



Universidade do Minho
Escola de Engenharia

Relatório do Projeto

Simulação de uma Clínica Médica

Licenciatura em Engenharia Biomédica

Grupo XX:

A107195, Clara Carvalho
A107234, Andreia Ferreira
A107193, Bianca Pereira

Docentes:
José Ramalho
Luís Cunha

Braga, 8 de janeiro de 2026



Índice

1	Introdução	1
2	Arquitetura do Sistema e Estrutura de Dados	1
2.1	Organização Modular (Arquitetura de <i>Software</i>)	1
2.2	Estruturas de Dados e Persistência	2
2.2.1	Estruturas de Dados e Persistência	2
3	Modelo de Simulação e Algoritmos	3
3.1	Modelação das Chegadas (Processo de Poisson)	3
3.2	Tempos de Atendimento	3
3.3	Algoritmo de Gestão da Fila e Prioridades	4
3.4	Atribuição por Especialidade	5
4	Interface Gráfica e Funcionalidades	6
4.1	Autenticação de Administrador	6
4.2	Menu Principal	7
4.3	Importação de Dados	7
4.4	Tratamento de Erros e Validação	8
4.5	Gestão e Alteração de Dados	8
4.6	Ambiente de Simulação	10
4.6.1	Janela de Configuração	10
4.6.2	Monitorização em Tempo Real	10
4.7	Análise de Resultados e Ações	11
4.7.1	Visualização Gráfica e Estatística	11
4.7.2	Comparação de Simulações	12
4.7.3	Lista Detalhada de Atendimentos	13
4.7.4	Estatísticas por Especialidade	15
4.7.5	Análise de Desempenho dos Médicos	16
4.7.6	Análise de Correlação: Fila vs. Taxa de Chegada	17
5	Funções Utilizadas e Lógica de Processamento	19
5.1	Gestão de Ficheiros e Persistência	19
5.2	Estrutura de Fila (FIFO)	19
5.3	Lógica de Limpeza de Estado	20
5.4	Formatação e Sanitização de Dados	20
5.5	Geração de Relatórios e Exportação	21
5.5.1	Manipulação de dados	21
6	Conclusão	21
7	Anexos	22



1 Introdução

O presente relatório descreve o desenvolvimento e a implementação de um sistema de simulação para uma clínica médica, desenvolvido no âmbito da unidade curricular de Algoritmos e Técnicas de Programação. O objetivo principal do projeto é modelar e analisar o fluxo de atendimento de pacientes, permitindo compreender a dinâmica entre a chegada de doentes, os tempos de espera e a eficiência do corpo clínico.

A simulação de sistemas de saúde é uma ferramenta fundamental na engenharia, pois permite testar cenários e prever estrangulamentos no atendimento sem a necessidade de intervenções reais dispendiosas ou arriscadas. Para este efeito, o *software* desenvolvido utiliza modelos probabilísticos, como a distribuição de Poisson para a modelação das chegadas e distribuições variadas (Exponencial, Normal e Uniforme) para os tempos de consulta.

Para além do motor de simulação, o projeto integra uma interface gráfica robusta que permite a gestão de dados de pacientes e médicos através de ficheiros JSON, bem como a visualização dinâmica de resultados através de gráficos estatísticos. Este trabalho foca-se não apenas na precisão algorítmica, mas também na modularidade do código e na experiência do utilizador final.

2 Arquitetura do Sistema e Estrutura de Dados

A robustez do simulador assenta numa arquitetura modular, onde a lógica de negócio, a interface de utilizador e a persistência de dados estão claramente separadas. Esta abordagem facilita a manutenção do código e a escalabilidade do sistema.

2.1 Organização Modular (Arquitetura de *Software*)

O sistema foi dividido em módulos principais, cada um com responsabilidades específicas, promovendo a reutilização de código:

- **main.py**: Atua como o orquestrador do sistema. Gere o arranque da aplicação, o ciclo de vida da interface e o controlo de acessos (*login*).
- **admin.py**: Responsável pela camada de segurança, gerindo a autenticação de utilizadores administrativos através da verificação de credenciais no ficheiro *usersadmin.json*.
- **funcoes.py**: O núcleo lógico do projeto. Contém as definições das filas (operações *enqueue* e *dequeue*), algoritmos de ordenação por prioridade, geradores aleatórios e funções de manipulação de ficheiros.
- **interface_menu.py**: Gere a navegação principal, permitindo ao utilizador escolher entre a importação de dados, alteração de registos ou configuração da simulação.
- **interface_simulacao.py**: Implementa a componente visual dinâmica, utilizando a biblioteca *FreeSimpleGUI* para mostrar a evolução da fila e o estado dos médicos em tempo real.



2.2 Estruturas de Dados e Persistência

A informação é armazenada e manipulada utilizando o formato **JSON**, garantindo interoperabilidade e facilidade de edição manual se necessário. As principais entidades são:

2.2.1 Estruturas de Dados e Persistência

A informação é armazenada e manipulada utilizando o formato **JSON**, garantindo interoperabilidade e facilidade de edição. As principais entidades são:

- **Pacientes (`pacientes.json`):** Cada registo inclui ID, nome, idade, doença e o nível de prioridade clínica e atributos clínicos. No código, estes dados são processados como listas de dicionários:

```
1 # Exemplo da estrutura de um paciente apos carregamento
2 paciente = {
3     {
4         "id": "d1",
5         "nome": "Neyanne Sampaio",
6         "idade": 47,
7         "sexo": "feminino",
8         "doenca": "Gripe",
9         "prioridade": "alta",
10        "atributos": {
11            "fumador": false,
12            "consome_alcool": false,
13            "atividade_fisica": "moderada",
14            "cronico": false
15        }
16    }
```

Listing 1: Representação interna de um paciente em dicionário Python

- **Médicos (`medicos.json`):** Armazena a identificação do clínico e a sua especialidade. O sistema utiliza um mapeamento dinâmico (*Mapeamento de Doenças*) para garantir o encaminhamento correto:

```
1 # Exemplo de mapeamento para encaminhamento especializado
2 mapeamento = {
3     "Arritmia": "Cardiologia",
4     "Gripe": "Clinica Geral",
5     "Fratura": "Ortopedia"
6 }
```

Listing 2: Dicionário de mapeamento para triagem por especialidade

```
1 # "medicos = [{
2     "id": "m1",
3     "nome": "Dr. Ant nio Silva",
```



```
4         "ocupado": false,  
5         "doente_corrente": null,  
6         "especialidade": "Clínica Geral",  
7         "total_tempo_ocupado": 0.0,  
8         "inicio_ultima_consulta": 0.0  
9     ]]
```

Listing 3: Dicionário de atributos do médico.

