

# Relatório Trabalho Prático

## Sistema Pericial para diagnóstico de pacientes

# Índice

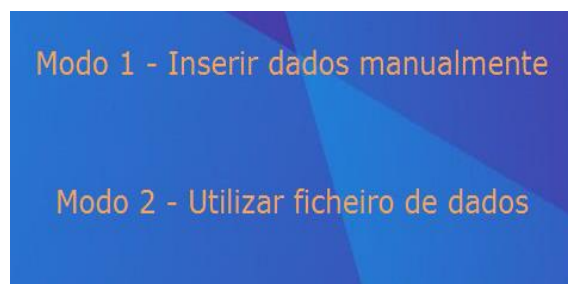
|  |    |
|--|----|
| Interface .....  | 3  |
| Diagrama de estados do sistema pericial.....             | 5  |
| Parte Lógica .....                                       | 6  |
| Organização do projeto.....                              | 6  |
| Modo um.....   | 8  |
| Funcionamento .....                                      | 8  |
| Guardar .....  | 9  |
| Modo dois .....  | 10 |
| Funcionamento .....                                      | 10 |
| Carregar ficheiro de dados.....                          | 10 |
| Executar testes.....                                     | 11 |
| Análise de Variáveis, Regras e diagramas de Regras ..... | 13 |
| Variáveis para seleccionar pacientes: .....              | 13 |
| Inclusão .....   | 13 |
| Exclusão .....   | 14 |
| Objetivo.....  | 14 |
| Diagrama de fluxo de Regras.....                         | 15 |
| Testes .....   | 17 |
| Anexos .....   | 18 |
| Conclusão.....   | 19 |

# Interface

O sistema pericial conta com uma interface gráfica, desenvolvida com recurso à biblioteca Javafx, de modo a garantir uma melhor interação com o utilizador e erradicar o processo, moroso, de aprendizagem e introdução de comandos.

O sistema pericial pode passar por 3 estados na sua execução estando cada um deles representado abaixo nas imagens.

Devido ao pouco tempo que restava, não conseguimos produzir a parte de interface que permite ao utilizador responder às perguntas extra, feitas pelo sistema pericial, que concluem se um paciente deve fazer exames extra. Estas perguntas extra são colocadas e respondidas pelo utilizador na consola.



**Figura 1-Ecrã inicial**

O ecrã inicial apenas permite ao utilizador seleccionar de que modo pretende usar os dados de um paciente, o Modo1 permite a inserção de um único passageiro de cada vez enquanto o Modo2 permite carregar um ficheiro que contém as fichas de vários pacientes.

Aqui presente, visualizamos o Modo1, o utilizador preenche os campos com os dados do paciente e carregando no botão “Confirmar” faz com que o paciente seja testado pelas regras.

Vários pacientes podem ser adicionados neste estado visualizando-se o resultado do teste efetuado a cada um no quadro branco no canto inferior esquerdo.



Figura 2-Modo um.

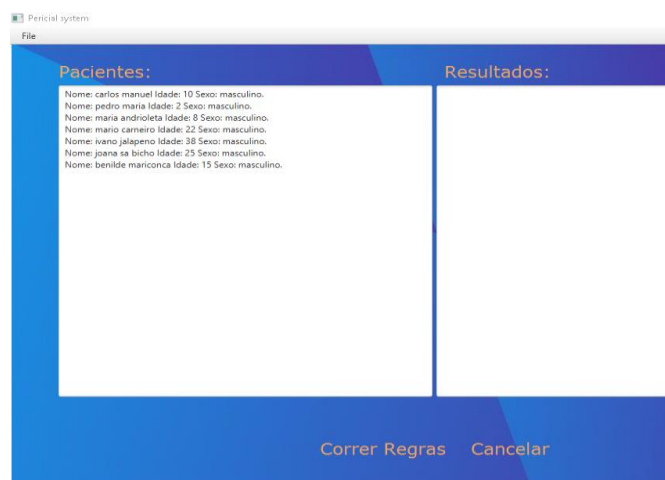
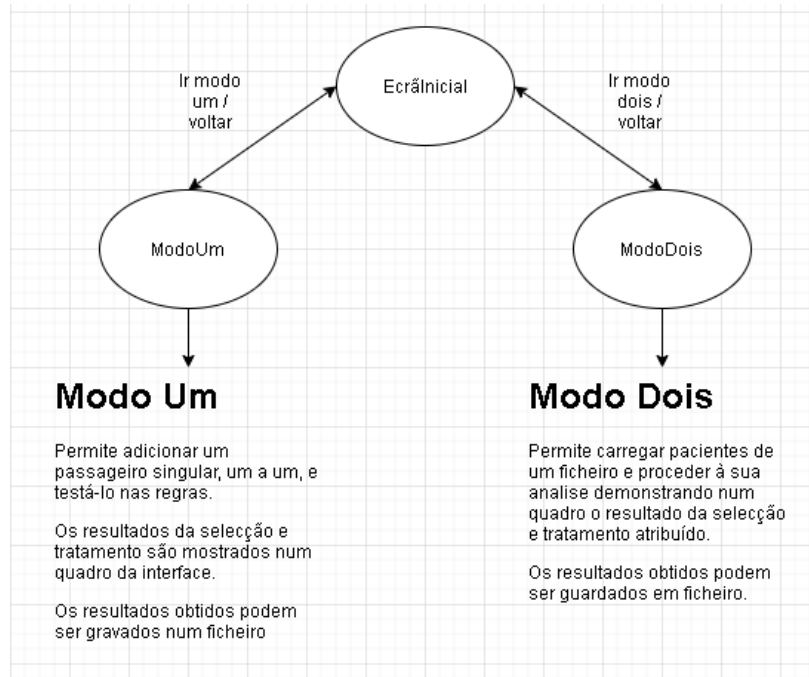


