Instituto Superior de Engenharia de Coimbra

Eng. Informática

2020/21

Programação Avançada

Trabalho Prático – 4 Em Linha

Realizado por:

Diogo Miguel Pinto Pascoal – 2018019825

Índice

1 – Opções e decisões tomadas na Implementação

Existem alguns pontos no enunciado deste trabalho prático que não esclarecem explicitamente alguns destes pormenores. Deste modo, existem algumas decisões que foram tomadas autonomamente:

Leitura do ficheiro texto para TypeRacer

Para o jogo de texto, foi assumido que o nome do ficheiro texto que contém as palavras para este jogo deve ser “typeracer.txt”, sendo que este é carregado quando o programa é iniciado e não no início de cada novo jogo, sendo que qualquer erro de leitura será apresentado sempre em consola.

Gravar e Carregar um jogo

A funcionalidade de gravar um jogo só está disponível no início de uma ronda (estado AguardaJogador), sendo que não é possível guardar o estado do jogo no meio de um minijogo ou quando espera a decisão de jogar ou não esse minijogo, assim como no fim do jogo e ao reproduzir um replay.

No menu no topo do programa é possível carregar um jogo ou um replay a qualquer momento.

Ao carregar um jogo ainda é possível usar jogadas anteriores.

Peça Especial

A peça especial, designada peça dourada no contexto do jogo, pode ser usada mesmo que a coluna esteja cheia, sendo que limpa a coluna toda e passa a vez para o próximo jogador.

