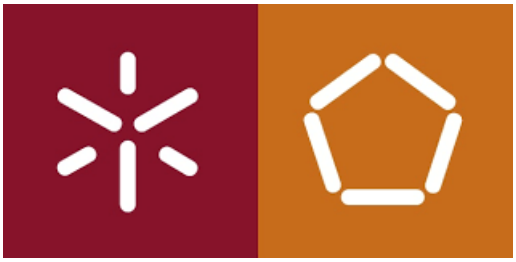

Relatório do Projeto

Simulação e Gestão Hospitalar

Algoritmos e técnicas de programação
Ano Letivo 2025/2026

Autores:

Diogo Pecegueiro (A111065)

Filipe Carvalho (A111961)

Gonçalo Rosa (A112331)

Docentes:

José Carlos Leite Ramalho (JCR)

Luís Filipe Costa Cunha

Braga, 11 de janeiro de 2026

Resumo

O presente relatório descreve o processo de desenvolvimento e teste da aplicação *Sistema de Gestão Hospitalar*, realizada no âmbito da unidade curricular de Algoritmos e Técnicas de Programação. O projeto visa a otimização de fluxos de atendimento em ambiente hospitalar, utilizando simulação computacional.

O sistema simula um turno de uma clínica hospitalar (tipicamente 8 horas), gerindo a chegada aleatória de pacientes, a sua triagem baseada na gravidade clínica (inspirada no Protocolo de Manchester) e o subsequente encaminhamento para médicos com a respetiva especialidade.

Para a interface com o utilizador, optou-se pela biblioteca `FreeSimpleGUI`, que permitiu criar um ambiente visual moderno e intuitivo. O núcleo da simulação utiliza eventos discretos para avançar o tempo e calcular métricas de desempenho. No final da execução, o sistema recorre à biblioteca `matplotlib` para gerar gráficos analíticos que ilustram a evolução das filas de espera, as desistências por tempo excessivo, a taxa de ocupação dos recursos médicos, as médias de espera, o volume de consultas, a eficiência médica, os atendimentos por especialidade e a taxa de chegada dos pacientes.

Os resultados obtidos demonstram que a simulação é uma ferramenta eficaz para prever comportamentos do sistema hospitalar, permitindo identificar sobrecargas no atendimento e ajustar o número de médicos necessários para cada especialidade, evitando a saturação do serviço.

Índice

Lista de Figuras

Lista de Tabelas

1 Introdução

A gestão eficiente de recursos em unidades de saúde é um desafio complexo. A variabilidade na chegada de doentes e a incerteza na duração dos tratamentos tornam difícil o planeamento estático. Neste contexto, a simulação computacional apresenta-se como uma ferramenta poderosa para modelar estes cenários sem os riscos associados a testes em ambiente real.

O presente projeto tem como objetivo o desenvolvimento de uma aplicação em Python, denominada *Sistema de Gestão Hospitalar*, que simula o funcionamento de uma clínica médica. O sistema foca-se na gestão de filas de espera, atribuição de prioridades de triagem e alocação de médicos especialistas.

1.1 Objetivos do Trabalho

Os principais objetivos definidos para este projeto foram:

1. **Segurança e Acesso:** Implementar um sistema de login seguro com leitura e escrita de ficheiros JSON para gestão de utilizadores.
2. **Motor de Simulação:** Desenvolver um algoritmo de simulação baseado em eventos discretos capaz de gerar chegadas aleatórias e gerir filas de prioridade.
3. **Interface Gráfica (GUI):** Criar uma interface amigável e responsiva para monitorização em tempo real do estado da clínica.
4. **Análise de Dados:** Gerar relatórios gráficos para análise estatística pós-simulação.

2 Arquitetura da Solução

A aplicação foi estruturada de forma modular, seguindo uma aproximação ao padrão MVC (*Model-View-Controller*), separando o algoritmo base, a demonstração gráfica dos resultados e a interface com o utilizador. Esta abordagem facilita a manutenção e a escalabilidade do código.

[Image of MVC software architecture pattern]

2.1 Estrutura de Ficheiros

O projeto divide-se nos seguintes módulos Python:

Tabela 1: Descrição dos módulos do projeto.