Figura 3-Modo dois.

No quadro da esquerda pode-se observar a informação dos pacientes carregados do ficheiro e no quadro da direita verifica-se se foram selecionados ou não para o tratamento.

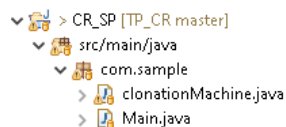
Nesta imagem podemos visualizar o aspeto do modo dois. A barra de menu que se encontra na parte de cima da janela permite abrir um ficheiro com as fichas dos pacientes, ou guardar os resultados obtidos.

## Diagrama de estados do sistema pericial



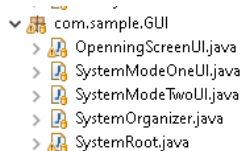
## Parte Lógica

### Organização do projeto

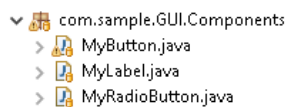


- Pasta que contém os métodos “main” da classe que inicia o sistema pericial (Main.java) e de uma classe (clonationMachine.java) que nos permitiu criar vários pacientes com parâmetros aleatórios para a realização de testes.

**O sistema pericial desenvolvido deve executar-se a partir da Main.java e não em clonationMachine.java visto que são programas diferentes e com objetivos distintos.**

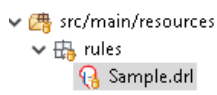


- Pasta relacionada com a interface, nela estão todas classes necessárias para a construção da interface de todos os modos de trabalho.



“Components”.

- Para a interface criámos o nosso próprio tipo de botões e *labels* para tornar a interface mais apelativa ao utilizador, estas classe encontram-se na pasta



- Esta pasta foi criada aquando a criação do projeto Drools e apenas contém o ficheiro .drl que contém as regras do sistema pericial.

▼ com.sample.model.data  
    > InferenceEngine.java  
    > ObservableModel.java  
    > Patient.java  
    > PericialSystem.java  
    > SystemData.java

- Dividiu-se a parte lógica da interface e as classes que tratam da lógica do sistema encontram-me na pasta “data”.

“Patient” representa cada paciente a analisar;

“SystemData” tem o papel de manobrar e deter os dados de trabalho armazenados e tem o papel de carregar pacientes de um ficheiro e guardar resultados em ficheiro;

“ParicialSystem” é a classe base do sistema pericial, ela é criada aquando o início de execução e responsável pela criação e ordenamento de instruções a “SystemData” e “InferenceEngine”;

“InferenceEngine” possui uma visão sobre “SystemData” e é a classe responsável pela inserção de pacientes na memória de trabalho e pela execução das regras;

“ObservableModel” trata-se de um observador da interface que reporta ao “PericialSystem” quando ocorre um evento.

▼ com.sample.model.states  
    > AdapterState.java  
    > ModeOneState.java  
    > ModeTwoState.java  
    > OpeningScreenState.java  
    > SystemState.java

- paste “states” contém todas as classes correspondentes aos estados do sistema pericial desenvolvido.

