

PROJETO MINI - AKINATOR

Prof. Glender Brás

Aluno: Lucelho Cristiano Vieira Da Silva

RA: 32218540

Objetivo:

- Fazer um jogo de adivinhação onde o usuário (Jogador) irá imaginar um personagem e responder perguntas e o jogo consegue adivinhar em qual personagem ele está perguntando.

O Programa deve ter:

- Exiba ao usuário 10 opções de animais (ou outro tema a sua escolha).
- Seu programa deverá solicitar ao usuário que pense em um animal (apenas pense, não deverá ser informado ao programa).
- Logo após você deve fazer algumas perguntas, cuja resposta seja apenas “sim” ou “não”
- À medida que o usuário for respondendo, seu programa deverá eliminar animais que não correspondem à resposta do usuário, até que reste apenas 1.
- Restando apenas 1, seu programa deverá exibir a mensagem ao usuário: “Seu animal é (...)” com a resposta encontrada
- Dependendo das respostas do usuário você pode precisar de mais ou menos perguntas. Vamos estabelecer um mínimo de 5 e máximo de 15 perguntas para chegar à resposta final.

Sugestão:

- Organize os “animais” em lista dimensionável (ex: ArrayList do java).

Crie 6 Classes:

- Main.java
- Features.java
- Question.java
- Filter.java
- GameInterface.java
- GameLogic.java

Classe main.java

Esta é a classe principal do jogo: "cérebro"

A classe Main é o ponto de entrada do programa. Ela inicializa o jogo, carrega os personagens e inicia a interface gráfica.

```
17 public static void main(String[] args) {
18     List<Features> listaPersonagens = new ArrayList<>();
19     adicionarPersonagens(listaPersonagens);
20     System.out.println("Quantidade de personagens disponíveis: " + listaPersonagens.size());
21
22     System.out.println();
23
24     System.out.println(x:"Iniciando o jogo de adivinhação...");
25     System.out.println(x:"\nA interface gráfica será aberta em breve. Prepare-se para responder às perguntas!");
26
27     SwingUtilities.invokeLater(() -> {
28         new GameInterface(listaPersonagens);
29     });
30 }
```

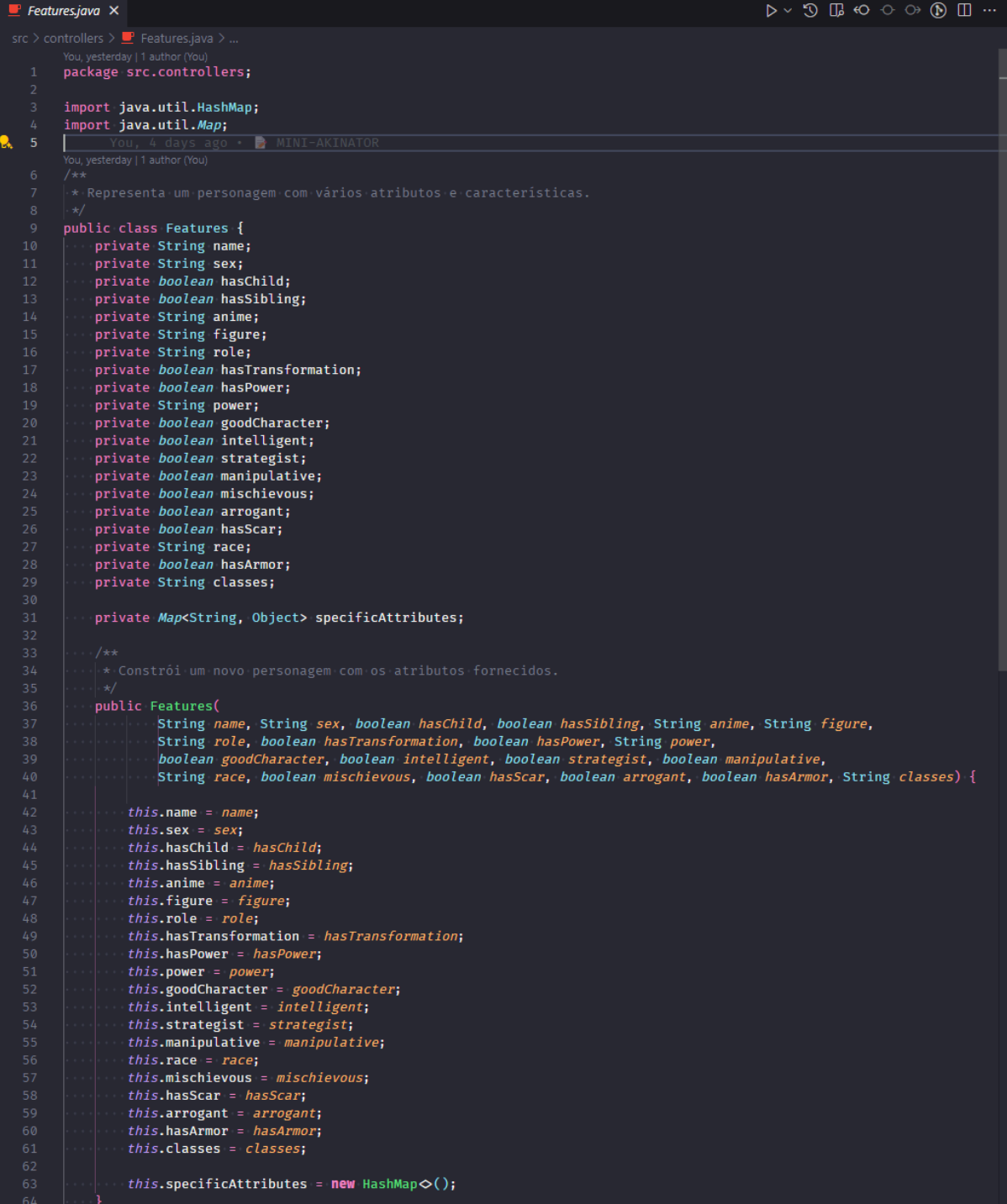
- Cria uma lista vazia para armazenar os personagens.
- Chama adicionarPersonagens para preencher a lista.
- Inicia a interface gráfica do jogo usando SwingUtilities.invokeLater para garantir que a UI seja criada na thread de eventos do Swing.

```
32 public static void adicionarPersonagens(List<Features> listaPersonagens) {
33     DragonBallCharacters.adicionarPersonagens(listaPersonagens);
34     // SoloLevelingCharacters.adicionarPersonagens(listaPersonagens);
35     // CavaleirosDoZodiacoCharacters.adicionarPersonagens(listaPersonagens);
36     HarryPotterCharacters.adicionarPersonagens(listaPersonagens);
37     // NarutoCharacters.adicionarPersonagens(listaPersonagens);
38     UniversoMarvelPersonagens.adicionarPersonagens(listaPersonagens);
39     UniversoDaDCPersonagens.adicionarPersonagens(listaPersonagens);
40 }
41 }
```