2 – Máquina de Estados

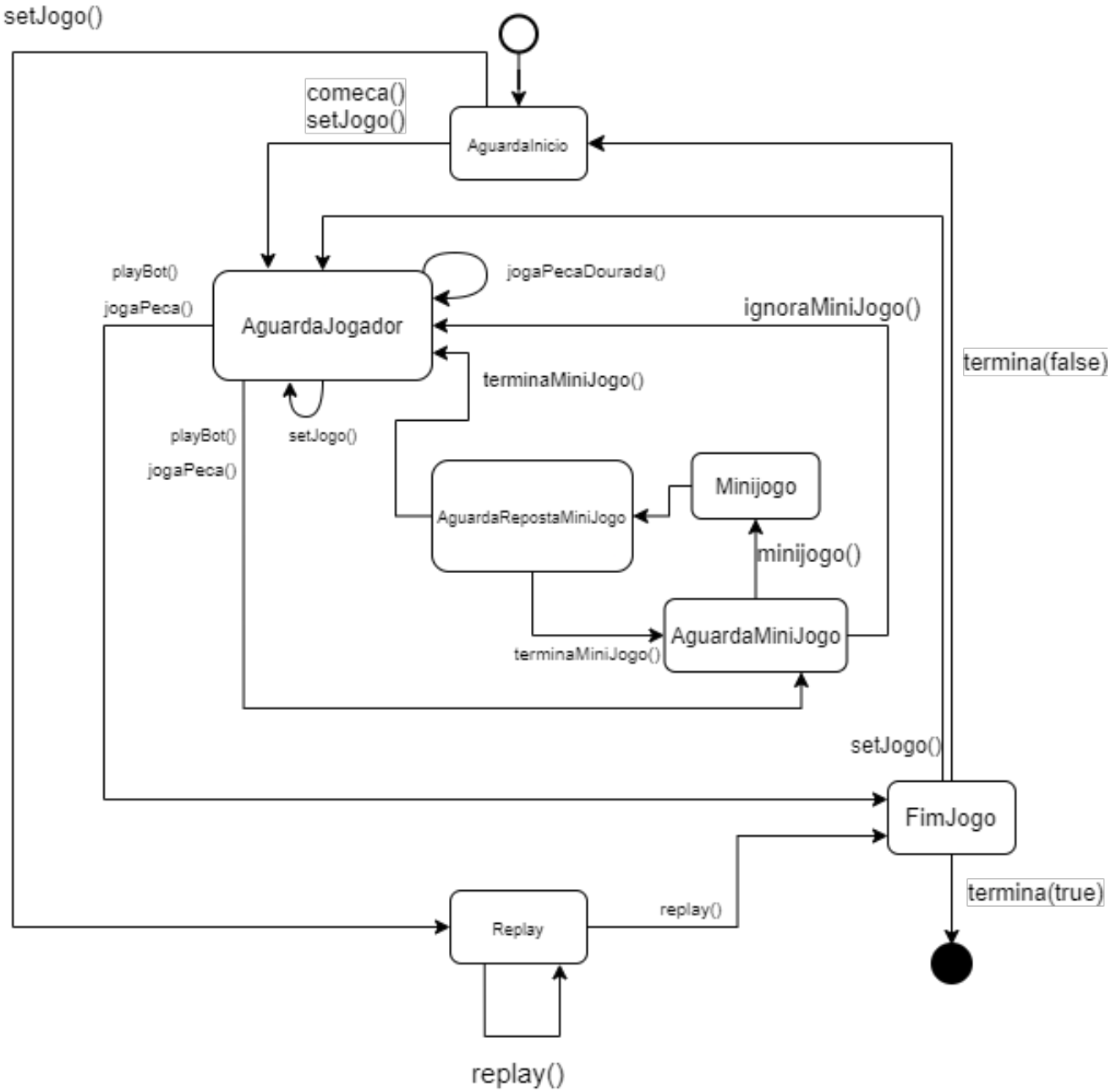

Figura 1 - Diagrama da Máquina de Estados

Existem nesta máquina de estados 7 estados:

2.1 – AguardaInicio

Neste estado apenas aguardamos pelo início de um novo jogo. É neste estado que obtemos o número de jogadores humanos a participar, assim como os seus nomes.

Saímos deste estado através da função comeca() que nos direciona para um AguardaJogador.

2.2 – AguardaJogador

Este estado dedica-se a obter a jogada de um jogador, seja esta Bot ou humano. Permite também analisar o log desta partida ou de partidas anteriores que tenham sido realizadas na mesma execução do programa e gravar ou carregar uma partida.

Após efetuar uma jogada, o jogo vai verificar se já existe um vencedor. Caso exista, irá passar para o estado FimJogo através da função termina(); Caso seja um empate, utilizará a mesma função.

Caso não exista nenhum vencedor, mas o próximo jogador tem um minijogo por jogar, irá automaticamente passar para o AguardaMiniJogo.

Caso nenhuma destas condições se verificar, volta para um novo estado AguardaJogador.

Estas transições estão presentes nas funções jogaPeca e playBot, sendo a única diferença que playBot existe para a execução de uma jogada do bot através da escolha de uma coluna aleatória que não se encontre preenchida.

2.3 – AguardaMiniJogo

Apenas para dar a opção de jogar um minijogo, este estado é um passo intermediário entre jogar o minijogo ou simplesmente continuar o jogo normal, sendo que são usadas respetivamente as funções minijogo() e ignoraMiniJogo().

2.4 – MiniJogo

Este estado serve unicamente para dar início ao minijogo, iniciando assim a contagem do tempo do jogo.

Passa automaticamente para um AguardaRespostaMiniJogo

2.5 – AguardaRepostaMinijogo

Neste estado é processado logicamente o minijogo atribuído, seja este o jogo da matemática ou o TypeRacer. Ao responder à questão, caso seja o jogo da matemática, o programa irá voltar para um novo estado do mesmo tipo à espera de mais respostas caso o tempo não tenha acabado e ainda não tenha acertado as 5 questões para vencer o minijogo. Caso seja o jogo TypeRacer, automaticamente valida e segue para um AguardaJogador.

2.6 – FimJogo

No FimJogo, é apresentado o vencedor do jogo e é dada a opção de sair ou recomeçar um jogo. A função termina irá começar um novo jogo, direcionando para um novo AguardaInicio, ou termina a execução do programa.

De notar que nos estados AguardaInicio, AguardaJogador e FimJogo, é sempre possível transicionar de volta para o AguardaJogador através do carregamento de um jogo guardado através da função setJogo().

2.7 – Replay

Este estado representa a reprodução de um replay de um jogo, sendo que segue iterativamente para um novo estado Replay a cada jogada reproduzida, direcionando para o FimJogo assim que o replay acabar.

3 – Diagrama de Mementos

Para a realização da funcionalidade “Voltar atrás”, foi usado o padrão Memento.