## Modo um

### Funcionamento

O modo de trabalho número um (Modo um), representado na **Figura 2**, permite fazer uma ficha de paciente com os dados introduzidos pelo utilizador, todos os dados necessários à seleção de pacientes para o tratamento estão presentes.

Todos os campos são de introdução obrigatória para que o desenrolar do processo de seleção decorra sem erros, o campo que permite indicar se o paciente corresponde a uma grávida ou lactente só é possível ser marcado caso o sexo escolhido naquele momento seja feminino.

Após o preenchimento dos campos é possível carregar no botão “Confirmar” e daí começa o processo de teste do paciente. Abaixo será explicado todo o processo de seleção até à obtenção do resultado a partir do clique.

```
private void registaListeners() {  
    confirms.setOnAction(new EventHandler<ActionEvent>() {  
        @Override  
        public void handle(ActionEvent event) {  
  
            Patient newPatient = new Patient();  
  
            if(getPatientDataFromScreen(newPatient) == false) {  
                Alert alerta = new Alert(AlertType.ERROR);  
                alerta.setTitle("Erro");  
                alerta.setContentText("Ocorreu um erro ao introduzi");  
                alerta.showAndWait();  
            }  
            else {  
                observableModel.testPatient(newPatient);  
                observableModel.savePatientData(newPatient);  
                results.setText(observableModel.getTextAreaContent());  
                results.setScrollTop(Double.MAX_VALUE);  
            }  
        }  
    });  
}
```

Quando o processo é iniciado, o programa vai buscar os valores presentes nos campos da interface gráfica com a função

“getPatientDataFromScreen”, caso algum campo esteja preenchido de forma irregular um alerta é apresentado ao utilizador, se todos os dados de

entrada estiver em conformidade com o que é necessário, eles serão atribuídos ao paciente “newPatient” que será enviado para teste no método “testPatient”.



Por indicação do “ObservableModel”, o sistema pericial dá ordem à classe “InferenceEngine” para testar o paciente nas regras guardando os dados do paciente na memória do sistema (SystemData) e colocando a informação resultante do teste numa String que será mostrada no quadro de resultados, o método “savePatientData” guarda e coloca a informação na String e é mostrada abaixo.

```
public void savePatientData(Patient patient) {  
    patients.add(patient);  
  
    textAreaContent = textAreaContent.concat(patients.{  
        if(patient.getTreatment() != null) {  
            if(patient.getTreatment() == "") patient.setTre  
            textAreaContent = textAreaContent.concat("Extra  
        }  
    }  
}
```

## Guardar

A interface gráfica permite ao utilizador, caso ele pretenda, guardar os resultados obtidos dos pacientes que testou durante aquela execução do modo um, para isto basta carregar no botão guardar.

```
saveData.setOnAction(new EventHandler<ActionEvent>() {  
    @Override  
    public void handle(ActionEvent event) {  
        File f = fileChooser.showSaveDialog(getScene()).  
        if (f != null) {  
            observableModel.saveResultsOnFile(f.getAbsolutePath()  
            System.out.println("funcionou");  
        }  
    }  
});
```

Depois que o clique do botão se verifique o utilizador escolhe a localização e nome do ficheiro, terminando o nome em .bin, de seguida a classe “PericialSystem” é alertada e comanda “SystemData” a executar o método “saveResultsOnFile” que guarda no ficheiro os dados de todos os pacientes introduzidos e testados naquela execução.

```
public void saveResultsOnFile(String filePath){  
    try {  
        FileOutputStream fileOutputStream = new FileOutputStream(  
        ObjectOutputStream objectOutputStream = new ObjectOutputStream(  
  
        int i=0;  
        while(i < patients.size()) {  
            objectOutputStream.writeObject(patients.get(i));  
            i++;  
        }  
  
        objectOutputStream.close();  
    } catch (Exception ex) {  
        ex.printStackTrace();  
    }  
}
```

## Modo dois

### Funcionamento

Na Figura 3 pode-se observar o aspeto da interface gráfica apresentada no modo de trabalho dois, este modo tem como objetivo carregar um ficheiro com dados de pacientes que se pretendem testar, concluir, e consequentemente mostrar ao utilizador se estes são selecionados para estudo e que tipo de tratamentos extras devem fazer, caso necessitem.