- Este método chama várias classes de personagens para adicionar personagens à lista principal.

Classe Features.java

A classe Features representa um personagem com seus atributos e características.



```
1 package src.controllers;
2
3 import java.util.HashMap;
4 import java.util.Map;
5
6 /**
7  * Representa um personagem com vários atributos e características.
8  */
9 public class Features {
10     private String name;
11     private String sex;
12     private boolean hasChild;
13     private boolean hasSibling;
14     private String anime;
15     private String figure;
16     private String role;
17     private boolean hasTransformation;
18     private boolean hasPower;
19     private String power;
20     private boolean goodCharacter;
21     private boolean intelligent;
22     private boolean strategist;
23     private boolean manipulative;
24     private boolean mischievous;
25     private boolean arrogant;
26     private boolean hasScar;
27     private String race;
28     private boolean hasArmor;
29     private String classes;
30
31     private Map<String, Object> specificAttributes;
32
33     /**
34      * Constrói um novo personagem com os atributos fornecidos.
35      */
36     public Features(
37         String name, String sex, boolean hasChild, boolean hasSibling, String anime, String figure,
38         String role, boolean hasTransformation, boolean hasPower, String power,
39         boolean goodCharacter, boolean intelligent, boolean strategist, boolean manipulative,
40         String race, boolean mischievous, boolean hasScar, boolean arrogant, boolean hasArmor, String classes) {
41
42         this.name = name;
43         this.sex = sex;
44         this.hasChild = hasChild;
45         this.hasSibling = hasSibling;
46         this.anime = anime;
47         this.figure = figure;
48         this.role = role;
49         this.hasTransformation = hasTransformation;
50         this.hasPower = hasPower;
51         this.power = power;
52         this.goodCharacter = goodCharacter;
53         this.intelligent = intelligent;
54         this.strategist = strategist;
55         this.manipulative = manipulative;
56         this.race = race;
57         this.mischievous = mischievous;
58         this.hasScar = hasScar;
59         this.arrogant = arrogant;
60         this.hasArmor = hasArmor;
61         this.classes = classes;
62
63         this.specificAttributes = new HashMap<>();
64     }
```

Este construtor inicializa um objeto Features com todos os atributos básicos de um personagem. Ele também cria um HashMap vazio para armazenar atributos específicos.

```

65
66 /**
67  * Adiciona um atributo específico ao personagem.
68  *
69  * @param key A chave do atributo
70  * @param valor O valor do atributo
71  */
72 public void addSpecificAttribute(String chave, Object valor) {
73     specificAttributes.put(chave, valor);
74 }
75

```

Este método adiciona um atributo específico ao personagem. É para características únicas que não são comuns a todos os personagens.

```

76 /**
77  * Recupera um atributo booleano específico.
78  *
79  * @param key A chave do atributo
80  * @return O valor booleano do atributo, ou false se não for encontrado ou não
81  *         for booleano
82  */
83 public boolean getSpecificBooleanAttribute(String chave) {
84     Object value = specificAttributes.get(chave);
85     return value instanceof Boolean ? (Boolean) value : false;
86 }
87
88 /**
89  * Recupera um atributo de string específico.
90  *
91  * @param key A chave do atributo
92  * @return O valor da string do atributo, ou uma string vazia se não for
93  *         encontrada ou não for uma string
94  */
95 public String getSpecificStringAttribute(String chave) {
96     Object value = specificAttributes.get(chave);
97     return value instanceof String ? (String) value : "";
98 }
99

```

Os métodos `getSpecificBooleanAttribute` e `getSpecificStringAttribute` recuperam atributos específicos do personagem, retornando um valor booleano ou string, respectivamente. Se o atributo não existir ou não for do tipo esperado, retornam um valor padrão.

