do utilizador (como, por exemplo, no caso do assisteJogada, onde espera que o utilizador vá passando jogada a jogada, para ir mostrando o tabuleiro de forma coerente e não tudo de uma vez). O esquema fala por si quanto à maior parte da informação. No entanto, as condições de passagem de estado para estado não estão representadas nele. São as seguintes:

- A transição **adicionarJogador** verifica se ainda há espaço para jogadores. Se sim, fica no mesmo estado, se não, passa para o estado **pedeDecisaoJogada**;
- A transição **jogarFicha** verifica se o jogo terminou. Se sim, passa ao estado **fimJogo**. Se não, fica no mesmo estado.
- A transição **enviarRespostaMinijogo** verifica se o minijogo já terminou. Se sim, volta ao estado **pedeDecisaoJogada**. Se não, fica no mesmo estado.
- A transição **avancar**, no estado **assisteJogada**, verifica se há mais jogadas a apresentar. Se sim, fica no mesmo estado. Se não, volta ao estado **pedeDecisaoInicio**.

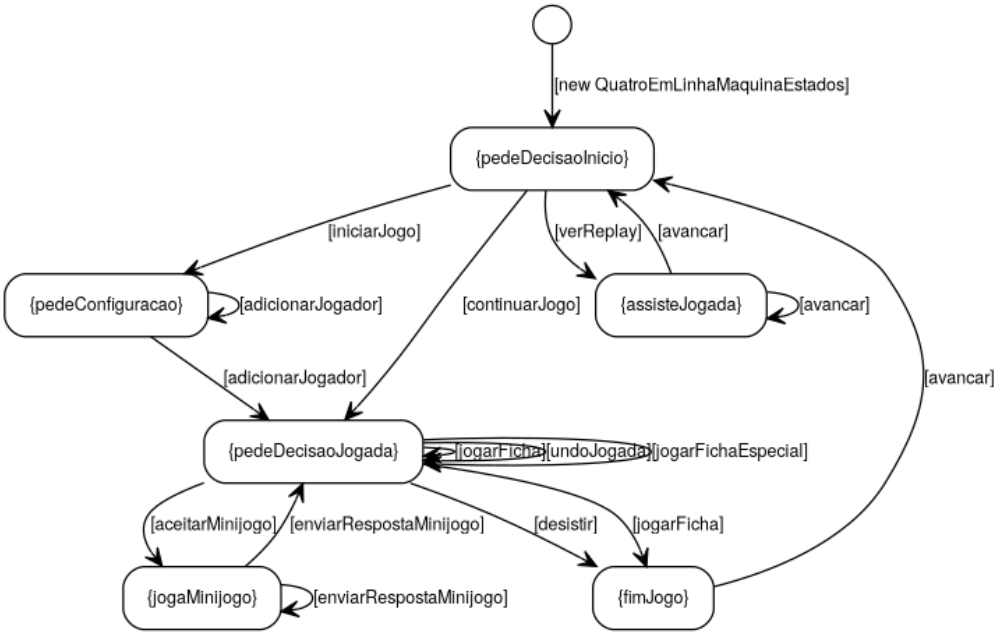

Figura 1: Diagrama de Estados

3.2 Command

Com o objetivo de implementar a funcionalidade de undo e replay, foi usado o padrão Command, embora um pouco alterado, comparativamente ao leccionado nas aulas. Neste caso, o undo é entendido como um elemento do jogo, logo é implementado também como um Command, que se limita a chamar o método undo dos comandos anteriores. A escolha de ser usado o Command e não o Memento (ou qualquer outro padrão) baseou-se no facto de que o Command permite uma maior perceção das mudanças ao jogo e uma redução do custo de memória. Neste caso, não foram encontradas desvantagens ao uso do Command.

O Command, nesta implementação, tem a dupla funcionalidade de suportar o undo de jogadas e o replay de jogos. No caso de se querer fazer undo a jogadas, é chamado o comando DesfazerJogadas, que desfaz um certo número dado de jogadas feitas.

Associado ao CommandManager, há um histórico completo de comandos executados - que inclui comandos de undo e derrotas/vitórias de minijogos. Quando se chega ao fim de um jogo, é guardado este CommandManager num ficheiro (cujo nome depende da data e hora do sistema). Para visualizar um replay, basta carregar este CommandManager e ir executando, um a um, num tabuleiro vazio, os comandos do histórico completo. Simples e intuitivo, certo?