- **Fila de Espera:** Implementada como uma lista dinâmica onde a ordem é determinada pelo peso que combina a *Prioridade Clínica* com o *Tempo de Chegada*:

```
1 # Definicao de pesos para o algoritmo de ordena o  
2 PRIORIDADE_MAP = {  
3     "urgente": 1,  
4     "alta": 2,  
5     "normal": 3,  
6     "baixa": 4  
7 }  
8  
9 # A ordenacao garante que a emergencia (0) passe frente da baixa  
10 # (3)  
11 fila_ordenada = sorted(fila, key=lambda x: (PRIORIDADE_MAP.get(x["  
12     prioridade"].lower(), 3), x["tempo_chegada"])))
```

Listing 4: Lógica de pesos e ordenação da fila de prioridade

3 Modelo de Simulação e Algoritmos

O núcleo deste projeto é o motor de simulação de eventos discretos, que modela o comportamento da clínica ao longo do tempo através de processos e gestão de prioridades.

3.1 Modelação das Chegadas (Processo de Poisson)

As chegadas de novos pacientes à clínica não ocorrem em intervalos fixos, mas seguem um processo de **Poisson**. A probabilidade de um determinado número de chegadas num intervalo de tempo é modelada pela taxa λ (lambda). No algoritmo, o intervalo entre chegadas sucessivas é calculado através de uma distribuição exponencial, garantindo que a simulação reflita a imprevisibilidade real de uma sala de espera.

3.2 Tempos de Atendimento

O sistema suporta diferentes perfis de atendimento, permitindo configurar a distribuição do tempo que cada médico dedica a um paciente. Na função `gera_tempo_consulta`, foram implementadas três distribuições principais:

- **Exponencial:** Modela consultas onde a maioria é rápida, mas algumas podem prolongar-se significativamente.



```
1 if dist == "exponential":  
2     # scale e o inverso da taxa de serviço (1/mu)  
3     tempo = np.random.exponential(scale=tempo_medio)
```

Listing 5: Implementação da distribuição exponencial para tempos de consulta

- **Normal (Gaussiana):** Modela um tempo de consulta em torno de uma média fixa com um desvio padrão definido (neste caso, 5 minutos), ideal para procedimentos padronizados.

```
1 elif dist == "normal":  
2     tempo = np.random.normal(loc=tempo_medio, scale=5)
```

Listing 6: Implementação da distribuição normal (Gaussiana)

- **Uniforme:** Define um intervalo rígido onde qualquer tempo de consulta entre o mínimo (50% da média) e o máximo (150% da média) tem a mesma probabilidade de ocorrer.

```
1 elif dist == "uniform":  
2     tempo = np.random.uniform(tempo_medio * 0.5, tempo_medio * 1.5)
```

Listing 7: Implementação da distribuição uniforme

Para garantir a estabilidade da simulação e evitar valores irreais (como tempos negativos ou consultas de 0 minutos), o algoritmo aplica um limite inferior e superior (*clamping*):

```
1 # Garante que o tempo fica entre 30% e 200% do valor medio esperado  
2 tempo = max(tempo_medio * 0.3, min(tempo, tempo_medio * 2.0))
```

Listing 8: Lógica de restrição (*clamping*) para valores de tempo

O código garante a integridade dos dados aplicando limites (*clamping*) para evitar tempos negativos ou excessivamente longos que comprometeriam a estabilidade da simulação.

3.3 Algoritmo de Gestão da Fila e Prioridades

Ao contrário de uma fila FIFO (*First-In, First-Out*) simples, a clínica utiliza uma **Fila de Prioridade Clínica**. A função `ordenar_fila_por_prioridade` utiliza o seguinte critério de ordenação:

$$f(x) = (\text{Peso da Prioridade, Fila}) \quad (1)$$

Para implementar esta lógica, o sistema utiliza um dicionário de mapeamento que converte as etiquetas de texto em valores numéricos, permitindo a comparação aritmética direta:



```
1 PRIORIDADE_MAP = {  
2     "urgente": 1,  
3     "alta": 2,  
4     "normal": 3,  
5     "baixa": 4  
6 }
```

Listing 9: Mapeamento de prioridades para valores numéricos

Os níveis de prioridade são mapeados numericamente: *Urgente* (1), *Alta* (2), *Normal* (3) e *Baixa* (4). O algoritmo ordena a lista de pacientes primeiro pelo peso (onde o menor valor tem maior precedência) e, em caso de empate, pelo tempo de chegada, assegurando justiça entre pacientes com a mesma gravidade através da função `sorted`:

```
1 # Ordena o utilizando uma tupla (Peso, Tempo) como critério  
2 fila_ordenada = sorted(fila, key=lambda x: (PRIORIDADE_MAP.get(x["  
    prioridade"].lower(), 3), x["tempo_chegada"]))
```

Listing 10: Implementação da ordenação de fila com chave composta

```
1 def ordenar_fila_por_prioridade(fila, prioridades):  
2     return sorted(fila, key=lambda x: (prioridades.get(x["  
    prioridade"].lower(), 3), x["tempo_chegada"]))
```

Listing 11: Implementação da ordenação estocástica

Esta implementação garante que um paciente que chegue em estado "Urgente" seja imediatamente colocado no topo da fila, independentemente de quantos pacientes com prioridade "Normal" já se encontravam em espera, respeitando os princípios éticos de triagem hospitalar.

3.4 Atribuição por Especialidade

Um diferencial do modelo implementado é a lógica de encaminhamento inteligente. Em vez de uma atribuição aleatória, o sistema cruza os dados clínicos do paciente com a qualificação do corpo clínico.

Para tal, utiliza-se um dicionário de mapeamento que traduz sintomas ou patologias na especialidade médica correspondente:

```
1 # Mapeamento definido no motor de simulação  
2 MAPEAMENTO_DOENCAS = {  
3     "Arritmia": "Cardiologia",  
4     "Taquicardia": "Cardiologia",  
5     "Gripe": "Clínica Geral",  
6     "Febre": "Clínica Geral",  
7     "Fratura": "Ortopedia"  
8 }
```

Listing 12: Dicionário de mapeamento de patologias

O algoritmo de atribuição verifica a patologia do paciente e procura, entre os médicos com estado `ocupado = False`, aquele cuja especialidade coincide com a necessária. Abaixo ilustra-se a lógica de seleção implementada:



```
1
2 if tempo_atual >= estado_simulacao["proximo_paciente_tempo"] and
   estado_simulacao["pacientes_disponiveis"]:
3
4     paciente = estado_simulacao["pacientes_disponiveis"].pop(0)
5
6     paciente["tempo_chegada"] = tempo_atual
7
8     medico_livre_especialidade = None
9     encontrou_medico_especialista = False
10    especialidade_necessaria = paciente.get("
    especialidade_necessaria", "Clínica Geral")
11
12    i = 0
13    while i < len(estado_simulacao["medicos"]) and not
    encontrou_medico_especialista:
14        medico = estado_simulacao["medicos"][i]
15        if (not medico["ocupado"] and not medico["em_pausa"]
    and
16            medico["especialidade"] == especialidade_necessaria
17        ):
18            medico_livre_especialidade = medico
19            encontrou_medico_especialista = True
20            i = i + 1
21
22    if encontrou_medico_especialista:
        iniciar_consulta(medico_livre_especialidade, paciente,
        tempo_atual)
```

Listing 13: Lógica de seleção de médico por especialidade

Caso nenhum médico compatível (especialista ou clínico geral) esteja disponível no momento, o paciente permanece na fila, mantendo a sua posição de prioridade até que ocorra um evento de libertação de médico. Esta lógica evita que um paciente com uma fratura seja atendido por um cardiologista enquanto um clínico geral está livre, otimizando os recursos da clínica.