Classe Question.java

A classe Question representa uma pergunta no jogo e fornece métodos para gerenciar perguntas.

```
30 public static List<Question> obterPerguntasRelevantes(List<Features> personagens, Set<String> perguntasFeitas) {
31     List<Question> todasPerguntas = criarListaPerguntas();
32     List<Question> perguntasRelevantes = new ArrayList<>();
33
34     for (Question pergunta : todasPerguntas) {
35         if (!perguntasFeitas.contains(pergunta.getPergunta()) && perguntaERrelevante(pergunta, personagens)) {
36             perguntasRelevantes.add(pergunta);
37         }
38     }
39
40     perguntasRelevantes.sort((p1, p2) -> Double.compare(
41         calcularEntropiaInformacao(p2, personagens),
42         calcularEntropiaInformacao(p1, personagens)));
43
44     return perguntasRelevantes;
45 }
46
```

Este método filtra e ordena as perguntas para obter as mais relevantes. Ele:

1. Cria uma lista de todas as perguntas possíveis.
2. Filtra para incluir apenas perguntas não feitas e relevantes.
3. Ordena as perguntas com base na entropia de informação.

```
47 public static List<Question> obterPerguntasEspecificasPersonagem(Features personagem, Set<String> perguntasFeitas) {
48     List<Question> todasPerguntas = criarListaPerguntas();
49     List<Question> perguntasEspecificas = new ArrayList<>();
50
51     for (Question pergunta : todasPerguntas) {
52         if (!perguntasFeitas.contains(pergunta.getPergunta())) {
53             boolean respostaPergunta = Filter1.getFeatures(personagem, pergunta.getFeatures());
54             String perguntaFormatada = String.format(format: "O personagem %s %s?",
55                 respostaPergunta ? "" : "não",
56                 pergunta.getPergunta().toLowerCase().replace(target: "o personagem ", replacement: ""));
57             perguntasEspecificas.add(new Question(perguntaFormatada, pergunta.getFeatures(), pergunta.isSpecific()));
58         }
59     }
60
61     Collections.shuffle(perguntasEspecificas);
62     return perguntasEspecificas;
63 }

```

Este método é usado para obter perguntas específicas sobre um personagem descoberto. Aqui está como ele funciona:

1. Cria uma lista de todas as perguntas possíveis.
2. Itera sobre todas as perguntas, ignorando as que já foram feitas.
3. Para cada pergunta não feita:

- Determina a resposta correta para o personagem descoberto.
 - Formata a pergunta de acordo com a resposta (afirmativa ou negativa).
 - Adiciona a pergunta formatada à lista de perguntas específicas.
4. Embaralha a lista de perguntas específicas para adicionar aleatoriedade.
 5. Retorna a lista de perguntas específicas.

Este método é útil para a fase final do jogo, quando o personagem já foi descoberto e o jogo está apenas confirmando características do personagem com o jogador.

```

65 private static boolean perguntaERrelevante(Question pergunta, List<Features> personagens) {
66     boolean temVerdadeiro = false;
67     boolean temFalso = false;
68
69     for (Features p : personagens) {
70         boolean valor = Filter1.getFeatures(p, pergunta.getFeatures());
71         if (valor) {
72             temVerdadeiro = true;
73         } else {
74             temFalso = true;
75         }
76
77         if (temVerdadeiro && temFalso) {
78             return true;
79         }
80     }
81
82     return false;
83 }

```

Este método determina se uma pergunta é relevante para o conjunto atual de personagens. Aqui está como ele funciona:

1. Inicializa duas variáveis booleanas: **temVerdadeiro** e **temFalso**.
2. Itera sobre todos os personagens na lista.
3. Para cada personagem, verifica se a característica da pergunta é verdadeira ou falsa.
4. Se encontrar pelo menos um personagem para o qual a característica é verdadeira e outro para o qual é falsa, a pergunta é considerada relevante.
5. Se, após checar todos os personagens, a pergunta não dividir o conjunto (ou seja, todos os personagens têm a mesma resposta), a pergunta é considerada irrelevante.

Este método é crucial para otimizar o processo de seleção de perguntas. Ele garante que apenas perguntas que efetivamente ajudam a distinguir entre os personagens restantes sejam consideradas, melhorando a eficiência do jogo.

```

85 private static double calcularEntropiaInformacao(Question pergunta, List<Features> personagens) {
86     int total = personagens.size();
87     int verdadeiros = 0;
88
89     for (Features p : personagens) {
90         if (Filter1.getFeatures(p, pergunta.getFeatures())) {
91             verdadeiros++;
92         }
93     }
94
95     double proporcaoVerdadeiros = (double) verdadeiros / total;
96     double proporcaoFalsos = 1 - proporcaoVerdadeiros;
97
98     if (proporcaoVerdadeiros == 0 || proporcaoFalsos == 0) {
99         return 0;
100     }
101
102     return -proporcaoVerdadeiros * Math.log(proporcaoVerdadeiros) / Math.log(2)
103         - proporcaoFalsos * Math.log(proporcaoFalsos) / Math.log(2);
104 }

```

Este método calcula a entropia de informação de uma pergunta, que é uma medida de quão bem a pergunta divide o conjunto de personagens. Ele:

1. Conta quantos personagens têm a característica verdadeira.
2. Calcula as proporções de respostas verdadeiras e falsas.
3. Calcula a entropia usando a fórmula de entropia de Shannon.