Ficheiro	**Componente**	**Descrição Funcional**
`Main.py`	Controller	Controlador principal que integra todos os componentes da simulação.
`Base.py`	Model	Motor de simulação, gestão de eventos, filas de prioridade e lógica de médicos.
`Interface.py`	View	Definição visual das janelas e elementos gráficos.
`Login.py`	Security	Gestão de autenticação, leitura/escrita de utilizadores e validação de acesso.
`Gráficos.py`	Analytics	Processamento de dados históricos e geração de gráficos com Matplotlib.

2.2 Armazenamento de Dados

A persistência de dados é garantida através de ficheiros JSON:

- `utilizadores.json`: Armazena as credenciais de acesso (ID, Password, Nome).
- `pessoas.json`: Base de dados com perfis fictícios para geração aleatória de pacientes realistas.

3 Implementação Técnica

3.1 Gestão de Acessos (Login.py)

O módulo `Login.py` assegura a gestão de acessos e a persistência de dados dos utilizadores, recorrendo a ficheiros JSON e à biblioteca `FreeSimpleGUI`. As suas funcionalidades principais dividem-se em:

- **Persistência de Dados:** As funções `ler_utilizadores()` e `gravar_utilizadores(bd)` garantem, respetivamente, a leitura da base de dados `utilizadores.json` para memória e a escrita de novos registos, assegurando a integridade da informação.
- **Validação da Autenticação:** A função `validar_login()` percorre os registos para verificar a correspondência entre as credenciais inseridas e as armazenadas.
- **Controlo de Interface:** A função `executar_login_controller()` gere o ciclo de vida da janela de autenticação, processando eventos de login e registo dos novos administradores.

Figura 1: Janela de Login e Autenticação

3.2 Motor de Simulação (Base.py)

O módulo `Base.py` constitui o núcleo algorítmico do *Sistema de Gestão Hospitalar*. A sua implementação baseia-se no paradigma de **Simulação de Eventos Discretos**, onde o estado do sistema evolui em pontos específicos do tempo através de uma lista de eventos agendados.

Abaixo descrevem-se as funções e algoritmos fundamentais que governam a simulação:

`enqueue(q, elemento)`

Esta função gere a lista de eventos (`q_eventos`), mantendo-a rigorosamente ordenada pelo tempo de ocorrência (`e_tempo`). Utiliza uma lógica de inserção ordenada, garantindo que o motor de simulação processe sempre o evento cronologicamente mais próximo.

`gera_tempo_consulta(triagem)`

Implementa a variabilidade estocástica do atendimento. Através de distribuições uniformes (usando `random.uniform`), calcula a duração de uma consulta com base na prioridade atribuída: pacientes em estado de **Emergência** requerem tempos de intervenção superiores e mais complexos do que casos **Normais**.

`simular(tempo_max, taxa_chegada, ...)`

É a função principal que orquestra todo o ciclo de vida da clínica virtual. O seu funcionamento divide-se em três pilares:

- **Gestão de Chegadas:** Gera novos pacientes usando uma distribuição exponencial e atribui uma especialidade baseada em probabilidades predefinidas.
- **Lógica de Triagem e Filas:** Gere a `q_espera`, uma fila de prioridade onde os pacientes são ordenados pelo seu nível clínico. Implementa também a lógica de **desistência**, onde pacientes de baixa prioridade abandonam o sistema se o tempo de espera for muito alto.
- **Alocação de Recursos:** Gere a equipa de médicos, distinguindo entre especialistas e médicos de reserva. O algoritmo tenta primeiro alocar o

paciente ao especialista da área correta e, caso este esteja ocupado, recorre a um **médico de reserva**.

Funções de Acesso (`e_tempo, e_tipo, e_doente, etc.`)

Um conjunto de funções auxiliares (*getters*) que abstraem a estrutura da tupla de eventos. Isto permite que o código seja mais legível e fácil de manter.

3.3 Interface Gráfica e Interação

O módulo `Interface.py` é responsável pela camada de visualização (*View*). A sua implementação baseia-se na biblioteca `FreeSimpleGUI`.

