Sistema de simulação no âmbito hospitalar

para

MODSS

Preparado por:
Tiago Nora (1201050)
João Figueiredo (1230194)

Instituto Superior de Engenharia do Porto, Porto
Sistemas de Informação e Conhecimento
Modelação e Simulação Inteligente
Paulo Sérgio dos Santos Matos (PSM)

Porto, 14 de junho de 2024

Conteúdo

Lista de Figuras

Listagens

Capítulo 1

Introdução

Neste capitulo é feita uma contextualização do trabalho desenvolvido onde são apresentados os objetivos definidos pelo grupo e por último é apresentado a divisão do documento.

1.1 Contextualização

No panorama atual dos cuidados de saúde, a gestão eficaz dos recursos no contexto hospitalar é fundamental para prestar cuidados de elevada qualidade aos doentes, ao mesmo tempo que se ultrapassam as restrições operacionais e as pressões financeiras. A natureza dinâmica da procura por parte dos doentes, aliada às limitações de recursos e às necessidades de cuidados de saúde em constante evolução, sublinha a importância de adotar estratégias inovadoras para otimizar a utilização dos recursos e melhorar a eficiência operacional. Neste contexto, a simulação surge como uma ferramenta poderosa para modelar e analisar sistemas complexos, oferecendo uma visão sobre a intrincada interação entre vários fatores operacionais e o seu impacto no desempenho global.

A gestão eficiente dos recursos em ambientes hospitalares é fundamental para garantir a qualidade dos cuidados prestados aos doentes, otimizando simultaneamente a eficácia operacional.

1.2 Objetivos

No que toca a objetivos desta parte do trabalho foram identificados os seguintes:

- Estudo sobre sistemas que já tenham sido aplicados em contexto real
- Simular diferentes cenários clínicos para otimizar protocolos de atendimento

- Avaliar e melhorar o fluxo de pacientes no hospital
- Facilitar estudos clínicos e pesquisas operacionais num ambiente controlado.
- Obter dados precisos sobre a eficácia de intervenções clínicas e operacionais.

1.3 Divisão do documento

O presente documento está divido nos seguintes capítulos:

- Capítulo 1: Introdução
- Capítulo 2: Compreensão do sistema
- Capítulo 3: Implementação do sistema
- Capítulo 4: Conclusão

Capítulo 2

Compreensão do sistema

Neste capítulo é descrito o sistema, nomeadamente o recurso hospitalares, as etapas e as caraterísticas do paciente.

2.1 Descrição do sistema

O diagrama apresentado na Figura 2.1 descreve o processo de atendimento de um paciente no hospital. O paciente chega ao hospital e, se a equipa da receção estiver ocupada, ele precisa esperar antes de fazer o registo. Quando a equipa de receção está disponível, o paciente faz o registo e então aguarda a consulta ou tratamento. Pacientes em estado crítico são encaminhados diretamente para intervenção de emergência. Pacientes críticos que necessitam de intervenção de emergência são atendidos imediatamente. Após a intervenção, o paciente vai para observação na UTI. **Caso haja alguma complicação, o paciente volta para uma sala de intervenção.** Após passar por observação, tratamento e qualquer outra intervenção necessária, o paciente entra na fase de planeamento de alta. Com um plano de alta definido, o paciente pode finalmente deixar o hospital. Em casos graves, onde há complicações significativas e o tratamento não é bem-sucedido, pode ocorrer o óbito. Uma vez registado, o paciente aguarda por uma consulta. Quando uma sala e um médico estiver disponível, ele passa pela consulta. Dependendo da avaliação médica, o paciente pode precisar de exames médicos. Após a consulta e possíveis exames, o paciente pode ser direcionado para tratamento. Se a condição do paciente exigir, ele pode ser encaminhado para cirurgia. Após a cirurgia, se houver uma baixa chance de complicações, o paciente é enviado para observação na UTI. Se houver complicações, ele pode ser observado numa área geral ou receber tratamento adicional. Após a utilização da sala de operações ou sala de cirurgia a sala deve ser limpa e só volta a ficar disponível após realizada essa limpeza.