A entropia é máxima quando a pergunta divide o conjunto de personagens em dois grupos iguais, o que a torna mais informativa.

A fórmula básica da entropia de Shannon é:

$$H = -\sum(p(x) * \log_2(p(x)))$$

Onde:

- H é a entropia
- $p(x)$ é a probabilidade de um evento x
- \sum representa o somatório de todos os eventos possíveis
- A entropia será máxima (1) quando a pergunta dividir o conjunto exatamente ao meio (50% verdadeiro, 50% falso).
- Será mínima (0) quando todos os personagens tiverem a mesma resposta.
- Quanto maior a entropia, mais informativa é a pergunta.

Exemplo:

Suponha que temos 8 personagens, e uma pergunta "O personagem é do sexo masculino?" divide o grupo em 5 masculinos e 3 femininos.

- `proporcaoVerdadeiros = 5/8 = 0.625`

- `proporcaoFalsos = 3/8 = 0.375`

Entropia = $-0.625 * \log_2(0.625) - 0.375 * \log_2(0.375) \approx 0.954$

Este valor alto indica que a pergunta é bastante informativa, pois divide bem o conjunto de personagens.

Conclusão

Ao usar a entropia de Shannon, o jogo pode selecionar perguntas que maximizam a informação obtida a cada etapa, tornando o processo de adivinhação mais eficiente e reduzindo o número de perguntas necessárias para identificar o personagem

Classe Filter.java

A classe Filter fornece funcionalidade para filtrar personagens com base em suas características.

```
114 public static void filtrarPersonagens(List<Features> listaPersonagens, String feature, boolean valor) {  
115     List<Features> personagensFiltrados = listaPersonagens.stream()  
116         .filter(personagem -> getFeatures(personagem, feature) == valor)  
117         .collect(Collectors.toList());  
118     listaPersonagens.clear();  
119     listaPersonagens.addAll(personagensFiltrados);  
120 }
```

Este método filtra a lista de personagens, mantendo apenas aqueles que têm o valor especificado para a característica dada. Ele:

1. Usa streams para filtrar os personagens.
2. Limpa a lista original.
3. Adiciona os personagens filtrados de volta à lista.

```
122 public static boolean getFeatures(Features personagem, String feature) {  
123     Function<Features, Boolean> check = filter.get(feature);  
124     return check != null ? check.apply(personagem) : false;  
125 }  
126
```

Este método verifica se um personagem possui uma característica específica. Ele usa um mapa de funções (filter) para verificar diferentes características.

Classe GameInterface.java

A classe GameInterface gerencia a interface gráfica do jogo.

```
34 public GameInterface(List<Features> characters) {  
35     this.gameLogic = new GameLogic(characters);  
36     initializeUI();  
37 }
```

Este construtor inicializa a interface do jogo. Ele cria uma nova instância de GameLogic com a lista de personagens fornecida e chama o método initializeUI para configurar a interface gráfica.

```
39 private void initializeUI() {  
40     setTitle(TITLE);  
41     setSize(WINDOW_WIDTH, WINDOW_HEIGHT);  
42     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
43     setLocationRelativeTo(c:null);  
44     setContentPane(createMainPanel());  
45     setVisible(b:true);  
46 }
```

Este método configura a janela principal do jogo, definindo título, tamanho, comportamento de fechamento, posição centralizada e conteúdo inicial.

```
48 private JPanel createMainPanel() {  
49     mainPanel = new JPanel();  
50     mainPanel.setLayout(new BoxLayout(mainPanel, BoxLayout.Y_AXIS));  
51     mainPanel.setBackground(BACKGROUND_COLOR);  
52     mainPanel.setBorder(new EmptyBorder(top:20, left:20, bottom:20, right:20));  
53  
54     mainPanel.add(Box.createVerticalGlue());  
55     mainPanel.add(createTitleLabel());  
56     mainPanel.add(Box.createRigidArea(new Dimension(width:0, height:20)));  
57     mainPanel.add(createSubtitleLabel());  
58     mainPanel.add(Box.createRigidArea(new Dimension(width:0, height:20)));  
59     mainPanel.add(createImageLabel());  
60     mainPanel.add(Box.createRigidArea(new Dimension(width:0, height:20)));  
61     mainPanel.add(createStartButton());  
62     mainPanel.add(Box.createVerticalGlue());  
63  
64     return mainPanel;  
65 }
```