Abaixo descrevem-se os aspetos técnicos e as funções principais deste módulo:

- **Design e Estética:** Foi adotado o tema `'DarkTeal9'`, que oferece uma interface de alto contraste e aspeto moderno. Foram definidas constantes globais para tipos de letra (`Helvetica` e `Arial`) e tamanhos de botões, garantindo a consistência visual em toda a aplicação.
- **Construção de Janelas:**
  - `criar_janela_login()`: Define o layout da primeira interação do utilizador, focando-se na simplicidade e na clareza dos campos de entrada de dados.
  - `criar_janela_principal(nome)`: Implementa o *Dashboard* central. Na interface, é possivel visualizar diferentes botões cujas funções divergem para os diferentes módulos.
- **Janelas de Suporte e Popups:**
  - `janela_log_popup()`: Gera uma janela modal que apresenta o historial detalhado de todos os eventos da simulação numa tabela formatada.
  - `janela_medicos_popup()`: Foca-se na apresentação da percentagem da performance de cada médico, permitindo uma análise quanto à disposição dos médicos.

- **Responsividade:** A interface foi desenhada de forma a que os elementos se ajustem automaticamente, utilizando *layouts* em listas que permitem uma leitura fluida tanto de dados textuais como de tabelas complexas.

Figura 2: Dashboard principal durante a execução da simulação

3.4 Demonstração Gráfica de Resultados (Gráficos.py)

O módulo `Gráficos.py` é responsável pelo processamento estatístico e pela renderização visual dos dados recolhidos durante a simulação.

Abaixo descrevem-se as funcionalidades e a lógica de implementação deste módulo:

- **Padronização Estética:** A função `aplicar_tema()` define uma identidade visual moderna e consistente (estilo *dark mode*), configurando cores de fundo, grelhas e fontes para que todos os gráficos mantenham a mesma linha de design da interface principal.

- **Processamento de Dados:** O módulo utiliza funções como `desembrulhar_seguro()` para extrair valores do dicionário de simulação.

- **Interatividade e Detalhe:** A função `adicionar_valores()` percorre as barras dos gráficos para inserir etiquetas de texto com os valores exatos (percentagens ou contagens), garantindo que a visualização tenha a devida legenda.

3.5 Controlador Principal (Main.py)

O módulo `Main.py` atua como o componente **Controlador** do sistema, sendo responsável por facilitar a interação entre o utilizador, o motor de simulação e as ferramentas de análise.

Abaixo descrevem-se as funções e a lógica de controlo implementadas:

- **Processamento e Tratamento de Dados:** Implementa funções auxiliares como `calcular_media()` e `calcular_media_de_listas()`. Estes métodos são importantes para o tratamento dos dados provenientes do módulo `Base.py`.
- **Gestão do Ciclo de Eventos:** Através da função `executar_app()`, o controlador mantém um ciclo ativo que interpreta as interações na interface principal. Cada evento desencadeia uma resposta:
  - **Desencadeamento da Simulação:** Chama o algoritmo do módulo `Base.py` para o retorno dos valores da simulação, para posterior análise e demonstração na *Dashboard*.
  - **Intermediação de Popups e Gráficos:** Gere as chamadas para o módulo `Gráficos.py` e para as janelas modais de `Interface.py`, como o histórico de eventos (`-LOG-`) e a análise de performance médica (`-TIME-`).
- **Integração e Coordenação de Módulos:** O `Main.py` funciona como o ponto de convergência de todo o software. É responsável por importar e coordenar os módulos de **Segurança** (Login), **Interface** (GUI), **Modelo** (Base) e **Analítica** (Gráficos), garantindo a facilidade de leitura, análise e posterior simulação do programa.

4 Análise de Resultados

O módulo `Gráficos.py` permite visualizar o desempenho do sistema após a simulação de um turno de 480 minutos. A análise destes dados é crucial para a tomada de decisão.

4.1 Evolução das Desistências

A funcionalidade `graf_8_evolucao_desistencias` gera um gráfico de linha que mostra o acumulado de pacientes que abandonaram a fila. **Interpretação:** Observa-se tipicamente que as desistências aumentam exponencialmente quando a capacidade de atendimento dos médicos é superada pela taxa de chegada, criando um efeito de "bola de neve"nas filas de espera.