4 Interface Gráfica e Funcionalidades

O sistema foi desenvolvido com uma interface gráfica baseada na biblioteca **FreeSimpleGUI**, estruturada para garantir que o utilizador siga um fluxo lógico de operações, prevenindo erros de execução através de validações em tempo real.

4.1 Autenticação de Administrador

Para garantir a integridade dos dados, o acesso às funcionalidades de gestão e configuração é protegido por um sistema de autenticação. O utilizador deve introduzir as suas credenciais, que são validadas contra o ficheiro `usersadmin.json`.

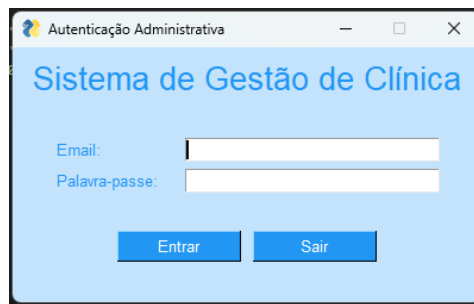


Figure 1: Janela de autenticação para acesso administrativo.

4.2 Menu Principal

O Menu Principal é o núcleo de navegação da aplicação. Apresenta três opções fundamentais: **Importar Dados**, **Simulação Clínica** e **Alterar Dados**. Esta interface atua como um "estado de espera" onde o sistema verifica as pré-condições necessárias para as fases seguintes.



Figure 2: Interface do Menu Principal do simulador.

4.3 Importação de Dados

O primeiro passo obrigatório é a importação de ficheiros. O utilizador deve carregar as bases de dados de **Médicos** e **Pacientes** a partir dos ficheiros JSON. Sem esta ação, as listas internas do programa permanecem vazias, impedindo qualquer processamento.

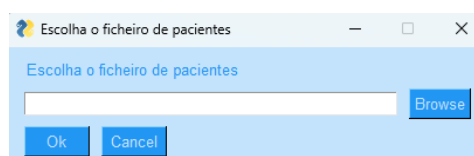


Figure 3: Janela de importação de ficheiro de pacientes.

4.4 Tratamento de Erros e Validação

Caso o utilizador tente aceder à **Simulação Clínica** ou ao menu **Alterar Dados** sem antes ter realizado a importação, o sistema interrompe a transição e exibe um *pop-up* de erro crítico.

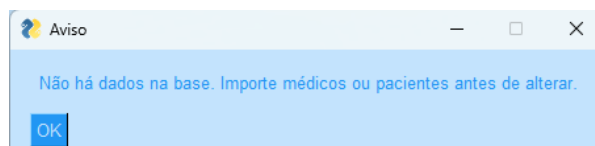


Figure 4: Aviso de erro disparado ao tentar avançar sem dados importados.

4.5 Gestão e Alteração de Dados

Ao aceder ao módulo de **Alterar Dados**, é aberta uma interface dedicada à manipulação direta das bases de dados em memória, com persistência nos ficheiros originais.



Figure 5: Interface de gestão de médicos e pacientes.

Esta interface permite as seguintes operações:

- **Adicionar/Listar Médicos:** Permite inserir novos médicos e visualizar os dados especialidade de cada um.

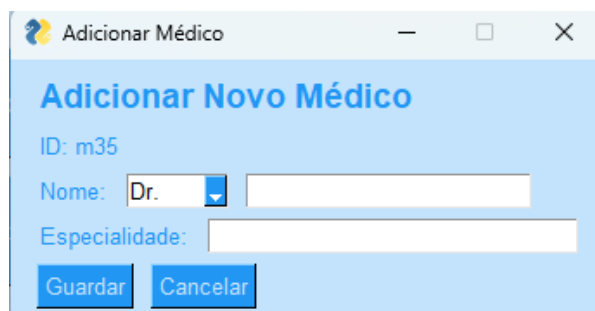


Figure 6: Janela para adicionar novo médico.

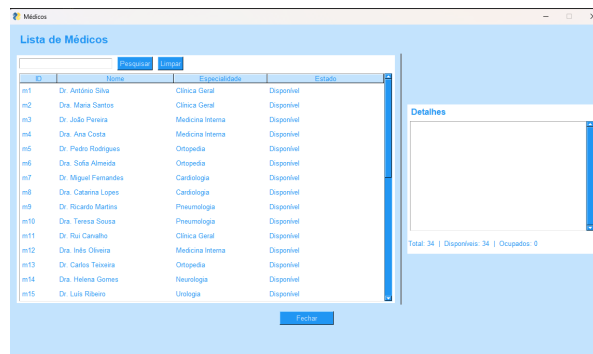


Figure 7: Janela para visualizar lista de médicos.

- **Adicionar/Listar Pacientes:** Permite o registo de novos doentes e a visualização dos mesmos.

Adicionar Novo Paciente

ID: d2001

Nome:

Idade:

Sexo:

Doença:

Prioridade:

Atributos:

☐ Fumador

☐ Consome Álcool

Atividade Física:

☐ Doença Crónica

Figure 8: Janela para adicionar novo paciente.

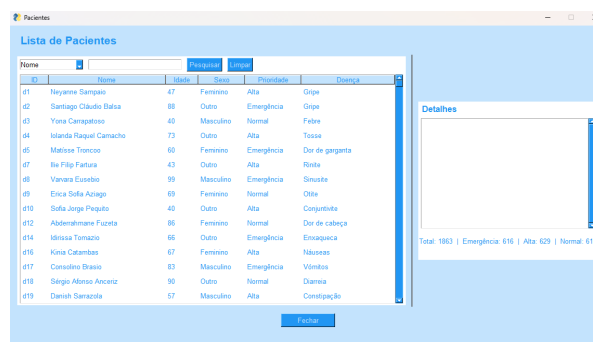


Figure 9: Janela para visualizar pacientes.



4.6 Ambiente de Simulação

Esta é a componente mais avançada da interface, onde o motor é visualizado em tempo real.

4.6.1 Janela de Configuração

Antes de iniciar a execução, o sistema solicita a configuração dos parâmetros da simulação: o tempo total de operação, a taxa de chegada de pacientes (λ) e a distribuição estatística para o tempo de consulta.