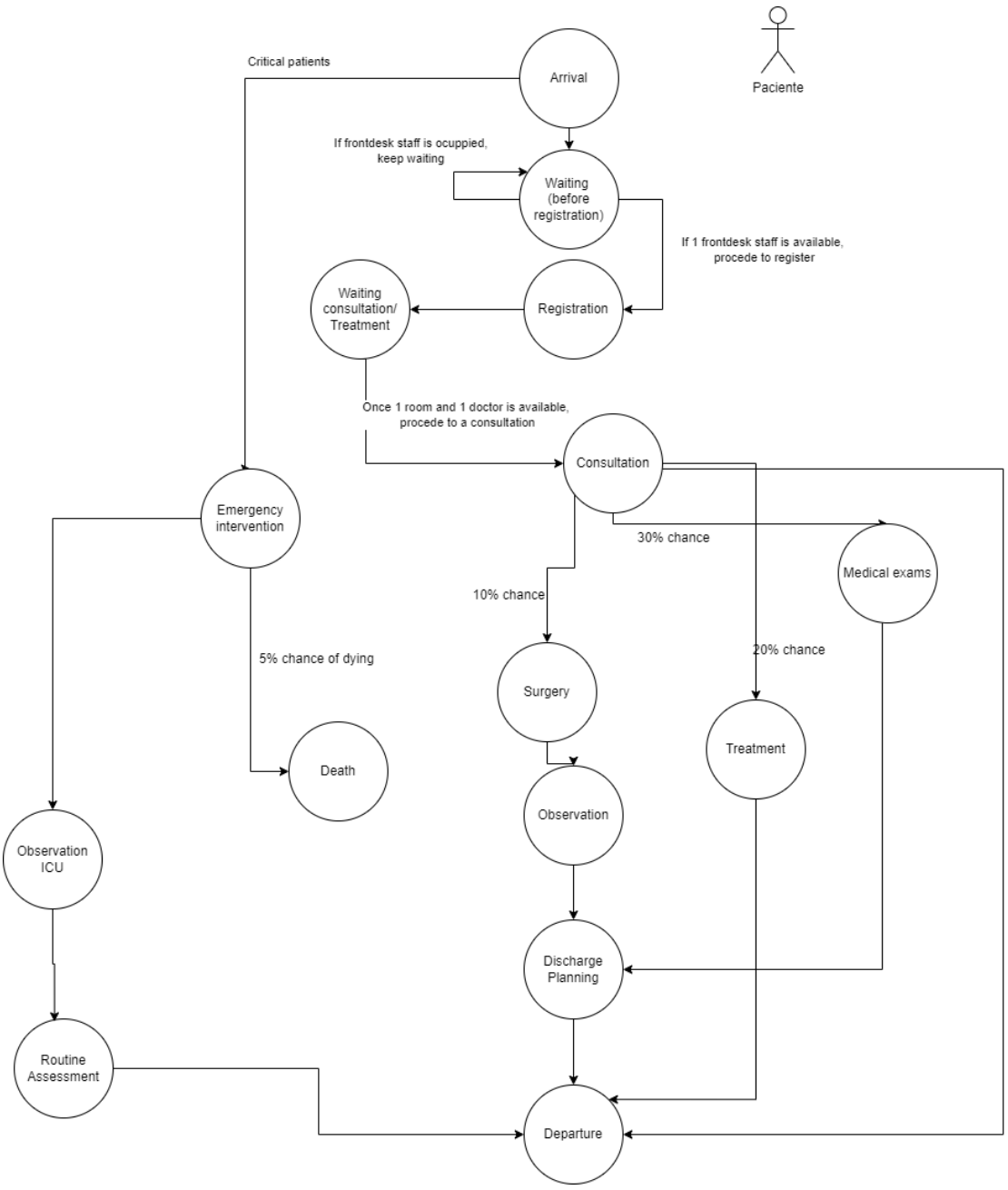

Figura 2.1: Diagrama de estados

2.2 Descrição dos recursos hospitalares

No que toca a recursos hospitalares neste sistema de simulação são considerados os seguintes:

- Médicos — tratamento urgente: Responsáveis pela realização de cirurgias.
- Médicos — tratamento normal: Responsáveis pela realização de consultas
- Enfermeiros: Responsáveis pelo tratamento dos pacientes.
- Salas de Tratamento: Espaços físicos onde os tratamentos ocorrem.
- Salas de tratamento cuidados intensivos: Espaços físicos onde os tratamentos ocorrem.
- Rececionistas: Responsáveis pela receção dos pacientes
- Staff de limpeza: Responsáveis pela limpeza dos blocos operatórios
- Blocos de cirurgias: Espaços físicos onde se realizam operações