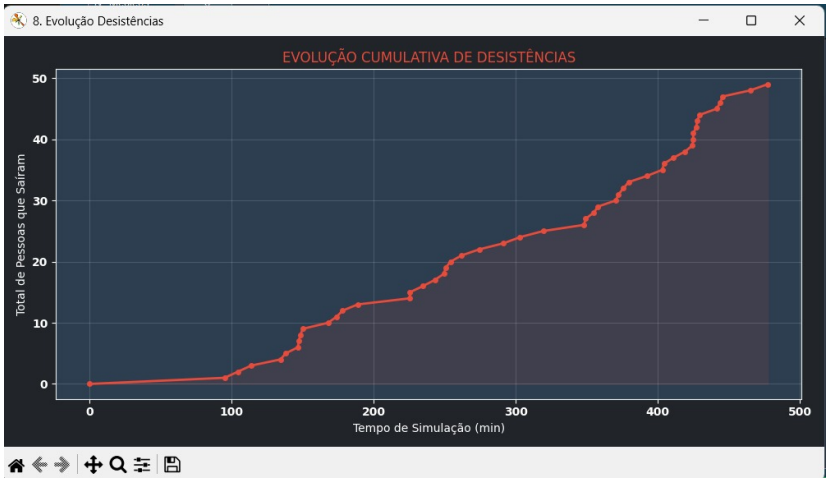

Figura 3: Gráfico da evolução temporal das desistências

4.2 Ocupação Médica

O gráfico de barras de ocupação compara a carga de trabalho entre Médicos Especialistas e Médicos de Reserva. **Interpretação:** Os resultados demonstram frequentemente que os especialistas (ex: Cardiologia, Ortopedia) operam na "Zona Crítica"(acima de 80% de ocupação), sugerindo a necessidade de reforço da equipa nestas áreas específicas.

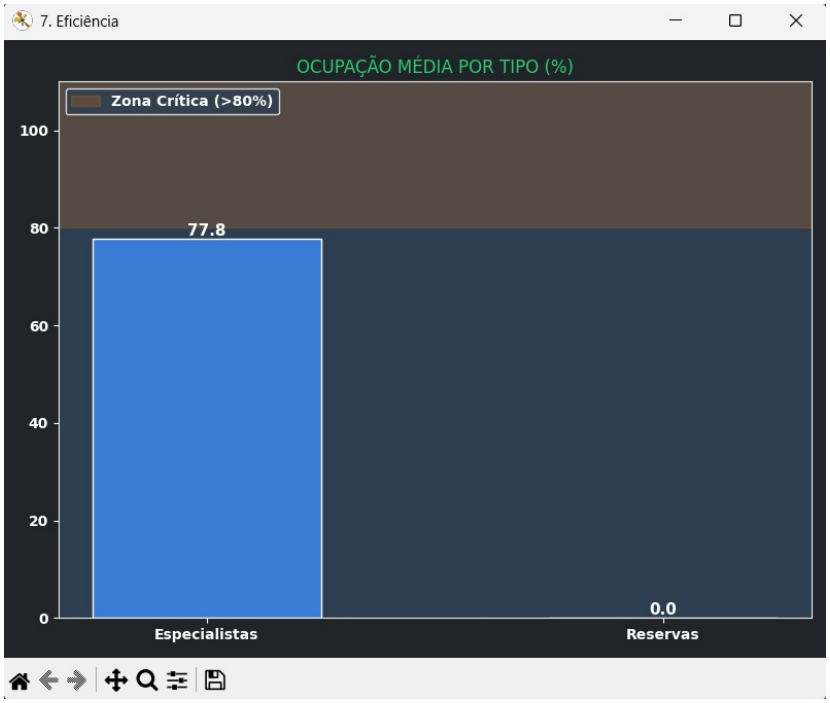

Figura 4: Comparação da taxa de ocupação

4.3 Médias de Espera por nível de Triagem

A análise dos tempos médios de espera por triagem permite identificar a prioridade dada a cada triagem. Através de um gráfico de barras horizontais, o sistema compara o tempo que os doentes aguardam, em média, desde a triagem até ao início da consulta.

Interpretação: Pacientes com níveis de prioridade superior esperam menos tempo do que pacientes com níveis de prioridade inferiores.

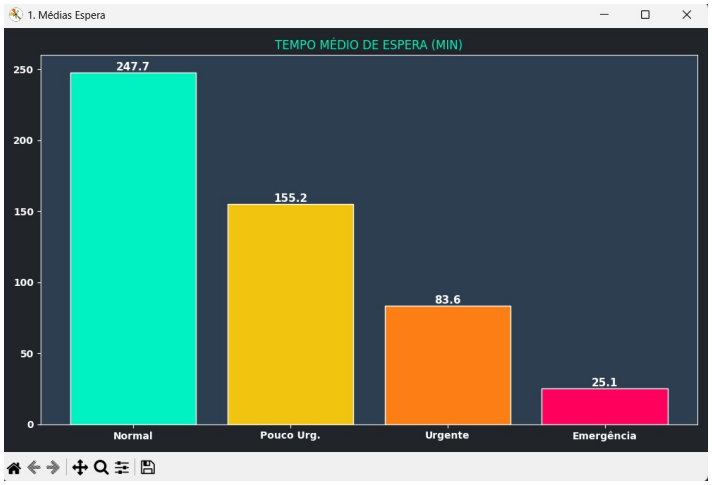

Figura 5: Tempos médios de espera distribuídos por nível de triagem.