The screenshot shows a window titled 'Configuração' with a subtitle 'CONFIGURAÇÃO DA SIMULAÇÃO'. It contains three main sections: 'Parâmetros de Chegada (Distribuição de Poisson)', 'Recursos e Tempo', and 'Pausas dos Médicos'. Each section has input fields for various parameters. At the bottom, there are 'Confirmar' and 'Cancelar' buttons. A red 'IMPORTANTE' warning is also present.

Section	Parameter	Value
Parâmetros de Chegada (Distribuição de Poisson)	Taxa de chegada (λ) pacientes/hora:	10
	Dataset disponível:	1864 pacientes
	Lambda máximo recomendado:	233.0 (para usar todo o dataset)
	Simulação padrão:	480 min = 8.0 horas
Recursos e Tempo	Tempo médio consulta (min):	15
	Tempo de simulação (minutos):	480
	Número de médicos:	34
Pausas dos Médicos	Frequência das pausas (min):	60
	Duração das pausas (min):	15
	Número de pausas por médico:	2
	Máx. médicos em pausa simultânea:	2

IMPORTANTE:
• Máx. 1/3 dos médicos em pausa simultânea

Figure 10: Configuração dos parâmetros estocásticos da simulação.

4.6.2 Monitorização em Tempo Real

Durante a simulação em andamento, a janela apresenta uma visão geral da clínica:

- **Janelas de Visualização:** Duas áreas distintas que mostram o estado dos Médicos (Livre/Ocupado) e a Fila de Espera (pacientes ordenados por prioridade).
- **Estatísticas Vivas:** Indicadores em tempo real como o tempo médio de espera acumulado e o número total de atendimentos.
- **Painel de Ações:** Botões de **Iniciar**, **Parar** e **Pausa**, além de botões para gerar **Análise de Gráficos** (que processam os dados recolhidos até ao momento).

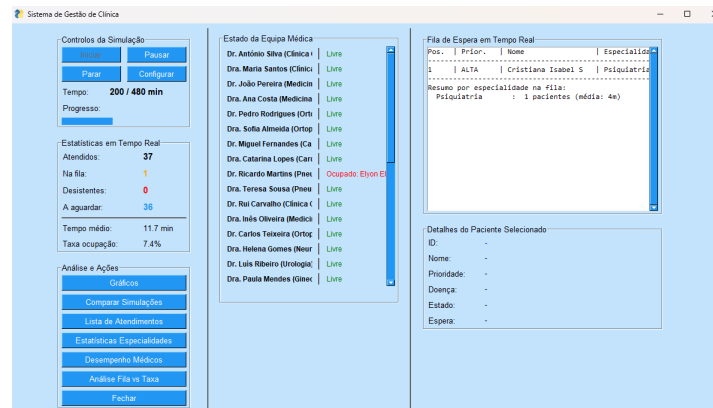


Figure 11: Interface de simulação ativa com monitorização de médicos e fila de espera.

4.7 Análise de Resultados e Ações

Após a execução da simulação, o sistema disponibiliza um conjunto de ferramentas de análise quantitativa. Estas funcionalidades permitem ao administrador avaliar a eficiência da clínica e identificar padrões operacionais.

4.7.1 Visualização Gráfica e Estatística

A opção **Gráficos** utiliza a biblioteca `matplotlib` para gerar uma visão panorâmica do desempenho clínico. Para evitar a extensão excessiva do documento, os resultados foram agrupados por categorias de análise.

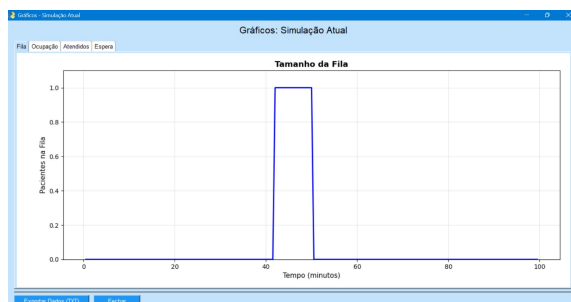


Figure 12: Gráfico do tamanho da fila.

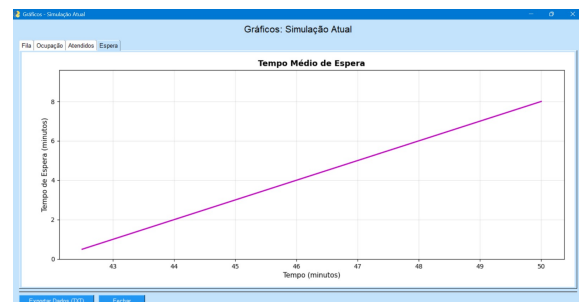


Figure 13: Tempo médio de espera.

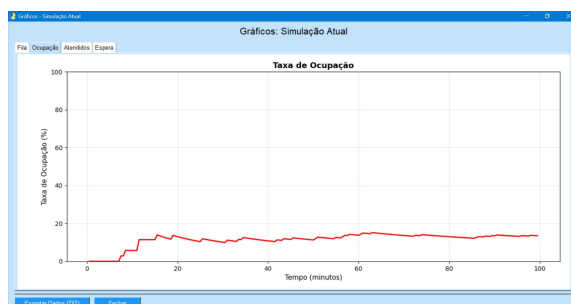


Figure 14: Taxa de Ocupação.

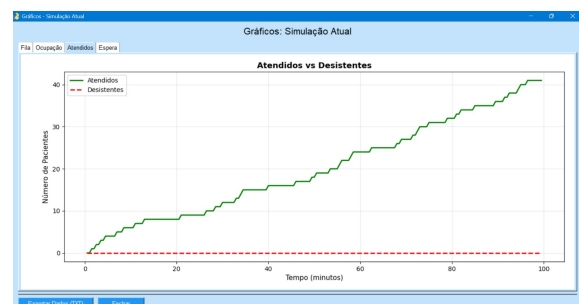


Figure 15: Atendidos VS Desistências.



4.7.2 Comparação de Simulações

Esta funcionalidade permite carregar dados de sessões anteriores e sobrepô-los aos dados atuais. É essencial para a **Análise de Cenários**, permitindo observar, por exemplo, como a alteração da distribuição de "Normal" para "Exponencial" afetou o tempo médio de espera sob a mesma carga de pacientes.

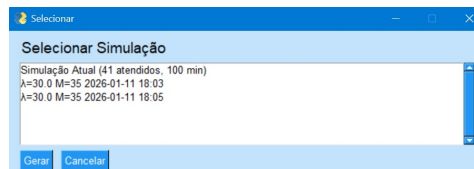


Figure 16: Definir simulações a comparar.