### Carregar ficheiro de dados

Para carregar um ficheiro de dados, no modo dois, existe um menu na parte superior da interface que permite selecionar as opções “Abrir” e “Guardar”, selecionar “Abrir” faz correr o código seguinte.

```
// ...  
fileOpen.setOnAction(  
    (ActionEvent e) -> {  
        File f = fileChooser.showOpenDialog(getScene().getWindow())  
        if (f != null) {  
            // METODO DO OBSERVAVEL PARA ABRIR O FICHEIRO DE PACIENTES  
            observableModel.openPatientsFile(f.getAbsolutePath());  
            pacientes.setText(observableModel.getTextAreaContent());  
            pacientes.setScrollTop(Double.MAX_VALUE);  
        }  
    }  
);
```

Analisando o código, uma janela é aberta e aguarda que o utilizador selecione o ficheiro pretendido.

Após a seleção o método “ObservableModel” alerta o “PericialSystem” que se pretende abrir um ficheiro com o método o “openPatientsFile”, o “PericialSystem” ordena à “SystemData” que o conteúdo do ficheiro seja lido e gravado na memória do sistema e que a informação de cada paciente seja guardada em String para apresentar no quadro do lado esquerdo da interface.

Os dados dos pacientes que estavam no ficheiro, depois de todo este processo, são então demonstrados no quadro esquerdo da interface.

```
public void openPatientsFile(String filePath) {  
    try {  
        FileInputStream fileInputStream = new FileInputStream(filePath);  
        ObjectInputStream objectInputStream = new ObjectInputStream(f  
  
        while(fileInputStream.available() != 0) {  
            Patient obj = (Patient)objectInputStream.readObject();  
            patients.add(obj);  
            textAreaContent = textAreaContent.concat(patients.get(pat  
        }  
  
        int i = 0;  
        while(i < patients.size()) {  
            System.out.println("Paciente "+ patients.get(i).getFirstN  
            i++;  
        }  
  
        objectInputStream.close();  
    } catch (Exception ex) {  
        ex.printStackTrace();  
    }  
}
```

## Executar testes

Após o carregamento do ficheiro, o sistema já tem guardados os dados dos pacientes que se pretendem testar, carregando no botão “Correr Regras” faz com que se inicie o processo de teste “startInferenceEngine”.

```
btnConfirma.setOnAction(new EventHandler<ActionEvent>() {  
    @Override  
    public void handle(ActionEvent event) {  
        observableModel.startInferenceEngine();  
        observableModel.loadPatientData();  
        resultados.setText(observableModel.getTextAreaContent  
        resultados.setScrollTop(Double.MAX_VALUE);  
    }  
});
```

A classe “InferenceEngine” executa o método “fireRules” adicionando cada paciente à memória de trabalho da sessão e iniciando todas as regras desenvolvidas para a seleção de pacientes.

```
public void fireRules() {  
    try {  
        // load up the knowledge base  
        KieServices ks = KieServices.Factory.get();  
        KieContainer kContainer = ks.getKieClasspathContainer();  
        KieSession kSession = kContainer.newKieSession("ksession-rul  
  
        int i = 0;  
  
        while(i < systemData.getNumberOfPatients()){  
            kSession.insert(systemData.getPatientByIndex(i));  
            i++;  
        }  
  
        kSession.fireAllRules();  
        kSession.dispose();  
    } catch (Throwable t) {  
        t.printStackTrace();  
    }  
}
```

Depois da execução das regras, cada paciente por este andar já deverá ter algum tipo de tratamento atribuído ou saber se foi selecionado para o tratamento, “loadPatientData” coloca na String pertencente a “SystemData” toda a nova informação dos pacientes que depois é mostrada no quadro direito da interface gráfica.

Os resultados obtidos podem ser guardados para analisar mais tarde num ficheiro de texto com recurso ao item “Guardar” do menu na parte superior da interface. //print com os resultados

## Análise de Variáveis, Regras e diagramas de Regras

### Variáveis para selecionar pacientes:

#### Inclusão

A=Idade < 18 anos;

B=Indicador de desempenho da Organização Mundial da Saúde (OMS) 0;

C=Indicador de desempenho da Organização Mundial da Saúde (OMS) 1;

D= Diagnóstico de carcinoma espinocelular,

E=carcinoma indiferenciado

F=Siewert I

G=Siewert II

Z= D and E and F and G

H=Doença clínica em estágio II

I=Doença clínica em estágio III

J=Borda superior do tumor pelo menos 3 cm abaixo do esfíncter superior do esôfago;