4.4 Volume de Consultas por Especialidade

O volume de consultas representa o número de consultas por especialidade.

Interpretação: A análise do histograma de consultas permite retirar as seguintes conclusões:

- **Produtividade Relativa:** Mediante as simulações que vão sendo feitas, o número de consultas por especialidade vai alterando. Mas isto permite-nos fazer a correlação com a tabela de perfomance médica e perceber o porquê de certos médicos terem níveis de ocupação muito elevados.

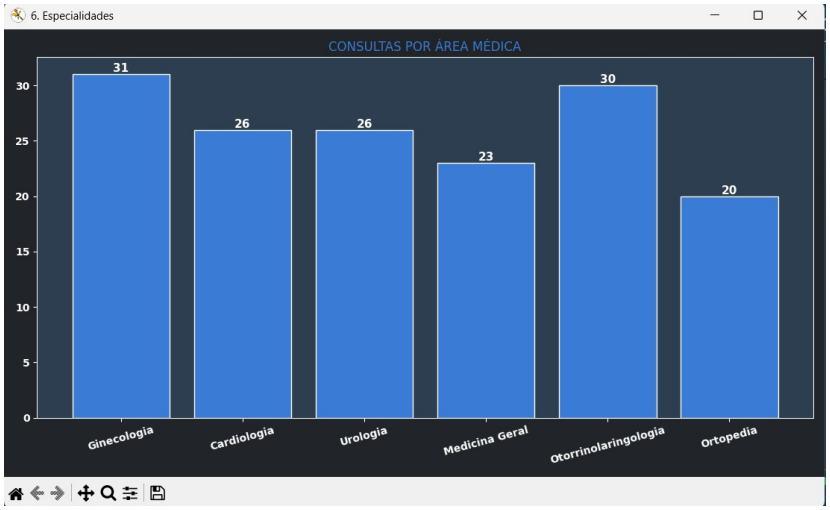

Figura 6: Volume total de consultas por especialidade médica

4.5 Atendimentos por Triagem

Esta métrica quantifica o número de consultas por nível de triagem à entrada da clínica médica.

Interpretação: A análise da distribuição de atendimentos (Figura 7) permite extrair as seguintes conclusões:

- Verificamos que, em níveis de maior prioridade, a quantidade de consultas é muito menor do que em níveis de menor prioridade. Na vida real, sabemos que a quantidade de pessoas que se deslocam ao hospital por consulta é superior à daqueles que chegam para as urgências.

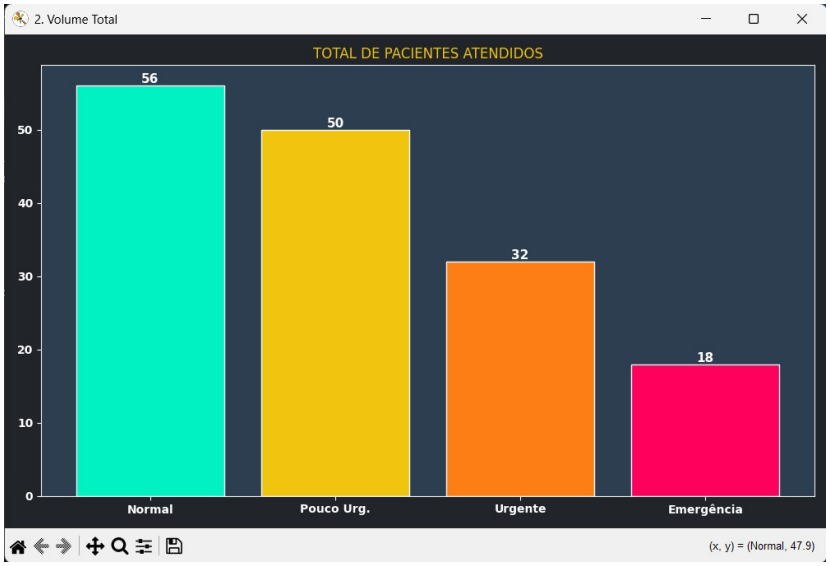

Figura 7: Número de consultas por nível de triagem

4.6 Taxa de Ocupação Total

A taxa de ocupação total é o indicador que resume a utilização global de todos os médicos da clínica durante o turno. Esta métrica faz a média de trabalho de toda a equipa, dando-nos uma visão geral de quão "cheio"ou "vazio"o hospital esteve naquelas 8 horas.