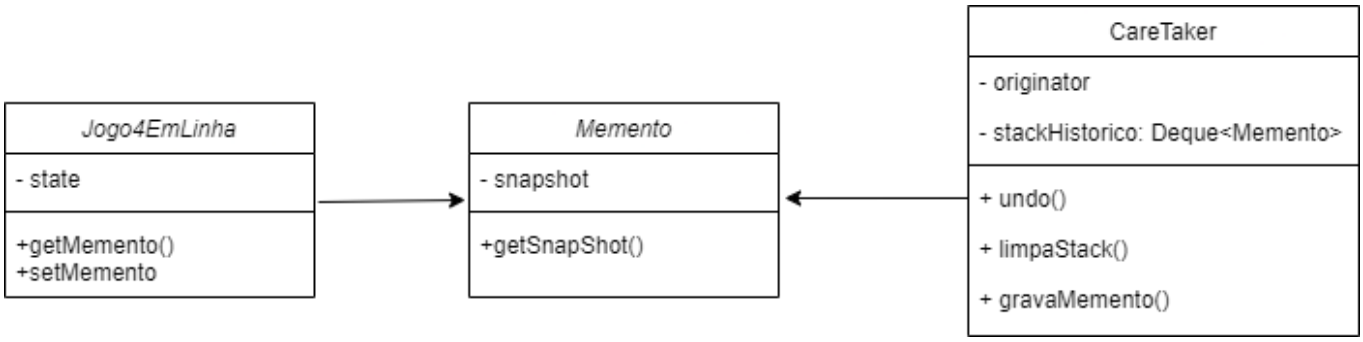

Figura 2 - Diagrama Padrão Memento

Nesta aplicação do padrão Memento, CareTaker é a classe que tem o conjunto de estados guardados do jogo, sendo esses estados provenientes da classe Memento.

Memento apenas guarda um objeto genérico, sendo que no contexto do programa é dado o cast onde é efetuada a leitura do memento em si, na classe Jogo4EmLinha.

Na classe Jogo4EmLinha, é criado um array de Objetos que contêm o tabuleiro, assim como o próximo jogador, sendo estas as únicas informações necessárias armazenar, pois o numero de rondas para minijogo volta a 0 automaticamente e o resto da informação (como por exemplo créditos e peças douradas) também não deve ser alterado com o uso desta funcionalidade. Este array de Objetos acaba por ser o snapshot que o Memento guarda e que envia ao CareTaker para armazenar para uso futuro.

3.1 – Mementos para Replay

Fora do contexto de Voltar atrás, o CareTaker tem um Deque do mesmo tipo que o stackHistorico, sendo a única diferença a remoção do limite de 10 jogadas guardadas, de modo a poder ler as jogadas todas ao iniciar um replay.

4 – Descrição de Classes Utilizadas

Seguindo a ordem dos packages do projeto, será descrita cada uma das classes presentes no projeto, ou o que um conjunto destas classes representa.

4.1 - jogo.iu.texto – Connect4IU

Esta classe, como o nome indica, serve apenas de interface texto para comunicar com o utilizador final para a realização do jogo.

4.2 – jogo.logica.dados

Existem 3 classes neste package: **Jogador**, **Jogo4EmLinha** e **MiniJogos**.

Jogador

Esta classe simplesmente representa um jogador, seja ele humano ou bot, armazenando o seu nome, as peças douradas que tem, assim como os créditos e a informação acerca da natureza deste jogador (bot ou humano).

Jogo4EmLinha

Classe principal do programa, a classe Jogo4EmLinha representa a central do programa, sendo esta a função que contêm o tabuleiro de jogo, os jogadores, logs do jogo atual e passados e também os minijogos.

Esta classe trata de toda a lógica, inclusive de correr os minijogos, direcionando as ações para o seu objeto minijogo. As funções de escrever o ficheiro texto para replay, guardar e carregar jogos e afins são todas declaradas nesta classe.

MiniJogos

Esta classe contém informação acerca da realização dos minijogos e processamento lógico, sendo que é o jogo em si que comunica com esta classe para pedir as questões e processar as respostas ao minijogo atribuído. Também é esta a classe que mantém informação acerca de qual é o jogo a atribuir ao próximo jogador.

4.3 – jogo.logica.estados

Todas as classes neste package representam a mesma coisa: um estado. Cada um dos estamos acima referidos no diagrama de estados estão aqui representados, existindo apenas 2 elementos que não estão representados no diagrama que são a classe abstrata EstadoAdapter, que serve para implementar a máquina de estados a estas classes, e a Interface IEstado.