Para além da informação específica a cada comando, todos eles incluem um toString com informação associada ao acontecimento em questão. Isto faz com que seja possível descrever ao utilizador, por texto, durante o replay, o que aconteceu no jogo, em vez de só mostrar visualmente.

3.3 Factory

Foi também usado o padrão Factory. Este padrão abstrai a criação de certos objetos, simplificando o código onde for usado. Neste caso, foi utilizado em duas situações distintas: a criação de jogadores e a escolha dos minijogos. A sua implementação verifica-se na classe **JogadorFactory**, onde devolve o jogador criado mediante o valor da enum **TipoJogador** dado, assim como classe **MinijogoFactory**, que trata de devolver, alternadamente, o minijogo a ser jogado.

4 Classes

Ao todo, foram usadas 33 classes, de entre as quais 3 são abstratas, 4 interfaces e 3 enums. Dessas, as mais relevantes são as seguintes:

- Interface **Estado** - Representa, de forma abstrata, os estados possíveis da máquina de estados. É implementada pela classe **EstadoAdapter**, que por sua vez é extendida pelas classes **AssisteJogada**, **FimJogo**, **JogaMinijogo**, **PedeConfiguracao**, **PedeDecisaoInicio**, **PedeDecisaoJogada**. Cada estado está também definido na enum **Situacao**;
- Interface **Minijogo** - Representa, de forma abstrata, os minijogos possíveis. É implementada pela classe MinijogoAdapter, que por sua vez é extendida pelas classes **Calculos** (jogo dos cálculos referido no enunciado) e **Palavras** (jogo das palavras);
- Interface **Jogador** - Representa, de forma abstrata, um jogador. É implementada pela classe **JogadorAdapter**, que é posteriormente extendida pela classe **Humano** (jogador humano) e **Computador** (jogador que joga de forma autónoma). Cada tipo de jogador está definido na enum **TipoJogador**;
- Interface **Command** - Representa, de forma abstrata, um comando a executar, disponibilizando, também, caso seja necessário, o seu undo. É implementada pela classe **CommandAdapter**, que é extendida pelas classes **AdicionaFichaEspecial** (receber uma ficha especial devido a ter ganho um minijogo), **DesfazerJogadasCommand** (executa o undo de outros comandos), **DesistirCommand** (desistir do jogo), **JogarFichaCommand** (adicionar uma ficha numa coluna do tabuleiro), **JogarFichaEspecialCommand** e **PerderMinijogoCommand** (informação relativa à derrota num minijogo). Certas classes não têm undo possível, pois servem apenas para adicionar ao histórico completo de modo a possibilitar o replay;
- Classe **QuatroEmLinha** - Representa o jogo em si. Tem todos os métodos associados ao mesmo, como métodos para jogar uma ficha numa determinada coluna, ou adicionar um jogador. É constituída por dois objetos também essenciais à aplicação: um objeto da classe **Tabuleiro**, que trata de tudo o que tem a ver com a gestão do tabuleiro de jogo, e a classe **ListaJogador**, que gere a lista de jogadores a participar no jogo;
- Classe **CommandManager** - Recebe, executa e reverte comandos que lhe são enviados. Mantém um histórico destes comandos, servindo de base à implementação das funcionalidades de undo e replay;
- Classe **QuatroEmLinhaGestor** - Abstrai a interação com as classes **QuatroEmLinha** e **CommandManager**, controlando o que é executado diretamente e o que é executado através do Command;
- Classe **QuatroEmLinhaMaquinaEstados** - Serve de camada de interação entre as classes de IU e a classe **QuatroEmLinhaGestor**. É composta por um objeto do tipo **Estado**, e controla a interação com ele;
- Classe **QuatroEmLinhaMaquinaUITexto** - Classe que trata de toda a interação com o utilizador.

Embora já descrita acima, a relação entre as classes existentes é mais facilmente explicada pelo esquema presente na próxima página.

Nota: As classes utilitárias e as enums não se encontram no esquema, devido a serem públicas e acessíveis em todo o programa.