Interpretação: Olhar para a ocupação total (Figura 8) ajuda a perceber se o hospital está bem dimensionado:

- **Estado de Saturação:** Se a barra total estiver muito alta (perto dos 100%), significa que a clínica está sobrecarregada. Não há margem para imprevistos!
- **Desperdício de Recursos:** Se a ocupação total for muito baixa (por exemplo, 30%), isto indica que temos médicos a mais para o volume de pessoas que costumam aparecer.
- **Ponto de Equilíbrio:** O ideal é uma ocupação total entre os 70% e 80%. Isto mostra que os médicos estão a trabalhar bem, mas que o hospital ainda tem uma "reserva de energia"para lidar com uma emergência súbita.

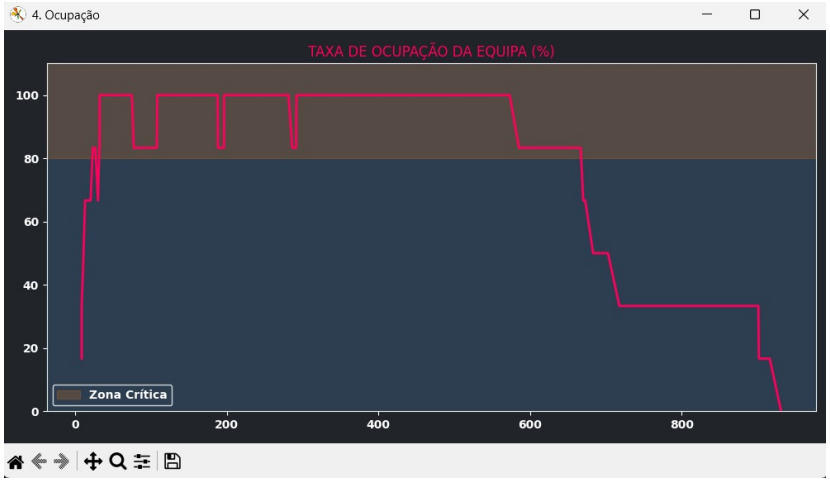

Figura 8: Gráfico da taxa de ocupação global de todos os recursos médicos da clínica

4.7 Stress Test: Impacto da Taxa de Chegada

O *stress test* à taxa de chegada foi realizado para identificar o "ponto de rutura"da clínica.

Interpretação: A análise dos dados sob stress (Figura 9) permite retirar conclusões críticas sobre a estabilidade do sistema:

- **Crescimento Exponencial das Filas:** Observou-se que existe um limite crítico onde a equipa deixa de conseguir processar as saídas à mesma velocidade das entradas. A partir desse ponto, a fila de espera deixa de ser linear e passa a crescer exponencialmente.
- **Saturação da Triagem:** Sob stress, os doentes com triagem **normal** e **pouco urgente** deixam de ser atendidos por completo, sendo "empurrados"indefinidamente por novas chegadas de casos **urgentes**.
- **Capacidade Nominal vs. Real:** O teste revelou que a clínica opera de forma estável até uma taxa de aproximadamente 35 doentes/hora.