2.3 Etapas

Neste sistema as etapas que um certo paciente pode seguir são as seguintes:

1. Chegada (Arrival)
    - O paciente chega ao hospital.
2. Espera (antes do registro) [Waiting (before registration)]
    - Se o pessoal da receção estiver ocupado, o paciente deve esperar.
3. Registro (Registration)
    - Quando um membro da receção estiver disponível, o paciente procede para o registro.
4. Espera para consulta/tratamento (Waiting consultation/Treatment)
    - Após o registro, o paciente espera por uma consulta ou tratamento.
5. Consulta (Consultation)
    - Assim que um quarto e um médico estiverem disponíveis, o paciente passa por uma consulta.

6. Exames médicos (Medical exams)
    - Durante a consulta, se necessário, o paciente pode ser encaminhado para exames médicos.
7. Tratamento (Treatment)
    - Com base na consulta e nos exames, o paciente pode receber tratamento.
8. Observação (Observation)
    - Após o tratamento, o paciente pode ser colocado em observação.
9. Planeamento da alta (Discharge Planning)
    - Se a observação for satisfatória e não houver complicações, inicia-se o planeamento da alta do paciente.
10. Saída (Departure)
    - O paciente recebe alta e sai do hospital.
11. Intervenção de emergência (Emergency intervention)
    - Pacientes críticos podem ser encaminhados diretamente para uma intervenção de emergência.
12. Cirurgia (Surgery)
    - Se necessário, o paciente pode passar por uma cirurgia.
13. Observação na UTI (Observation ICU)
    - Após a cirurgia, se houver uma baixa hipótese de complicações, o paciente é colocado em observação na UTI.
14. Limpeza da sala de operações
    - Após a realização de um operação num das salas de operações, esta deve ser fechada após a conclusão da operação e deste modo deve ser procedido à limpeza da mesma.

2.4 Paciente

Neste sistema foram gerados vários pacientes ao longo da simulação e por isso foram definidos alguns parâmetros que este deveria ter. Por isso para cada paciente gerado foram gerados os seguintes parâmetros:

- Id
- Nome
- Sexo
- Data de nascimento
- Endereço
- Idade
- Prioridade
  - 1: Prioridade critica
  - 2: Prioridade vulnerável
  - 3: Prioridade mobilidade reduzida
  - 4: Prioridade moderada
  - 5: Prioridade não urgente

Capítulo 3

Implementação do sistema

Neste capítulo são mencionadas as tecnologias utilizadas, como foi efetivamente implementado o sistema e por último é mencionada a validação do sistema.

3.1 Implementação da simulação

A implementação da simulação de um sistema hospitalar é uma tarefa complexa que requer a modelagem detalhada de diversos processos e recursos. Nesta secção, descrevemos os passos e decisões tomadas para configurar e executar o modelo de simulação utilizando a biblioteca SimPy. O objetivo principal é replicar o funcionamento dinâmico do hospital, desde a chegada dos pacientes até a alta, levando em consideração a interação entre diferentes áreas e a alocação eficiente dos recursos disponíveis.

3.1.1 Importe das bibliotecas necessárias

Na Listagem 3.1 são apresentadas as bibliotecas importadas e necessários para o funcionamento do sistema.

```
1 import simpy
2 import random
3 from faker import Faker
4 import os
5 from datetime import datetime
6 import streamlit as st
7 import pandas as pd
8 import plotly.express as px
```

Listagem 3.1: Importe das bibliotecas necessárias.

3.1.2 Definição de tempo

Para todas as atividades mencionadas no sistema foram definidos intervalos, nomeadamente os seguintes:

- Registo: 3 a 7 minutos
- Consulta: 5 a 15 minutos
- Exames: 10 a 20 minutos
- Tratamento: 15 a 25 minutos
- Cirurgia: 20 a 40 minutos
- Observação: 50 a 70 minutos
- Planeamento de alta: 5 a 15 minutos
- Limpeza da sala: 10 a 20 minutos
- UTI (Unidade de Terapia Intensiva): 1440 a 2880 minutos (1 a 2 dias)
- Despedida: 5 a 10 minutos

3.1.3 Criação dos pacientes

A criação e o controlo de fluxo por parte destes no hospital é feito através da utilização de duas funções apresentadas na Listagem 3.2 e na Listagem 3.3.