4.4 – jogo.logica.memento

Neste package encontram-se as duas classes e uma interface descritas no Diagrama do Padrão Mementos que permitem a concretização deste padrão.

4.5 – jogo.logica.MaquinaEstados

Esta classe é o intermédio entre a interface com o utilizador e todos os estados, a Máquina de Estados em si.

Esta classe é responsável saber o estado atual, assim como redirecionar o tráfego para cada um dos estados a fim de realizar as ações pretendidas em cada uma das situações.

Também comunica com o jogo diretamente em algumas ocasiões (como por exemplo ao carregar um novo jogo), sendo esta classe a única a ter referência direta com o Jogo4EmLinha e o CareTaker.

4.6 – jogo.logica.SaveAndLoad

Esta classe apenas contém 2 funções estáticas que são usadas para gravar um jogo para ficheiro binário e para carregar esse mesmo ficheiro binário para o programa, respetivamente

4.7 – jogo.logica.Operacao e jogo.logica.Situacao

Estes dois Enums usados para indicar qual a operação que está a ser usada para o minijogo de matemática e para indicar em que situação a maquina de estados se encontra, respetivamente.

4.8 – jogo.logica.Utils.Utils

Conjunto de funções úteis ao pedido de informação ao utilizador.

4.9 – jogo.Main

Classe que inicia a execução do programa.

2ª Meta

4.10 – jogo.iu.gui.estados

Conjunto de classes que contêm panes que representam os diferentes estados da máquina de estados.

4.11 – jogo.iu.gui.Connect4Grid

GridPane que desenha o tabuleiro do jogo e que representa o estado atual do jogo.

4.12 – jogo.iu.gui.PrincipalPane

Pane onde a maioria da informação é apresentada e atualizada enquanto o jogador está em jogo.

Esta classe está encarregue de incluir o Connect4Grid, de apresentar a informação do jogador (nome, créditos e peças douradas) e dar as opções para continuar com a progressão no jogo e na aplicação.

4.13 – jogo.iu.gui.VoltarAtrasPane

Pequena VBox para perguntar ao utilizador quantas rondas deseja voltar atrás.

4.14 – jogo.iu.gui.ConstantesGUI

Conjunto de Strings e booleans que são utlizados pelos diversos panes, especialmente para dar uso ao PropertyChangeSupport.

4.15 – jogo.iu.gui.Connect4IU_Grafico

Pane pai da aplicação, que contêm todos os panes que representam as diferentes alturas do jogo (Aguardar início do jogo, jogar, terminar jogo, etc.).

Também é neste pane que é criado a barra de menus que contém as opções para carregar um jogo ou um replay e para guardar o jogo.

4.16 – jogo.logica.JogoObservavel

Layer de ligação entre a Máquina de Estados e o PropertyChangeSupport.