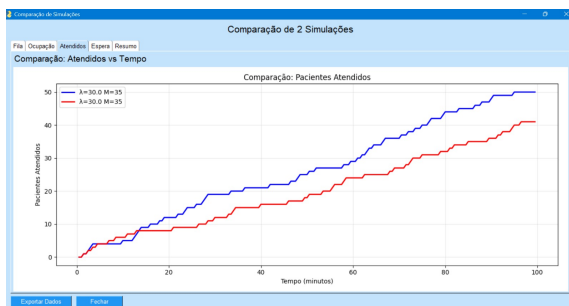


Figure 17: Comparação de simulações, pacientes atendidos.

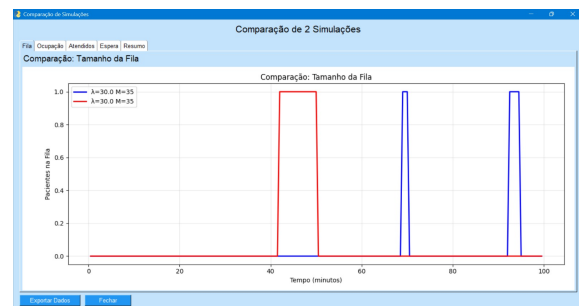


Figure 18: Comparação de simulações, tamanho da fila.

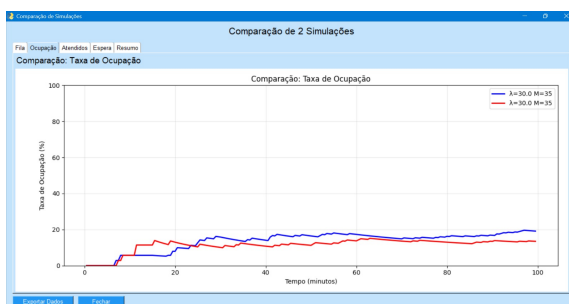


Figure 19: Comparação de simulações, taxa de ocupação.

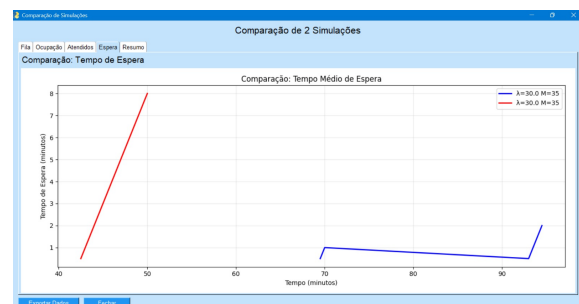


Figure 20: Comparação de simulações, tempo médio de espera.

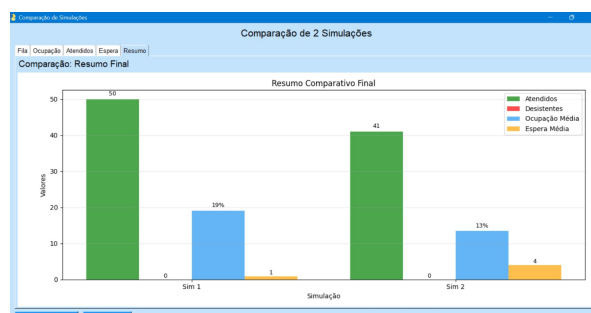


Figure 21: Comparação de simulações, resumo final.

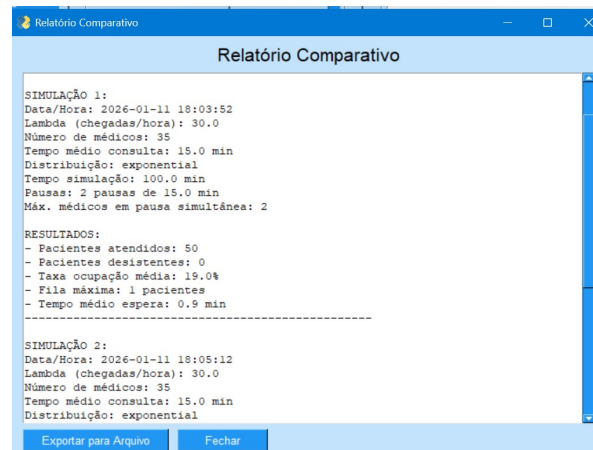


Figure 22: Relatório Comparativo de simulações.

4.7.3 Lista Detalhada de Atendimentos

A **Lista de Atendimentos** funciona como um histórico ou *log* de auditoria. Para cada paciente que passou pela clínica, o sistema registra: o nome, o médico assistente, a especialidade, a hora de entrada, a hora de saída e o tempo total de permanência no sistema.



Figure 23: Lista detalhada de atendimentos.



Ficha: Adaylton Entresede

FICHA DO PACIENTE

DADOS PESSOAIS

ID: d778
Nome: Adaylton Entresede
Idade: 99
Sexo: outro
Prioridade: ALTA

DADOS DA CONSULTA

Status: ATENDIDO
Especialidade: Neurologia
Espera: 0.0 min
Duração: 26.7 min
Médico: Dra. Helena Gomes

HÁBITOS DE SAÚDE

Fumador: Não
Álcool: Sim
Crónico: Não

Fechar

Figure 24: Detalhes do paciente dentro da lista de pacientes.

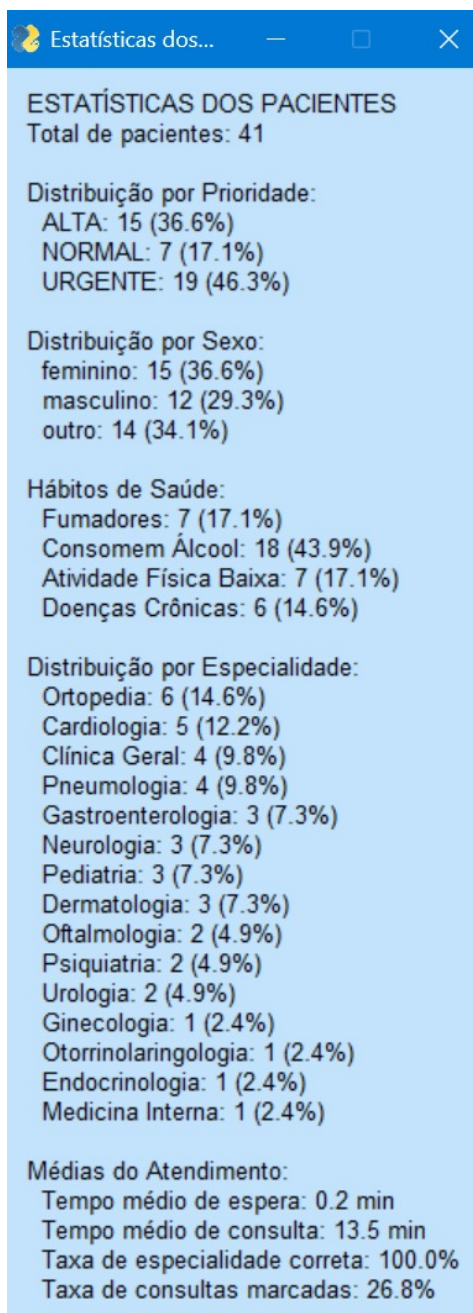


Figure 25: Estatística relativas aos atendimentos da simulação.