Na Listagem 3.2 é apresentada a função "criar_paciente". Esta função é responsável por criar um paciente com dados fictícios, como o identificador, nome, endereço, sexo, data de nascimento, idade e a prioridade de atendimento do paciente. É importante mencionar que o nível mais prioritário tem uma menor chance de ser escolhido, sendo assim o número de pacientes com um nível menor de prioridade será maior.

```
def criar_paciente():
    d_nascimento = fake.date_of_birth().strftime("%Y-%m-%d")
    dob = datetime.strptime(d_nascimento, "%Y-%m-%d")
    current_date = datetime.now()
    age = current_date.year - dob.year - ((current_date.month, current_date.day) < (
        dob.month, dob.day))
    priority_levels = [1, 2, 3, 4, 5]
    weights = [1, 2, 3, 4, 5]
    prioridade = random.choices(priority_levels, weights, k=1)[0]
    return {
        'id': fake.uuid4(),
        'nome': fake.name(),
        'endereco': fake.address(),
        'sexo': fake.random_element(elements=('M', 'F')),
        'data_nascimento': d_nascimento,
        'idade': age,
        'Prioridade': prioridade
    }
```

Listagem 3.2: Criação do paciente.

Por sua vez, na Listagem 3.3 é apresentada a função "gerar_pacientes". Esta função, primeiramente, utiliza a função "expovariate" para determinar os intervalos de tempo entre a chegada de grupos de pacientes. Esse método gera intervalos de tempo aleatórios recorrendo a uma distribuição exponencial com uma média de 10 minutos. A quantidade de pacientes que chegam ao mesmo tempo, é determinada aleatoriamente, variando de 1 a 3 pacientes por grupo. Isso introduz uma variação realista na quantidade de pacientes admitidos simultaneamente, o que é comum em hospitais onde a entrada de pacientes pode ocorrer em grupos, especialmente em emergências ou picos de necessidade pelo serviço. Para cada paciente que chega, a função "gerar_pacientes" chama a função "criar_paciente". Após a criação dos dados do paciente, a função "gerar_pacientes" inicia um novo processo para cada paciente no ambiente de simulação. Esse processo representa a entrada do paciente no sistema hospitalar virtual, permitindo simular não apenas a chegada de pacientes, mas também o subsequente tratamento e movimentação no hospital conforme as regras e lógicas definidas na simulação.

```
def gerar_pacientes(env, hospital):
    while True:
        yield env.timeout(random.expovariate(1/10))
        num_pacientes = random.randint(1, 3)
        for _ in range(num_pacientes):
            dados_paciente = criar_paciente()
            env.process(paciente(env, dados_paciente, hospital))
```

Listagem 3.3: Processamento dos pacientes criados.

3.1.4 Inicio da simulação

Na Listagem 3.4 é apresentada a criar do ambiente de simulação, atribuindo-o à variável "env", neste caso o objeto "env" é uma instância da classe "simpy.Environment", essencial para gerir e controlar a execução da simulação. Em seguida, um objeto hospital é criado, este representa um hospital na simulação. A classe Hospital é uma classe anteriormente definida com métodos e atributos que descrevem as operações e características do hospital. O objeto "env" é passado como argumento no construtor da classe Hospital, o que permite que o hospital tenha acesso ao ambiente de simulação e possa interagir com ele durante a execução da simulação. Para simular a chegada contínua de pacientes ao hospital, um novo processo na simulação é iniciado, onde é utilizado o método "env.process". O processo chama a função "gerar_pacientes", onde esta recebe o objeto "env" e a classe hospital como argumentos, permitindo-lhe interagir com o ambiente de simulação e com as características específicas do hospital. Por fim, a simulação é executada com o comando "env.run(until=24*60*7)". Este comando mantém a simulação em execução por um período de 7 dias, equivalentes a 24 horas multiplicadas por 60 minutos, multiplicadas novamente por 7 dias. Durante esse tempo, todos os processos definidos anteriormente, incluindo a criação de pacientes e outras operações simuladas, ocorrerão conforme as regras e lógicas programadas na simulação.

```
1 env = simpy.Environment()
2 hospital = Hospital(env)
3 env.process(gerar_pacientes(env, hospital))
4 env.run(until=24*60*7)
```

Listagem 3.4: Inicio da simulação.