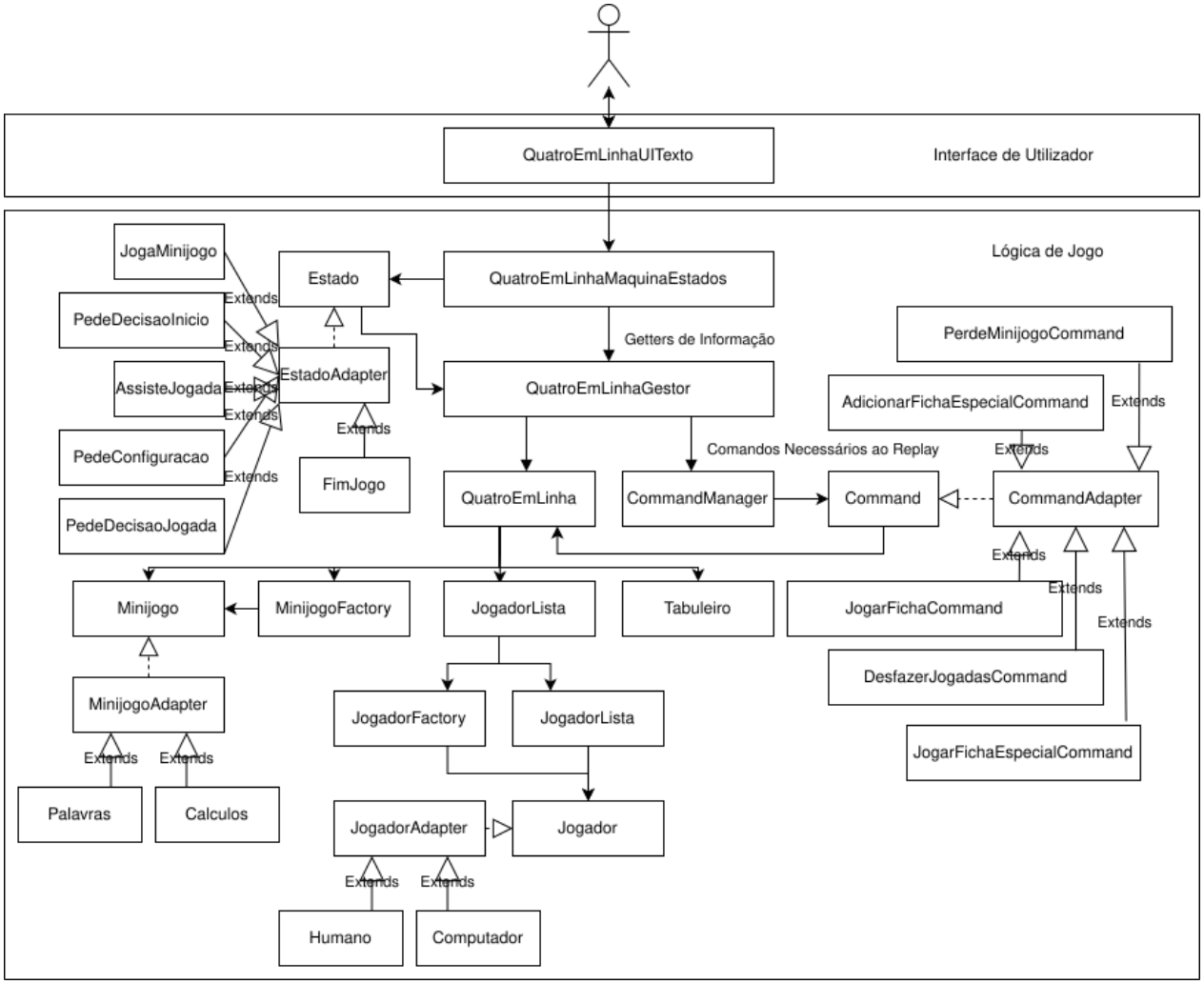

Figura 2: Diagrama de Classes

5 Funcionalidades Implementadas

Componente do Trabalho	Realizado	Realizado parcialmente	Não realizado
Organização ficheiros / classes	✓		
Nome e Tipo dos Jogadores	✓		
Quatro em Linha Tradicional	✓		
Minijogos e Peça Especial	✓		
Replay dos 5 Últimos jogos	✓		
Save e Load de Jogos a Decorrer	✓		
Undos e Sistema de Créditos	✓		

6 Anexos

Lista de Figuras