4.7.4 Estatísticas por Especialidade

Esta análise quantifica a procura por cada área médica. Através de um gráfico de barras, o gestor pode identificar quais as especialidades com maior volume de trabalho, auxiliando na decisão de contratar novos médicos para áreas saturadas (ex: Clínica Geral vs. Cardiologia).



Estatísticas de Especialidades - Análise Completa

ESTATÍSTICAS DE ESPECIALIDADES - PACIENTES ATENDIDOS

Data/Hora: 11/01/2026 18:20:08
Tempo simulação: 100 minutos
Total de pacientes atendidos: 41
Total de atendimentos: 41
Total médicos disponíveis: 35
Pacientes encontrados nos dados: 41

DISTRIBUIÇÃO DE PACIENTES ATENDIDOS POR ESPECIALIDADE MEDICINA

Especialidade	Pacientes	% Total	Urgentes	Médicos	Ocupação	Corresp.
Ortopedia	6	14.6%	4	3	24.0%	100.0%
Cardiologia	5	12.2%	3	2	36.0%	100.0%
Clínica Geral	4	9.8%	2	3	15.0%	100.0%
Pneumologia	4	9.8%	0	2	32.0%	100.0%
Gastroenterologia	3	7.3%	0	2	32.7%	100.0%
Neurologia	3	7.3%	2	2	6.0%	100.0%
Dermatologia	3	7.3%	0	2	6.0%	100.0%
Neurocirurgia	3	7.3%	3	3	5.5%	100.0%
Psiquiatria	2	4.9%	1	2	17.2%	100.0%
Oftalmologia	2	4.9%	1	2	10.7%	100.0%
Urologia	2	4.9%	1	3	6.9%	100.0%
Medicina Interna	1	2.4%	0	3	10.0%	100.0%
Otorrinolaringologia	1	2.4%	0	2	9.5%	100.0%
Endocrinologia	1	2.4%	0	2	14.3%	100.0%
Endocrinologia	1	2.4%	1	2	14.2%	100.0%

ANÁLISE DE BALANÇAMENTO DE RECURSOS

TOP 4 ESPECIALIDADES COM MAIOR DEMANDA:

- Ortopedia: 6 pacientes, 3 médico(s) (2.0 pacientes/médico)
- Cardiologia: 5 pacientes, 2 médico(s) (2.5 pacientes/médico)
- Clínica Geral: 4 pacientes, 3 médico(s) (1.3 pacientes/médico)
- Pneumologia: 4 pacientes, 2 médico(s) (2.0 pacientes/médico)
- Gastroenterologia: 3 pacientes, 2 médico(s) (1.5 pacientes/médico)

RESPOSTA MÁXIMA POR HORA: 0.70 (1.7h)

Figure 26: Distribuição do volume de consultas por especialidade médica.

4.7.5 Análise de Desempenho dos Médicos

Esta métrica foca-se na eficiência individual e coletiva do corpo clínico. O sistema calcula a **Taxa de Ocupação**, que é o rácio entre o tempo que o médico passou em consulta e o tempo total da simulação.

Desempenho Médico - Simulação Atual

Análise de Desempenho Médico: Simulação Atual

Efficiência | Tempo | Correspondência | Ocupação | Relatório

Relatório Completo

RELATÓRIO DE DESEMPENHO MÉDICO - Simulação Atual

Data/Hora: 11/01/2026 18:20:48
Tempo total de simulação: 100 minutos (1.7 horas)
Total de médicos analisados: 35

ESTATÍSTICAS GERAIS:

Total de atendimentos realizados: 41
Eficiência média: 0.70 pacientes/hora
Tempo médio de consulta: 9.2 minutos
Taxa média de correspondência: 60.0%
Taxa média de ocupação: 15.8%

ESTATÍSTICAS POR ESPECIALIDADE:

CLÍNICA GERAL (3 médicos):

- Atendimentos: 4
- Eficiência média: 0.80 pacientes/hora
- Tempo médio: 7.9 minutos
- Correspondência: 66.7%
- Ocupação: 15.8%

MEDICINA INTERNA (3 médicos):

- Atendimentos: 1
- Eficiência média: 0.20 pacientes/hora
- Tempo médio: 10.0 minutos
- Correspondência: 33.3%
- Ocupação: 10.0%

ORTOPEDIA (3 médicos):

Exportar Relatório | Fechar

Figure 27: Indicadores de produtividade e taxa de ocupação dos médicos.

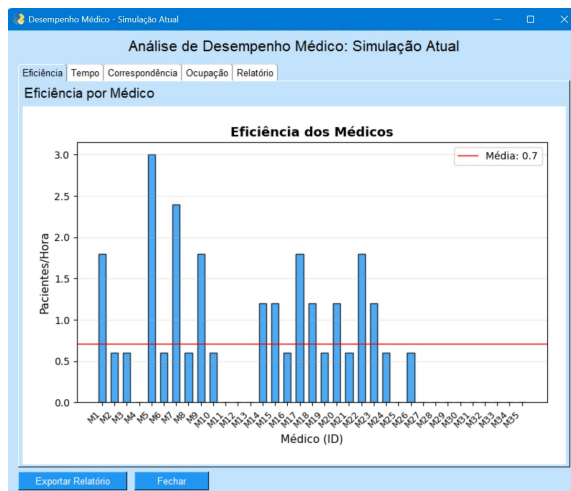


Figure 28: Desempenho dos médicos, eficiência dos mesmos.

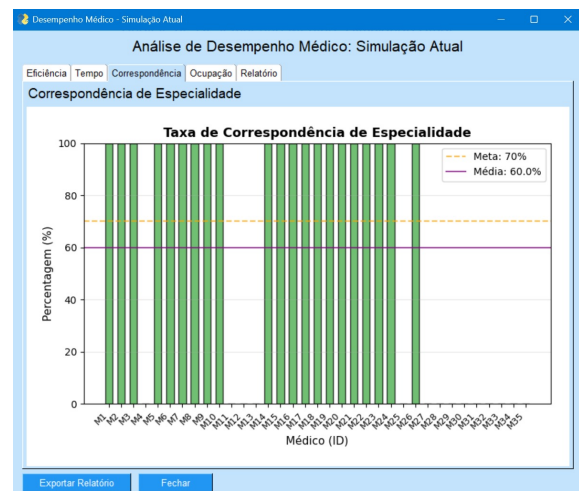


Figure 29: Desempenho dos médicos, taxa de correspondência de especialidade.