5 – Relacionamento entre as classes

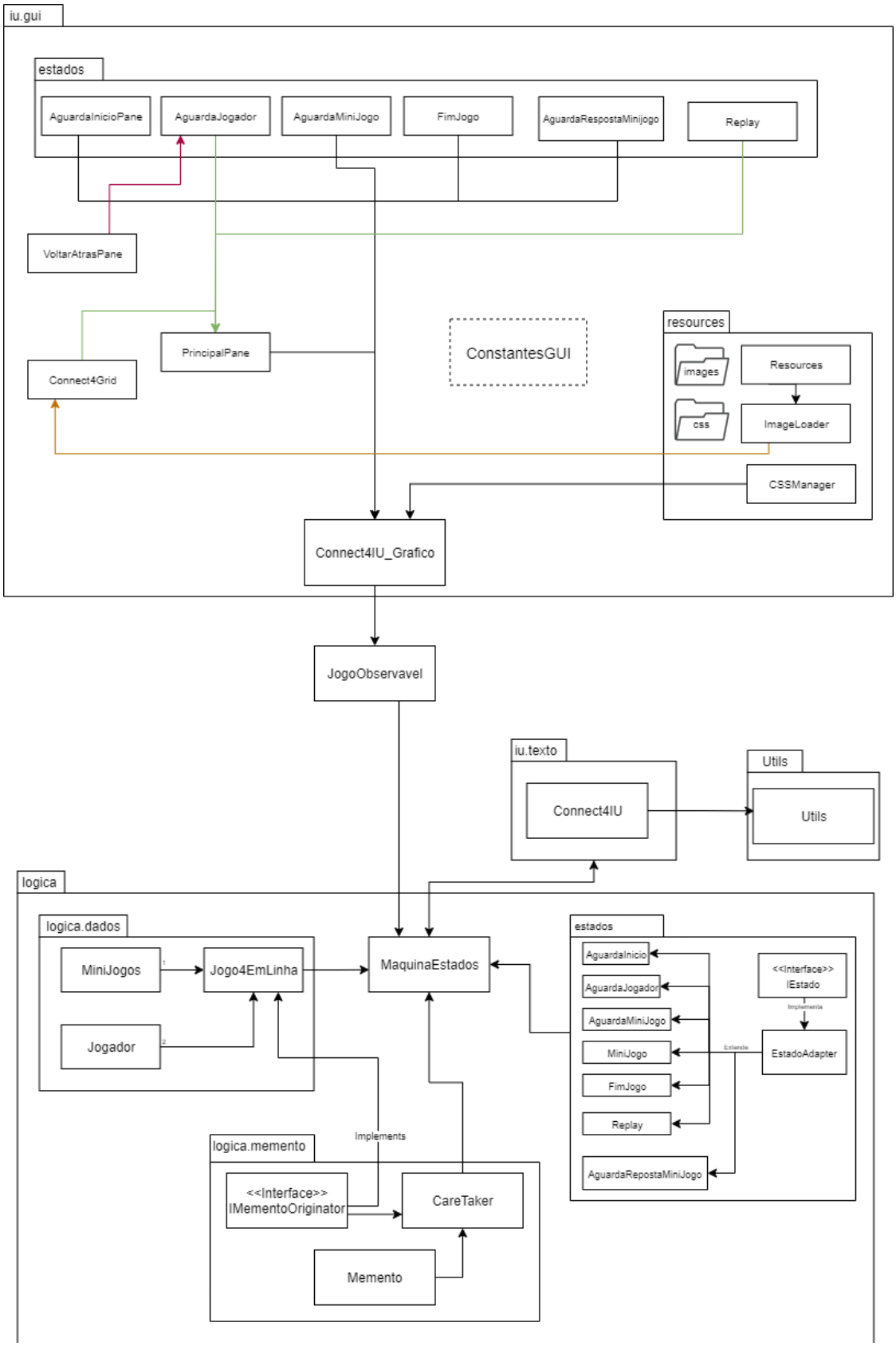

Figura 3 - Relacionamento entre as classes

Como podemos ver, a MaquinaEstados é a peça central do sistema, sendo que todos comunicam com esta classe e esta reencaminha a informação para onde for necessário, sendo que o único acesso que o IU tem ao jogo é através da Máquina de Estados.

Todas as classes dentro do package Estados estendem o EstadoAdapter, sendo que é isto que as torna "estados", sendo este EstadoAdaptar a implementação da interface IEstado.

O CareTaker apenas contacta com a MaquinaEstados, sendo que obtem os seus mementos a partir do Jogo4EmLinha que implementa a interface IMementoOriginator, podendo assim criar objetos Memento, sendo este os objetos que o CareTaker armazena.

Os MiniJogos e os Jogadores são responsabilidade exclusiva do Jogo.

Para usar o PropertyChangeSupport, que dá base à maior parte da funcionalidade do IU em JavaFX, é criado uma layer intermédia entre o IU gráfico e a máquina de estados, o JogoObservavel, que irá implementar esta funcionalidade.

6 – Funcionalidades Cumpridas

Funcionalidade / Regras	Cumprido
Implementação do jogo em modo texto	X
Suporte para dois jogadores humanos, humano vs. computador ou computador vs. computador	X
Gravação/carregamento do jogo (um jogo carregado de ficheiro deve permitir a sua continuação)	X
Sortear Jogador a efetuar a primeira jogada	X
Implementação dos Minijogos e da Peça especial	X
Funcionalidade "Voltar Atrás"	X

Iniciar um novo jogo após o término	X
Histórico dos últimos 5 jogos completos para possibilitar o “replay”	incorreto
Padrão máquina de estados para concretizar a lógica do jogo	X
Log (“registo”) interno	X

2ª Meta

Funcionalidade / Regras	Cumprido
Implementação do jogo em modo gráfico	X
Correção de erros da 1ª Meta	X
Histórico dos últimos 5 jogos completos para possibilitar o “replay”	X
Funcionalidade Completa do Jogo	X