Cria o painel principal da interface inicial do jogo, configurando layout, cor de fundo e adicionando componentes como título, subtítulo, imagem e botão de início.

```
67 private JLabel createTitleLabel() {  
68     return createStyledLabel(text:"MINI AKINATOR", TITLE_FONT, TEXT_COLOR);  
69 }  
70  
71 private JLabel createSubtitleLabel() {  
72     return createStyledLabel(text:"Consigo adivinhar qualquer personagem em que você esteja pensando", SUBTITLE_FONT,  
73         Color.DARK_GRAY);  
74 }
```

Cria e retorna o rótulo do título do jogo e subtítulo do jogo

```

76 private JLabel createImageLabel() {
77     JLabel label = new JLabel();
78     label.setAlignmentX(Component.CENTER_ALIGNMENT);
79     ImageIcon imageIcon = new ImageIcon(filename:"C:/Users/citap/Documents/Nova pasta/developers");
80     Image image = imageIcon.getImage().getScaledInstance(width:400, height:300, Image.SCALE_SMOOTH);
81     label.setIcon(new ImageIcon(image));
82     return label;
83 }

```

Cria e retorna um rótulo contendo a imagem inicial do jogo, redimensionada para se ajustar à interface.

```

85 private JButton createStartButton() {
86     return createStyledButton(text:"Topa o desafio?", BUTTON_COLOR, e → startGame());
87 }

```

Cria e retorna o botão de início do jogo, com texto e ação personalizados.

```

97 private JButton createStyledButton(String text, Color bgColor, java.awt.event.ActionListener action) {
98     // You, yesterday | 1 author (You)
99     JButton button = new JButton(text) {
100         @Override
101         protected void paintComponent(Graphics g) {
102             Graphics2D g2 = (Graphics2D) g.create();
103             g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
104             g2.setColor(getBackground());
105             g2.fillRoundRect(x:0, y:0, getWidth(), getHeight(), arcWidth:15, arcHeight:15);
106             g2.dispose();
107             super.paintComponent(g);
108         }
109     };
110     button.setFont(BUTTON_FONT);
111     button.setBackground(bgColor);
112     button.setForeground(Color.WHITE);
113     button.setBorderPainted(b:false);
114     button.setFocusPainted(b:false);
115     button.setContentAreaFilled(b:false);
116     button.addActionListener(action);
117     button.setAlignmentX(Component.CENTER_ALIGNMENT);
118     button.setPreferredSize(new Dimension(width:250, height:60));
119     // Add hover effect
120     // You, yesterday | 1 author (You)
121     button.addMouseListener(new MouseAdapter() {
122         @Override
123         public void mouseEntered(MouseEvent e) {
124             button.setBackground(bgColor.brighter());
125             button.setCursor(new Cursor(Cursor.HAND_CURSOR));
126         }
127         @Override
128         public void mouseExited(MouseEvent e) {
129             button.setBackground(bgColor);
130             button.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
131         }
132     });
133     // Add scale effect on press
134     // You, yesterday | 1 author (You)
135     button.addMouseListener(new MouseAdapter() {
136         @Override
137         public void mousePressed(MouseEvent e) {
138             button.setSize(new Dimension((int) (button.getWidth() * 0.95), (int) (button.getHeight() * 0.95)));
139         }
140         @Override
141         public void mouseReleased(MouseEvent e) {
142             button.setSize(new Dimension((int) (button.getWidth() / 0.95), (int) (button.getHeight() / 0.95)));
143         }
144     });
145     return button;
146 }

```

Método utilitário para criar botões estilizados com texto, cor de fundo e ação personalizados. Inclui efeitos de hover e pressionamento.

```

150     private void startGame() {
151         gameLogic.resetGame();
152         setContentPane(createGamePanel());
153         askNextQuestion();
154         revalidate();
155         repaint();
156     }

```

Inicia um novo jogo, resetando a lógica, criando um novo painel de jogo e fazendo a primeira pergunta.

```

158     private JPanel createGamePanel() {
159         JPanel panel = new JPanel(new BorderLayout(hgap:20, vgap:20));
160         panel.setBackground(BACKGROUND_COLOR);
161
162         questionLabel = createStyledLabel(text:"", QUESTION_FONT, TEXT_COLOR);
163         panel.add(questionLabel, BorderLayout.CENTER);
164
165         JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, hgap:20, vgap:20));
166         yesButton = createStyledButton(text:"Sim", new Color(r:100, g:200, b:100), e → processAnswer(answer:true));
167         noButton = createStyledButton(text:"Não", new Color(r:200, g:100, b:100), e → processAnswer(answer:false));
168         buttonPanel.add(yesButton);
169         buttonPanel.add(noButton);
170         panel.add(buttonPanel, BorderLayout.SOUTH);
171
172         return panel;
173     }

```

Cria o painel principal para a interface de jogo ativa, incluindo o rótulo da pergunta e os botões de resposta.

```

175     private void processAnswer(boolean answer) {
176         gameLogic.processAnswer(answer);
177         if (gameLogic.isGameOver()) {
178             showResult();
179         } else {
180             askNextQuestion();
181         }
182     }

```

Processa a resposta do jogador, atualiza a lógica do jogo e decide se deve mostrar o resultado ou fazer a próxima pergunta.

```

184     private void askNextQuestion() {
185         if (gameLogic.canAskMoreQuestions()) {
186             String question = gameLogic.getNextQuestion();
187             questionLabel.setText(question);
188         } else {
189             showResult();
190         }
191     }