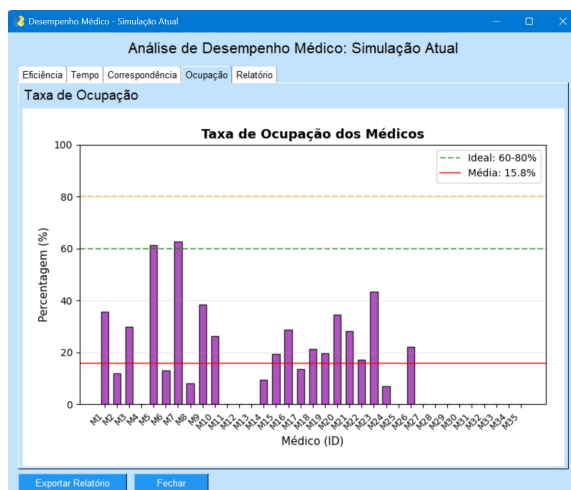


Figure 30: Desempenhos dos médicos, taxa de ocupação.

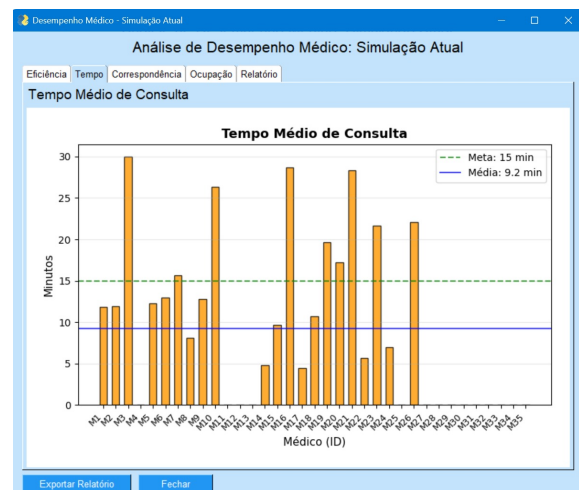


Figure 31: Desempenho dos médicos, tempo médio de consulta.

4.7.6 Análise de Correlação: Fila vs. Taxa de Chegada

Esta é a ferramenta mais técnica do módulo de análise. Ela mostra a relação entre a taxa de chegada (λ) e o tamanho médio da fila. Este gráfico permite validar a teoria das filas de espera, demonstrando o ponto de saturação onde a clínica deixa de conseguir dar vazão aos novos pacientes.



Análise Fila vs Taxa

Este gráfico mostra como o tamanho médio da fila varia com a taxa de chegada de pacientes.

Parâmetros da análise:

Configuração da Análise

Taxas a testar: 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30 (valores separados por vírgula)

Número de médicos: 35

Tempo por simulação: 100 minutos

Usar simulação anterior: λ=30.0 M=35 2026-01-11

Iniciar Análise Cancelar

Figure 32: Janela de configuração de análise Fila vs Taxa de Chegada.

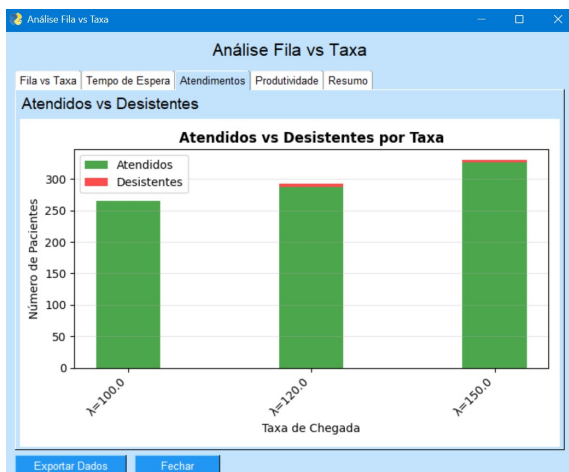


Figure 33: Atendidos Vs Desistências.

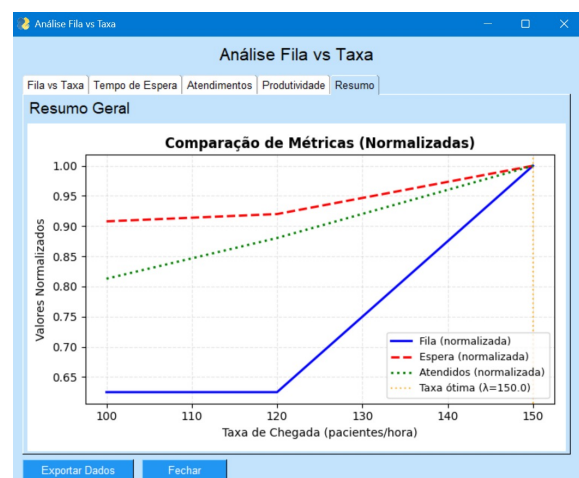


Figure 34: Comparação de métricas normalizadas.

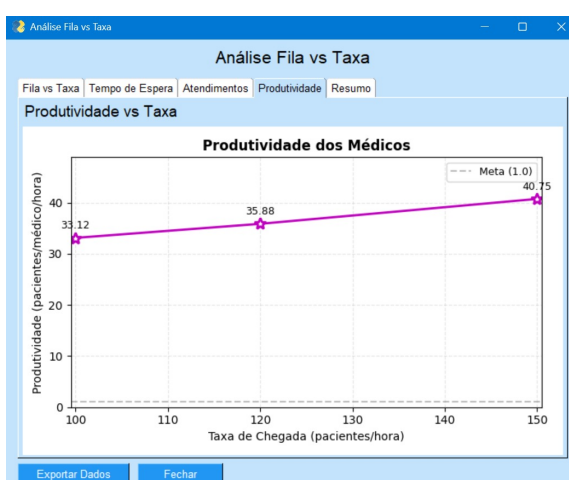


Figure 35: Produtividade dos médicos.

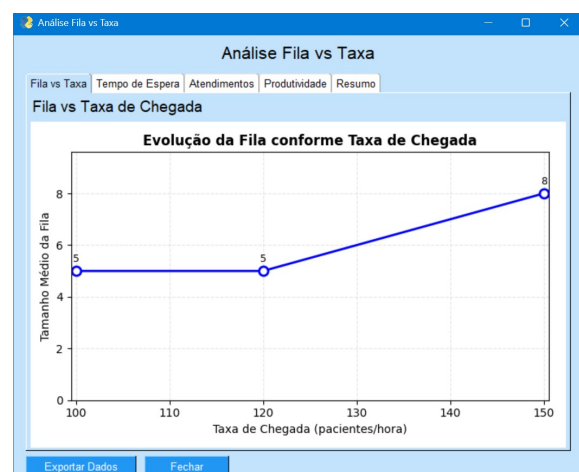


Figure 36: Evolução da fila conforme taxa de chegada.