3.1.5 Operações

- registrar(self, paciente): Simula o registo de um paciente.
- consulta(self, paciente): Simula a consulta de um paciente.
- exames(self, paciente): Simula a realização de exames num paciente.
- tratamento(self, paciente): Simula o tratamento de um paciente.
- cirurgia(self, paciente): Simula a realização de uma cirurgia.
- observacao(self, paciente): Simula a transferência de um paciente para observação.
- planeamento_alta(self, paciente): Simula o planeamento de alta de um paciente.
- limpeza_sala(self): Simula a limpeza de uma sala de cirurgia.

- icu(self, paciente): Simula a admissão de um paciente na Unidade de Terapia Intensiva
- despedida(self, paciente): Simula a saída do paciente

Cada um destes métodos utiliza a função random.uniform para gerar um tempo aleatório dentro de um intervalo definido, simulando assim a variabilidade dos tempos dos procedimentos.

3.1.6 Definição dos recursos do hospital

Na Listagem 3.5 é apresentado a inicialização da classe "Hospital" com capacidades específicas para diferentes recursos.

```
1  class Hospital:
2      def __init__(self, env, reception_capacity, consulta_capacity, exames_capacity,
           doctors_capacity, surgeons_capacity, nurses_capacity,
           surgery_room_capacity, icu_capacity, cleaning_capacity):
3          self.env = env
4          self.recepcao = simpy.PriorityResource(env, capacity=reception_capacity)
5          self.sala_consulta = simpy.PriorityResource(env, capacity=consulta_capacity)
6          self.sala_exames = simpy.PriorityResource(env, capacity=exames_capacity)
7          self.doctors = simpy.PriorityResource(env, capacity=doctors_capacity)
8          self.surgeons = simpy.PriorityResource(env, capacity=surgeons_capacity)
9          self.nurses = simpy.PriorityResource(env, capacity=nurses_capacity)
10         self.sala_cirurgia = simpy.PriorityResource(env, capacity=
               surgery_room_capacity)
11         self.uti = simpy.PriorityResource(env, capacity=icu_capacity)
12         self.sala_limpeza = simpy.PriorityResource(env, capacity=cleaning_capacity)
```

Listagem 3.5: Definição dos recursos do hospital.

3.1.7 Definição das atividades

Na Listagem 3.6 é apresentado a função "paciente", que simula o percurso de um paciente no hospital. A função começa por verificar se o paciente tem prioridade crítica. Se a prioridade for crítica, a função chamada outra função com o nome "critical_patient", responsável por tratar pacientes críticos. Para pacientes que não são críticos, a função procede para solicitar atendimento na receção. A função regista o tempo atual antes de esperar pela disponibilidade do recurso e, após a espera, calcula o tempo de espera na receção. Este tempo de espera é então adicionado à lista de tempos de espera no dicionário stats. Depois do registro, a função solicita simultaneamente uma sala de consulta e um médico. Após a consulta, há uma chance de 30% de que o paciente