```

Obtém e exibe a próxima pergunta ou mostra o resultado se não houver mais perguntas.

```

193 private void showResult() {
194     JPanel resultPanel;
195     if (gameLogic.hasDiscoveredCharacter()) {
196         resultPanel = createDiscoveredCharacterPanel();
197     } else if (gameLogic.getRemainingCharacters().size() > 1) {
198         resultPanel = createMultipleCharactersPanel();
199     } else {
200         resultPanel = createRemainingCharactersPanel();
201     }
202
203     JPanel endGamePanel = new JPanel(new BorderLayout());
204     endGamePanel.setBackground(BACKGROUND_COLOR);
205     endGamePanel.add(resultPanel, BorderLayout.CENTER);
206     endGamePanel.add(createStyledButton(text:"Jogar Novamente", BUTTON_COLOR, e → startGame()), BorderLayout.SOUTH);
207
208     setContentPane(endGamePanel);
209     revalidate();
210     repaint();
211 }

```

Mostra o resultado do jogo, escolhendo entre diferentes painéis de resultado dependendo do desfecho.

```

213 private JPanel createMultipleCharactersPanel() {
214     JPanel panel = new JPanel();
215     panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
216     panel.setBackground(BACKGROUND_COLOR);
217     panel.setBorder(new EmptyBorder(top:20, left:20, bottom:20, right:20));
218
219     JLabel titleLabel = createStyledLabel(text:"Não foi possível definir um único personagem.", QUESTION_FONT,
220     TEXT_COLOR);
221     titleLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
222     panel.add(titleLabel);
223
224     panel.add(Box.createRigidArea(new Dimension(width:0, height:20)));
225
226     JLabel subtitleLabel = createStyledLabel(text:"Os personagens mais próximos são:", SUBTITLE_FONT, TEXT_COLOR);
227     subtitleLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
228     panel.add(subtitleLabel);
229
230     panel.add(Box.createRigidArea(new Dimension(width:0, height:20)));
231
232     JPanel charactersPanel = new JPanel();
233     charactersPanel.setLayout(new BoxLayout(charactersPanel, BoxLayout.Y_AXIS));
234     charactersPanel.setBackground(BACKGROUND_COLOR);
235
236     List<Features> remainingCharacters = gameLogic.getRemainingCharacters();
237     for (Features character : remainingCharacters) {
238         JLabel characterLabel = createStyledLabel("- " + character.getName(), CHARACTER_LIST_FONT, TEXT_COLOR);
239         characterLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
240         charactersPanel.add(characterLabel);
241         charactersPanel.add(Box.createRigidArea(new Dimension(width:0, height:10)));
242     }
243
244     JScrollPane scrollPane = new JScrollPane(charactersPanel);
245     scrollPane.setBorder(BorderFactory.createEmptyBorder());
246     scrollPane.setBackground(BACKGROUND_COLOR);
247     scrollPane.getViewport().setBackground(BACKGROUND_COLOR);
248     panel.add(scrollPane);
249
250     // Adicionar informação sobre o número de personagens restantes
251     JLabel countLabel = createStyledLabel("Total de personagens restantes: " + remainingCharacters.size(),
252     SUBTITLE_FONT, TEXT_COLOR);
253     countLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
254     panel.add(Box.createRigidArea(new Dimension(width:0, height:20)));
255     panel.add(countLabel);
256
257     return panel;
258 }

```

Cria um painel para exibir múltiplos personagens quando o jogo não consegue determinar um único personagem.

```

259 private JPanel createDiscoveredCharacterPanel() {
260     JPanel panel = new JPanel();
261     panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
262     panel.setBackground(BACKGROUND_COLOR);
263     panel.setBorder(new EmptyBorder(top:20, left:20, bottom:20, right:20));
264
265     try {
266         panel.add(createCharacterImageLabel());
267     } catch (Exception e) {
268         System.err.println("Erro ao carregar a imagem: " + e.getMessage());
269     }
270
271     panel.add(Box.createRigidArea(new Dimension(width:0, height:20)));
272     String characterName = gameLogic.getDiscoveredCharacter().getName();
273     panel.add(createStyledLabel("O personagem em que você está pensando é " + characterName + "!",
274         QUESTION_FONT, TEXT_COLOR));
275
276     return panel;
277 }

```

Cria um painel para exibir o personagem descoberto quando o jogo adivinha corretamente.

```

279 private JLabel createCharacterImageLabel() {
280     String characterName = gameLogic.getDiscoveredCharacter().getName().toLowerCase();
281     String[] possibleExtensions = { ".jpeg", ".jpg", ".png" };
282     String imagePath = null;
283
284     for (String extension : possibleExtensions) {
285         String potentialPath = IMAGE_PATH + characterName + extension;
286         if (new File(potentialPath).exists()) {
287             imagePath = potentialPath;
288             break;
289         }
290     }
291
292     if (imagePath == null) {
293         System.err.println("No image found for character: " + characterName);
294         imagePath = IMAGE_PATH + "default.png";
295     }
296
297     ImageIcon characterImage = new ImageIcon(imagePath);
298     Image image = characterImage.getImage();
299     Image resizedImage = image.getScaledInstance(width:300, height:400, Image.SCALE_SMOOTH);
300     characterImage = new ImageIcon(resizedImage);
301
302     JLabel characterImageLabel = new JLabel();
303     characterImageLabel.setIcon(characterImage);
304     characterImageLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
305     return characterImageLabel;
306 }