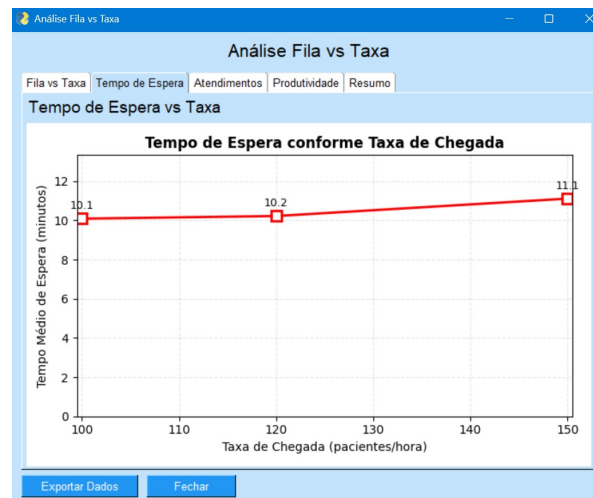


Figure 37: Tempo de espera conforme taxa de chegada.

5 Funções Utilizadas e Lógica de Processamento

Este capítulo descreve as funções modulares implementadas no ficheiro `funcoes.py`, que constituem o núcleo lógico do sistema. Estas funções isolam a complexidade algorítmica da interface gráfica, facilitando a manutenção do código.

5.1 Gestão de Ficheiros e Persistência

O sistema utiliza funções específicas para garantir que os dados são lidos e guardados corretamente em formato JSON. A função `carregar_dados(arquivo)` exemplifica o tratamento de erros e a construção de caminhos dinâmicos.

```
1 def carregar_dados(arquivo):
2
3     if 'medicos' in arquivo:
4         caminho = CAMINHO_MEDICOS
5     else:
6         caminho = CAMINHO_PACIENTES
7
8     if os.path.exists(caminho):
9         with open(caminho, 'r', encoding='utf-8') as f:
10             return json.load(f)
11     else:
12         return {"medicos": []} if "medicos" in arquivo else {"pacientes": []}
```

Listing 14: Carregamento de dados

5.2 Estrutura de Fila (FIFO)

Embora a fila seja reordenada por prioridade, a base da manipulação de dados segue a estrutura *First-In, First-Out*. Foram implementadas funções de conveniência para gerir a lista de espera:



- **enqueue:** Adiciona um elemento ao final da lista.
- **dequeue:** Remove e retorna o primeiro elemento (índice 0).
- **queue_peek:** Observa o próximo paciente sem o remover da fila.

```
1 def enqueue(q, item):
2     """Adiciona um item ao final da fila (FIFO)."""
3     return q + [item]
4
5 def dequeue(q):
6     """Remove e retorna o primeiro item da fila (FIFO)."""
7     if not q:
8         return None, []
9     return q[0], q[1:]
10
11 def queue_empty(q):
12     """Verifica se a fila est vazia."""
13     return len(q) == 0
14
15 def queue_peek(q):
16     """Retorna o primeiro item da fila sem remov-lo."""
17     if queue_empty(q):
18         return None
19     return q[0]
```

Listing 15: Funções utilitárias para manipulação da fila

5.3 Lógica de Limpeza de Estado

Para garantir que cada execução da simulação comece de forma "limpa", foi criada a função `limpar_flag_importacao`. Esta função lida com o sistema de ficheiros para remover indicadores de sessões anteriores, garantindo que o utilizador seja forçado a importar dados novos a cada arranque do `main.py`.

```
1 def limpar_flag_importacao():
2     if os.path.exists(CAMINHO_FLAG_IMPORTACAO):
3         os.remove(CAMINHO_FLAG_IMPORTACAO)
```

Listing 16: Gestão de estado global através de ficheiros de flag

5.4 Formatação e Sanitização de Dados

Dada a natureza clínica dos dados, é essencial garantir que nomes e especialidades sejam comparados sem erros de acentuação ou caixa (maiúsculas/minúsculas). A função `remover_acentos` utiliza a biblioteca `unicodedata` para normalizar strings:

```
1 def remover_acentos(texto):
2     if not texto: return ""
3     return "".join(c for c in unicodedata.normalize('NFD', texto))
```



```
4 if unicodedata.category(c) != 'Mn')
```

Listing 17: Normalização de strings para comparação robusta

5.5 Geração de Relatórios e Exportação

Após a simulação, o sistema não só apresenta os dados como permite a sua exportação. Neste excerto de código podemos ver que exporta um relatório com os histórico dos atendimentos.

```
1 elif event == "Exportar para TXT":
2     nome_arquivo = f"relatorio_atendimentos_{titulo.lower()
3     .replace(' ', '_')}_{datetime.now().strftime('%Y%m%d_%H%M%S')}.
4     txt"
5     arquivo_salvo = salvar_arquivo(texto_relatorio_completo
6     , nome_arquivo)
7     if arquivo_salvo:
8         sg.popup(f"Relatório exportado para:\n{
9         arquivo_salvo}", title="Sucesso")
10    else:
11        sg.popup_error("Falha ao salvar o relatório.",
12        title="Erro")
```

Listing 18: Exportação de resultados com timestamp

5.5.1 Manipulação de dados

Além da execução do motor de simulação, o sistema oferece uma camada de gestão administrativa que permite a manipulação direta e em tempo real dos dados de médicos e pacientes. Através de funções de filtragem e edição, o utilizador pode atualizar especialidades médicas, ajustar níveis de prioridade clínica ou registar novas patologias, garantindo que a base de dados (*pacientes.json* e *medicos.json*) se mantém fidedigna à realidade da clínica. Esta funcionalidade de *CRUD* (*Create, Read, Update, Delete*) simplificada é essencial para testar cenários específicos, permitindo que o administrador altere variáveis de entrada sem necessidade de editar manualmente os ficheiros de texto, assegurando assim a integridade estrutural do sistema.

6 Conclusão

Os objetivos propostos no enunciado foram integralmente cumpridos. O sistema é capaz de:

- Simular a chegada de pacientes via processo de Poisson e gerir o atendimento com diferentes distribuições de tempo.
- Implementar uma triagem dinâmica baseada em prioridades clínicas e especialidades médicas.
- Fornecer uma análise estatística visual que facilita a tomada de decisão sobre a alocação de recursos.



A arquitetura modular adotada demonstrou ser eficaz, permitindo que a lógica matemática fosse testada independentemente da interface gráfica, o que garantiu a estabilidade do programa durante simulações de longa duração.

Com este projeto, reforçou-se a ideia de que a simulação computacional é um pilar indispensável na gestão hospitalar moderna, permitindo otimizar o atendimento e, consequentemente, melhorar os cuidados prestados ao paciente.

7 Anexos

Nesta secção são apresentadas as bibliotecas utilizadas.

- **NumPy**: Para geração de números aleatórios com distribuições de Poisson e Exponencial.
- **Matplotlib**: Para a geração dos gráficos de desempenho da clínica.
- **FreeSimpleGUI**: Para a criação da interface gráfica (GUI).
- **JSON**: Para a serialização e armazenamento local de dados.