K=termo de consentimento;

## Exclusão

L=Mulheres grávidas ou lactantes;

M=Radioterapia torácica prévia;

N= Contagem de neutrófilos  $< 1,5 \times 10^9 / L$ ,

O= Contagem de plaquetas  $< 100 \times 10^9 / L$ ,

P= Concentração sérica de bilirrubina total  $> 1,5 \times \text{LSN}$  (LSN = Limite Superior Normal),

Q= Creatinina  $> 120 \text{ mcmol} / L$ ,

R= Controlo biológico da função pulmonar  $\text{FEV1} < 1,5 L$ ;

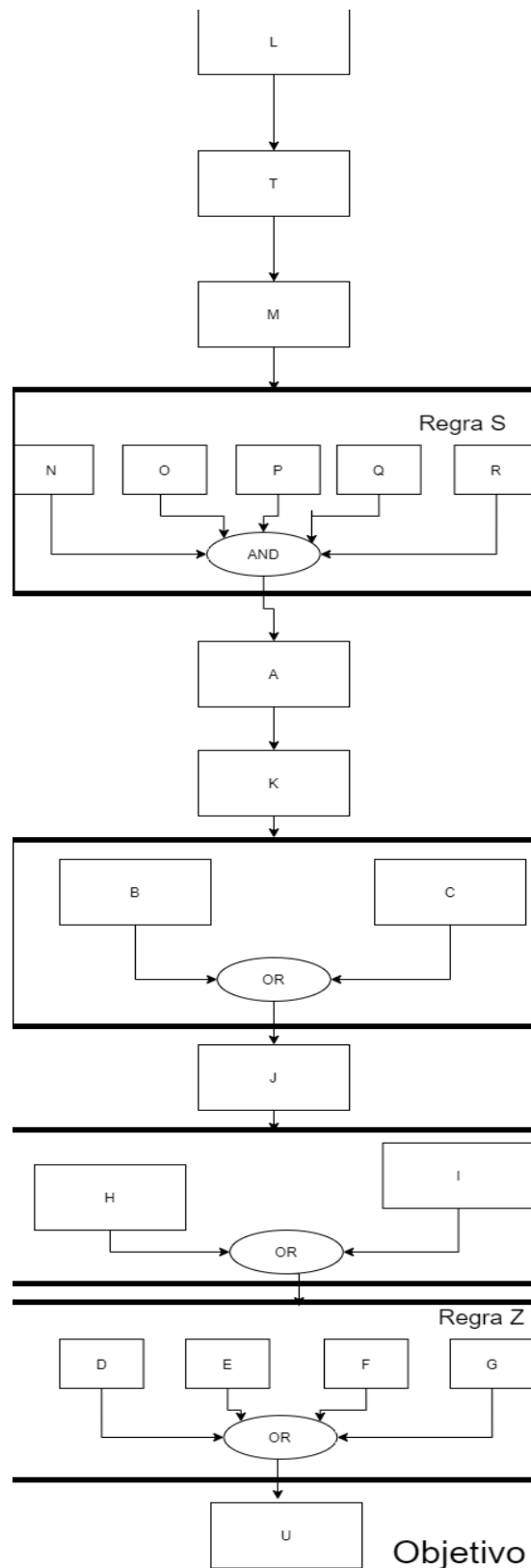
S= M or N or O or P or Q

T=Infeção ativa ou outra condição médica que impeça o paciente de receber o tratamento planejado.

## Objetivo

U=Selecionado para exame

## Diagrama de fluxo de Regras



A partir da Regra U e obtendo mais variáveis a partir do utilizador o sistema pericial, mostra ao utilizador os exames que este terá que realizar.

As possíveis variáveis de entrada para a seleção dos exames são as seguintes:

A-tumor relacionado com a árvore respiratória

B-Lesões endoscópicas

C-Estômago incluído no volume de tratamento

E os exames a que as variáveis correspondem são os seguintes:

D-Efetuar Broncoscopia

E-Efetuar Endoscopia

F-Efetuar Jejum

If A then D

If B then E

If C then F

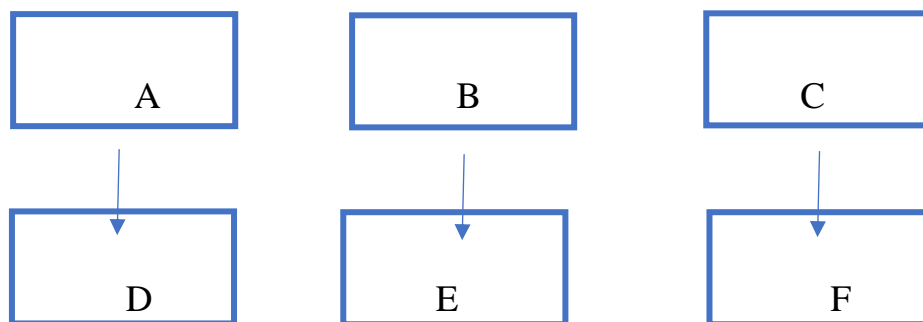
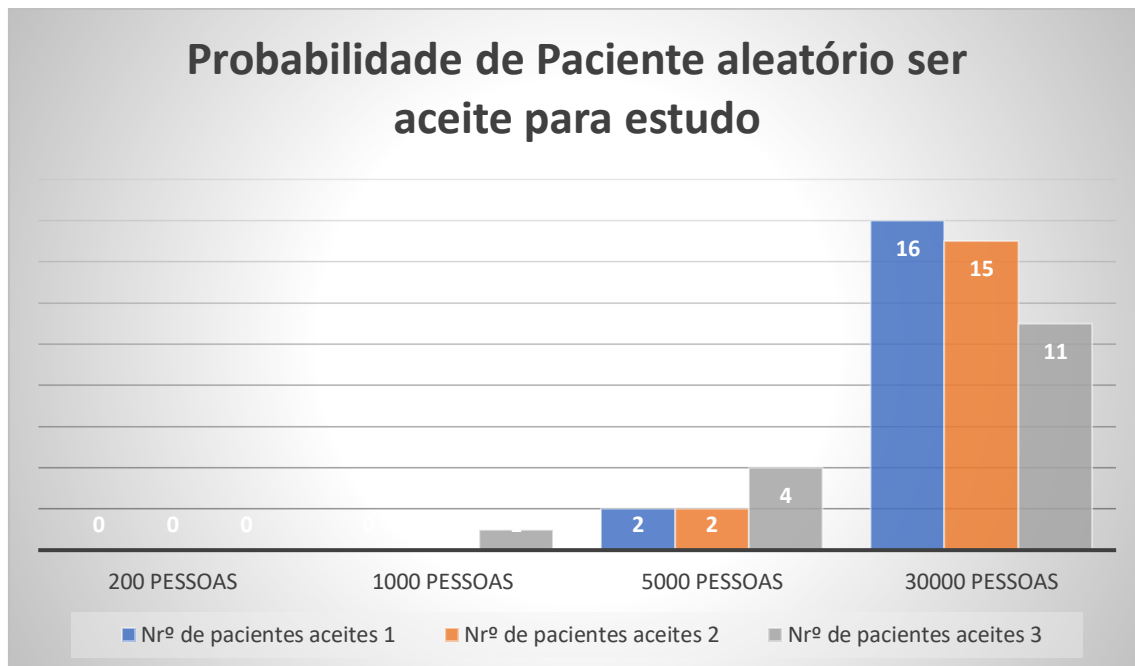


Figura 2 – Inputs e outputs dos exames que o utilizador necessita de realizar



## Testes



Podemos concluir que gerar dados aleatórios para criar pacientes, a probabilidade de estes serem aceites no estudo é bastante baixa, sendo aproximadamente 1 Paciente a cada 1500.

## Anexos

Em anexo enviamos um ficheiro excel com 18 pacientes, que foram testados manualmente para que cada uma das regras fosse testada.

Podemos concluir que dos 18 pacientes inseridos, 9 foram admitidos para o estudo e outros 9 não foram.

Dos 9 Pacientes que reprovaram a admissão foi devido às várias causas possíveis, como infeção ativa, radioterapia prévia, idade, gravidez, funções corporais comprometidas, distância do tumor ao esófago insuficiente, entre outras.

Aos restantes pacientes, que foram aceites, foi lhes pedido informação adicional para verificar a necessidade de exames extra, e assim foram verificadas as combinações possíveis para a endoscopia, broncoscopia e jejum.

## Conclusão

Concluimos que os sistemas periciais conseguem, não sendo a mesma a coisa, fazer diagnósticos com base na experiência de profissionais, podem enquadrar-se em várias áreas profissionais e são de extrema utilidade porque permitem tirar conclusões e obter resultados de forma rápida e eficiente.