```

Cria um rótulo com a imagem do personagem descoberto, buscando a imagem correspondente ou usando uma imagem padrão.

```

308     private JPanel createRemainingCharactersPanel() {
309         JPanel panel = new JPanel(new BorderLayout());
310         panel.setBackground(BACKGROUND_COLOR);
311
312         JTextArea remainingCharactersArea = new JTextArea();
313         remainingCharactersArea.setFont(CharacterListFont);
314         remainingCharactersArea.setEditable(false);
315         remainingCharactersArea.setBackground(BACKGROUND_COLOR);
316
317         StringBuilder sb = new StringBuilder(
318             str: "Não foi possível determinar o personagem com certeza. Os personagens restantes são:\n\n");
319         for (Features character : gameLogic.getRemainingCharacters()) {
320             sb.append(str: "- ").append(character.getName()).append(str: "\n");
321         }
322         remainingCharactersArea.setText(sb.toString());
323
324         JScrollPane scrollPane = new JScrollPane(remainingCharactersArea);
325         scrollPane.setBorder(BorderFactory.createEmptyBorder());
326         panel.add(scrollPane, BorderLayout.CENTER);
327
328         return panel;
329     }
330 }

```

Cria um painel para exibir os personagens restantes quando o jogo não consegue determinar um único personagem e há poucos personagens restantes.

Classe GameLogic.java

A classe GameLogic gerencia a lógica do jogo, incluindo a seleção de perguntas e o processamento de respostas.

```
17 public GameLogic(List<Features> characters) {
18     this.originalCharacterList = new ArrayList<>(characters);
19     resetGame();
20 }
```

Este construtor inicializa a lógica do jogo.

```
22 public void resetGame() {
23     currentCharacterList = new ArrayList<>(originalCharacterList);
24     questionsAsked = new HashSet<>();
25     discoveredCharacter = null;
26     lastQuestionAsked = null;
27     characterDiscovered = false;
28     updateRelevantQuestions();
29 }
```

Este método reinicia o jogo, resetando todas as variáveis para seu estado inicial.

```
39 public void processAnswer(boolean answer) {
40     if (!relevantQuestions.isEmpty()) {
41         Question currentQuestion = relevantQuestions.remove(index:0);
42     }
43     if (!characterDiscovered) {
44         Filter1.filtrarPersonagens(currentCharacterList, currentQuestion.getFeatures(), answer);
45     }
46     questionsAsked.add(currentQuestion.getPergunta());
47     if (currentCharacterList.size() == 1 && !characterDiscovered) {
48         discoveredCharacter = currentCharacterList.get(index:0);
49         characterDiscovered = true;
50         updateRelevantQuestions(); // Atualiza as perguntas para serem específicas do personagem descoberto
51     }
52     printQuestionAndRemainingCharactersInfo(answer);
53 }
54 }
55 }
56 }
57 }
```

Este método processa a resposta do usuário:

1. Remove a pergunta atual da lista de perguntas relevantes.
2. Filtra a lista de personagens com base na resposta.
3. Adiciona a pergunta à lista de perguntas feitas.
4. Verifica se um personagem foi descoberto.
5. Atualiza as perguntas relevantes se necessário.

```
59 private void updateRelevantQuestions() {
60     if (characterDiscovered) {
61         relevantQuestions = Question.obterPerguntasEspecificasPersonagem(discoveredCharacter, questionsAsked);
62     } else {
63         relevantQuestions = Question.obterPerguntasRelevantes(currentCharacterList, questionsAsked);
64     }
65 }
```

Este método atualiza a lista de perguntas relevantes. Se um personagem foi descoberto, ele obtém perguntas específicas para esse personagem. Caso contrário, obtém perguntas relevantes para todos os personagens restantes.

```
67 public boolean isGameOver() {
68     return questionsAsked.size() >= 15 || relevantQuestions.isEmpty();
69 }
```

