Programação Avançada

4 em Linha

2020 - 2021

TheForgotten

Licenciatura de Engenharia Informática
23 de maio de 2021

Conteúdo

1 Introdução

O trabalho prático consiste na implementação de um jogo em Java.

O jogo é o 4 em linha. Neste, dois jogadores tentam alinhar 4 das suas peças na vertical, horizontal ou diagonal. O primeiro a fazer uma linha de quatro peças ganha.

O trabalho prático foi concretizado em linguagem Java e, nesta meta, acompanhado por uma interface consola. Na meta seguinte será construída uma GUI para substituir a atual CLI.

2 Diagrama da Máquina de Estados

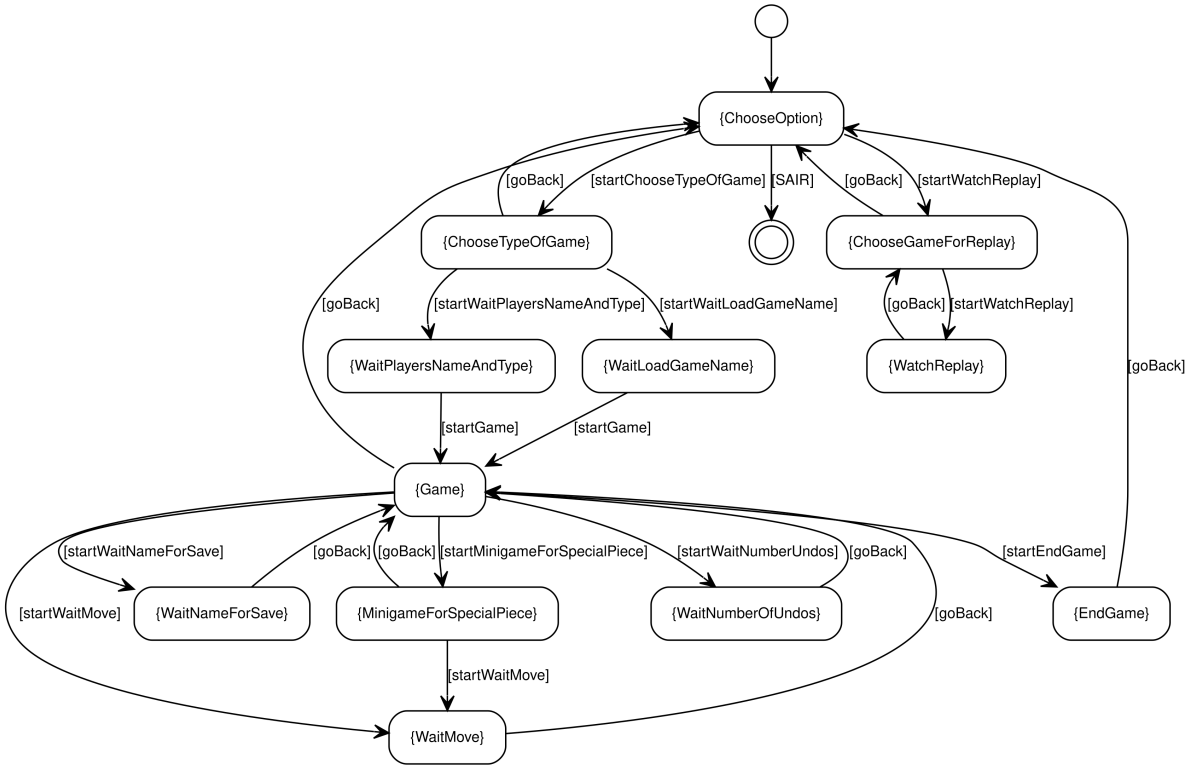

Figura 1: Diagrama da Máquina de Estados

Este diagrama de estados foi feito em yUML e o ficheiro código encontra-se em anexo.

3 Arquitetura Geral

O IDE usado foi IntelliJ IDEA.

Usando o paradigma de programação com máquina de estados, a classe **JogoCLI** comunica exclusivamente com a máquina de estados, que irá então depois delegar o resto da lógica. Para além disso faz recurso a alguns métodos static para tratamento de input e output de modo a manter o código mais limpo e legível.

Existem duas classes utilitárias no programa, sendo uma a **UtilsUI** e a outra a **UtilsFiles**. A primeira contém vários métodos que ajudam no tratamento de input e output, a segunda contém vários métodos para ajudar na leitura e gravação de ficheiros. Estas possuem os seus métodos declarados como static.

A máquina de estados trata da lógica de evolução da interface, ou seja, é com esta que evoluímos de estados mas toda a lógica envolvente ao jogo é depois delegada para uma classe chamada **JogoManager**.

A classe **JogoManager** é a primeira a ter acesso direto ao jogo. Esta classe trata assim do **Jogo** e do **CommandManager**. Deste maneira, esta é responsável por decidir que métodos devem ou não fazer uso do **CommandManager**. Esta é também a classe que é gravada em ficheiro de save ou replay dum jogo.

Exemplificando, se eu quiser jogar uma peça na coluna 1, a **JogoCLI** pede-me a coluna, esta coluna é enviada para a **StateMachine**, a **StateMachine** envia para o estado, o estado envia para o **JogoManager**, o **JogoManager** envia para o **CommandManager**, o **CommandManager** invoca o comando necessário, o comando invocado finalmente comunica com o **Jogo**.