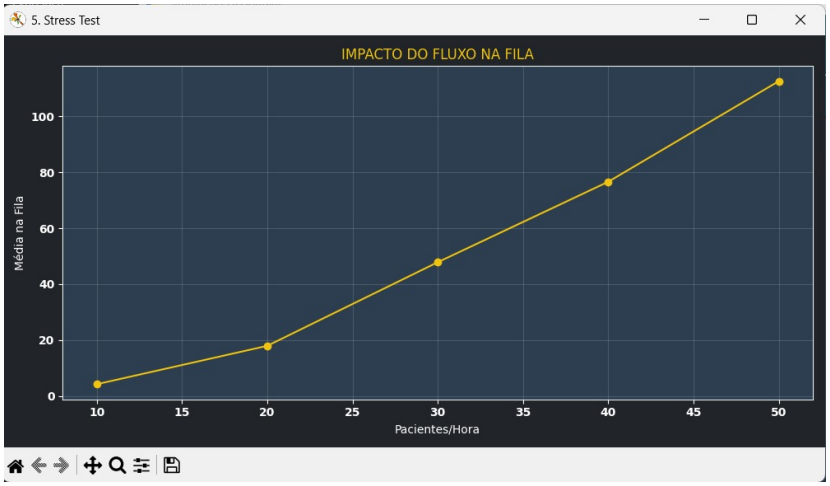

Figura 9: Análise comparativa da performance do sistema sob diferentes taxas de chegada

4.8 Evolução da Fila de Espera

A evolução da fila de espera ao longo do tempo descreve o equilíbrio entre a taxa de chegada de pacientes e a capacidade de rendimento da equipa médica.

Interpretação: A análise da curva de evolução da fila (Figura 10) permite extrair as seguintes conclusões técnicas:

- **Picos de Afluência:** O gráfico revela os períodos em que a `TAXA_CHEGADA` superou temporariamente a capacidade de atendimento.
- **Capacidade de Recuperação:** Se a curva desce rapidamente após um pico, significa que a equipa tem uma boa "velocidade de processamento".
- **Impacto das Desistências:** As quebras bruscas representam os momentos em que o tempo de espera excedeu a tolerância dos doentes, resultando na limpeza forçada da fila por **desistência**.

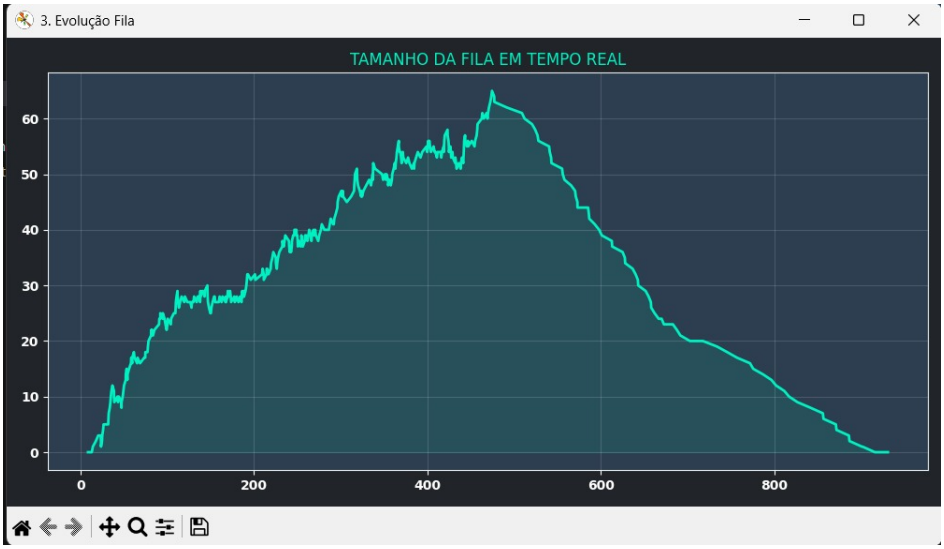

Figura 10: Gráfico do número total de pacientes na fila de espera

5 Conclusão

O projeto *Sistema de Gestão Hospitalar* foi concluído com sucesso, cumprindo todos os requisitos funcionais e não-funcionais propostos. A aplicação resultante é robusta, visualmente apelativa e permite simular cenários variados de afluência hospitalar.

A utilização de estruturas de dados, como filas de prioridade e a modularização do código foram fundamentais para o sucesso da implementação. A separação entre a lógica de simulação (`Base.py`) e a visualização (`Interface.py`) provou ser um desafio muito complexo, mas que acabou por ser frutífero, permitindo-nos ganhar perceções e conhecimentos que sem a elaboração do projeto não seria possível.

Referências

[1] FreeSimpleGUI. Freesimplegui documentation - a simple gui framework for python. `https://github.com/spyoungtech/FreeSimpleGUI`, 2024. Acedido em: Janeiro 2026.

[2] John D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[3] Ecma International. The json data interchange standard. `https://www.json.org/json-en.html`, 2017.