verifica se o jogo terminou


```

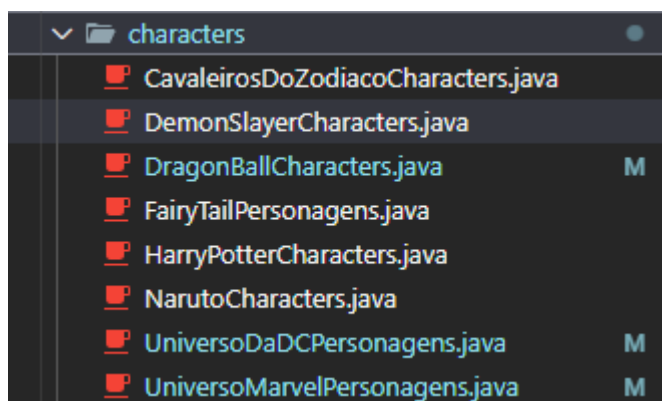
71 public boolean canAskMoreQuestions() {
72     return !relevantQuestions.isEmpty() && questionsAsked.size() < 15;
73 }
74
75 public boolean hasDiscoveredCharacter() {
76     return characterDiscovered;
77 }
78
79 public Features getDiscoveredCharacter() {
80     return discoveredCharacter;
81 }
82
83 public List<Features> getRemainingCharacters() {
84     return currentCharacterList;
85 }
86
87 public int getQuestionsAskedCount() {
88     return questionsAsked.size();
89 }
90
91 private void printQuestionAndRemainingCharactersInfo(boolean answer) {
92     System.out.println(x:"\n— Pergunta respondida —");
93     System.out.println("Pergunta: " + lastQuestionAsked);
94     System.out.println("Resposta: " + (answer ? "Sim" : "Não"));
95     System.out.println(x:"—————");
96     System.out.println(x:"— Informações dos personagens restantes —");
97     System.out.println("Quantidade de personagens restantes: " + currentCharacterList.size());
98     System.out.println(x:"Personagens restantes:");
99     for (Features character : currentCharacterList) {
100         System.out.println("- " + character.getName());
101     }
102     System.out.println(x:"—————\n");
103 }
104 }

```

- verifica se ainda é possível fazer mais perguntas
- retorna o personagem descoberto
- retorna a lista atual de personagens restantes
- retorna o número de perguntas já feitas durante o jogo atual.

imprimir informações no console. mostra:

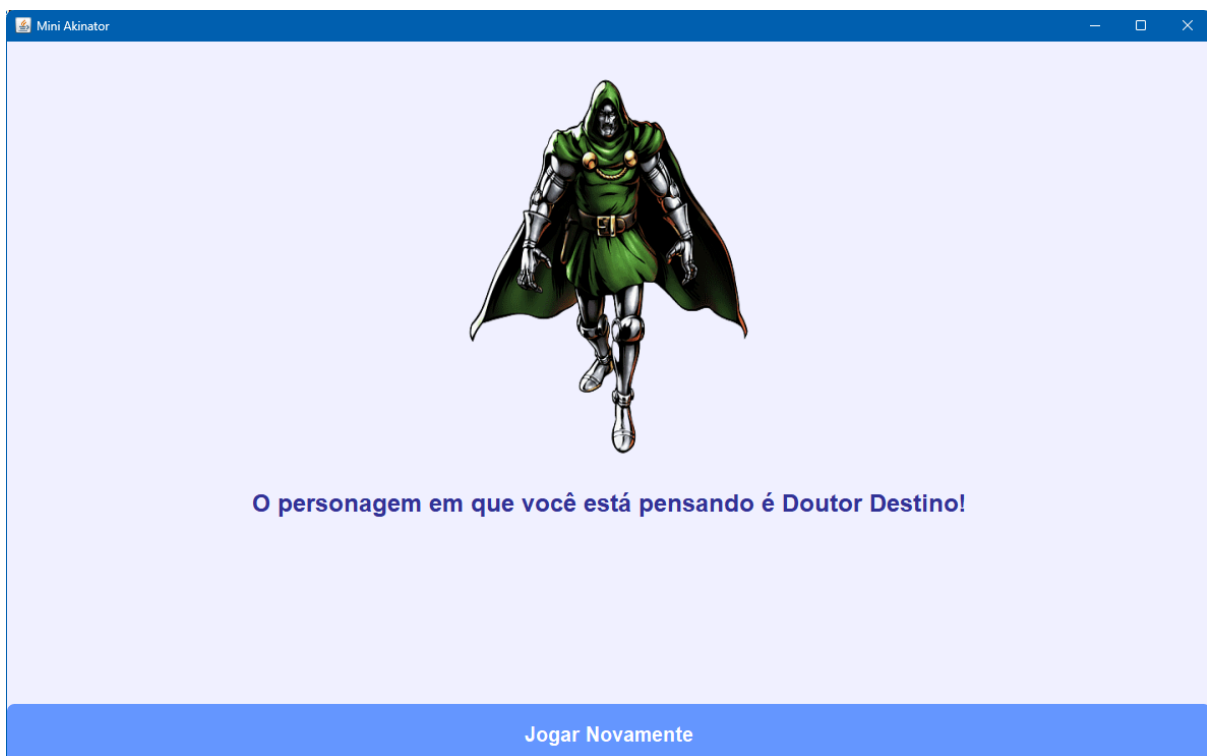
1. A última pergunta feita e a resposta dada.
2. O número de personagens restantes.
3. Uma lista dos nomes dos personagens restantes.



Neste arquivo cria e adiciona personagens dos universos, cada personagem é criado como um objeto **Features**, que representa suas características e habilidades. Após a criação, atributos específicos são adicionados usando o método **addSpecificAttribute**.

No final de cada arquivo, todos os personagens criados são adicionados a uma lista chamada `listaPersonagens`, que é usada para o registro de todos os personagens disponíveis.

Teste



Não foi possível definir um único personagem.

Os personagens mais próximos são:

- Steve Rogers (Capitão América)
- T'Challa (Pantera Negra)

Total de personagens restantes: 2

Jogar Novamente