4 Classes Usadas

As principais classes usadas no programa são as que se seguem. De modo a não ter um relatório excessivamente extenso, irão apenas ser mencionadas Interfaces e Adapters.

4.1 JogoApp

Classe que contém a main. Apenas responsável por inicializar a **StateMachine** e a **JogoCLI**.

4.2 JogoCLI

Classe que serve de interface. É esta que interage com o utilizador. Toda a informação recebida é depois tratada pelas classes de lógica.

4.3 StateMachine

Esta classe representa a máquina de estados. É esta que trata toda a lógica da máquina.

4.4 IState

Classe interface dos estados. Esta é depois implementada pelo **StateAdapter**.

4.5 StateAdapter

Classe que implementa a **IState**. Esta é depois extendida por todos os outros estados.

4.6 JogoManager

Classe que gere o **Jogo** e o **CommandManager**. Todos os comandos para enviar ao jogo passam por aqui e esta decide se precisam de integrar o **CommandManager** ou não.

4.7 CommandManager

Classe que gere certos comandos do jogo e guarda um histórico destes. Atinge isto com recurso a métodos que invocam comandos e fazem undo dos mesmo implementados por classes que implementam e extendem a classe interface **ICommand**.

4.8 ICommand

Classe interface dos commandos. Esta é depois implementada pelo **CommandAdapter**.

4.9 CommandAdapter

Classe que implementa a **ICommand**. Esta é depois extendida por todos os outros comandos.

4.10 Minigame

Classe abstrata que possuí a lógica base para o funcionamento de um minijogo. Esta é depois extendida pelos minigames.

4.11 Player

Classe abstrata que possuí a lógica base para o funcionamento de um jogador. Esta é depois extendida pelos players.

4.12 Jogo

Esta é a classe que possuí toda a lógica do jogo.

4.13 UtilsUI

Classe com apenas métodos estáticos. Serve para ajudar no tratamento de informação vinda do utilizador.

4.14 UtilsFiles

Classe com apenas métodos estáticos. Serve para ajudar no tratamento de ficheiros.

4.15 AppStates, JogoStates & TypePiece

Para além das classes mencionadas anteriormente, o jogo possuí também 3 enumerações.

- **AppStates:** possui os estados da máquina de estados;
- **JogoStates:** possui os estados do jogo;
- **TypePiece:** possui os tipos de peça que o tabuleiro pode ter;

5 Relacionamento Entre as Classes

A interface **IState** é implementada pela classe **StateAdapter**, que por sua vez é extendida pelas classes que representam estados na máquina de estados: **ChooseGameForReplay**, **ChooseOption**, **ChooseTypeOfGame**, **EndGame**, **Game**, **MinigameForSpecialPiece**, **WaitLoadGameName**, **WaitMove**, **WaitNameForSave**, **WaitNumberOfUndos**, **WaitPlayersNameAndType** e **WatchReplay**.

Os estados são geridos pela classe **StateMachine** que vai invocando métodos dos estados e mantém sempre o estado atual.

A interface **ICommand** é implementada pela classe **CommandAdapter**, que por sua vez é extendida pelas classes que representam os diferentes comandos: **PlacePieceOnColumn**, **PlaceSpecialPieceOnColumn** e **UpdateJogo**.

Os comandos são geridos pela classe **CommandManager** que, para além de os invocar, também os guarda para possibilitar as funcionalidades de desfazer jogada e ver replay.

A classe abstrata **Minigame** é extendida pelas classes que representam os diferentes minijogos: **MathMinigame** e **TypingMinigame**.

Os minijogos são geridos exclusivamente pelo estado **MinigameForSpecialPiece**.

A classe abstrata **Player** é extendida pelas classes que representam os diferentes tipos de jogador: **PlayerAI** e **PlayerHuman**.

Os jogadores são geridos pela classe **Jogo**.

A classe **Jogo** possuí toda a lógica para o funcionamento do jogo.

Esta é gerida pela classe **JogoManager**, que gere também a classe **CommandManager**. Esta é a responsável por invocar métodos do **Jogo** e decidir quais devem ser invocados pelo **CommandManager**.

A introdução de dados pelo utilizador é tratada pela **JogoCLI**, que por sua vez delega essa informação para a **StateMachine**, que por sua vez a envia para o **IState** atual, que depois a envia para o **JogoManager**, que irá então escolher se precisa de usar o **CommandManager** ou se pode comunicar diretamente com o **Jogo**.

6 Funcionalidades Implementadas

Funcionalidade	**Realizado**	**Realizado Parcialmente**	**Não Realizado**
Interface de Linha de Consola	✓		
Jogo Base 4 Em Linha	✓		
Jogador Virtual	✓		
Minijogos para Peça Especial	✓		
Desfazer Jogadas	✓		
Replay de Jogos Finalizados	✓		
Manter um Máximo de 5 Replays	✓		
Criar e Carregar Saves	✓		

7 Conclusão

Servindo como maneira para aprender programação com máquinas de estado e também como maneira para aplicar o paradigma de programação orientada a objetos, este trabalho foi uma excelente oportunidade não só de aprendizagem como também de reforço de conhecimentos.

Para além disto, é também uma boa introdução a programação em Java, ficando assim mais familiarizado com as suas vantagens e desvantagens.

8 Anexos

Lista de Figuras