precise de exames adicionais. Se o paciente precisar de exames, a função solicita uma sala de exames e espera até que o recurso esteja disponível. Em seguida, a função processa os exames do paciente. Além disso, há uma chance de 20% de que o paciente precise de tratamento adicional após os exames. Se o paciente precisar de tratamento, a função inicia o processo de tratamento. Há também uma chance de 10% de que o paciente precise de cirurgia. Se o paciente precisar de cirurgia, a função solicita uma sala de cirurgia e espera até que o recurso esteja disponível. Após a cirurgia, a função processa a limpeza da sala de cirurgia. Depois, há uma chance de 20% de que o paciente precise ser observado. Se o paciente precisar de observação, a função inicia o processo de observação. Caso contrário, a função inicia o processo de planeamento da alta do paciente. Caso não aconteça nenhuma das atividades (exame, tratamento e cirurgia) a função inicia diretamente o processo de dar alta ao paciente.

```
def paciente(env, dados_paciente, hospital, stats):
    if dados_paciente['Prioridade'] == PRIORITY_CRITICAL:
        yield env.process(critical_patient(env, dados_paciente, hospital, stats))
        return

    with hospital.recepcao.request(priority=dados_paciente['Prioridade']) as request_recepcao:
        start_wait_recepcao = env.now
        yield request_recepcao
        wait_time_recepcao = env.now - start_wait_recepcao
        stats['reception_wait_times'].append(wait_time_recepcao)
        yield env.process(hospital.registrar(dados_paciente))

    with hospital.sala_consulta.request(priority=dados_paciente['Prioridade']) as request_consulta, \
         hospital.doctors.request(priority=dados_paciente['Prioridade']) as request_doctor:

        start_wait_consulta = env.now
        yield request_consulta & request_doctor
        if env.now > start_wait_consulta:
            wait_time_consulta = env.now - start_wait_consulta
        else:
            wait_time_consulta = 0
        stats['consulta_wait_times'].append(wait_time_consulta)
        yield env.process(hospital.consulta(dados_paciente))

        if random.random() < 0.3:
            with hospital.sala_exames.request(priority=dados_paciente['Prioridade']) as request_exames:
                yield request_exames
                yield env.process(hospital.exames(dados_paciente))

        if random.random() < 0.2:
            yield env.process(hospital.tratamento(dados_paciente))

        if random.random() < 0.1:
            with hospital.sala_cirurgia.request(priority=dados_paciente['Prioridade']) as request_cirurgia:
                yield request_cirurgia
                yield env.process(hospital.cirurgia(dados_paciente))
                yield env.process(hospital.limpeza_sala())
                if random.random() < 0.2:
                    yield env.process(hospital.observacao(dados_paciente))
                else:
                    yield env.process(hospital.plano_alta(dados_paciente))
        else:
            yield env.process(hospital.plano_alta(dados_paciente))
```

Listagem 3.6: Definição das atividades do paciente.

3.1.8 Definição das atividades de paciente crítico

Na Listagem 3.7 é apresentado a função "critical_patient". Esta função é projetada para lidar com pacientes críticos no hospital.

Na função, utiliza-se um contexto with para solicitar simultaneamente vários recursos essenciais para a realização de uma cirurgia crítica. Estes recursos são:

- Um cirurgião, solicitado através de hospital.surgeons.request, com prioridade definida como PRIORITY_CRITICAL.
- Duas enfermeiras, cada uma solicitada através de hospital.nurses.request, também com prioridade definida como PRIORITY_CRITICAL.
- Uma sala de cirurgia, solicitada através de hospital.sala_cirurgia.request, com prioridade definida como PRIORITY_CRITICAL.

A função então entra em um estado de espera até que todos os recursos solicitados estejam disponíveis. Isso é feito utilizando o comando yield para cada solicitação:

- yield request_surgeon: Espera até que um cirurgião esteja disponível.
- yield request_nurse_1: Espera até que a primeira enfermeira esteja disponível.
- yield request_nurse_2: Espera até que a segunda enfermeira esteja disponível.
- yield request_surgery_room: Espera até que uma sala de cirurgia esteja disponível.

Uma vez que todos os recursos estejam disponíveis, a função regista o tempo atual em start_time antes de iniciar a cirurgia. Em seguida, a função inicia o processo de cirurgia do paciente utilizando yield env.process(hospital.cirurgia(dados_paciente)). Após a conclusão da cirurgia, o tempo de duração da cirurgia é calculado como a diferença entre o tempo atual e start_time, e esse tempo é adicionado à lista de tempos de cirurgia no dicionário de estatísticas stats.

Após a cirurgia, há uma chance de cinco por cento de que o paciente não sobreviva aos procedimentos. Esta decisão é feita através do código "if random.random() < 0.05". Se o paciente não precisar de mais procedimentos, a função retorna. Caso contrário, a função inicia o processo de limpeza da sala de cirurgia.

A função então prossegue para solicitar simultaneamente uma sala de consulta e um médico. Os recursos são solicitados da seguinte forma:

- Uma sala de consulta, solicitada através de hospital.sala_consulta.request, com prioridade definida pela prioridade do paciente.

- Um médico, solicitado através de hospital.doctors.request, também com prioridade definida pela prioridade do paciente.

Dentro deste contexto, a função regista o tempo atual em start_wait_consulta antes de esperar pela disponibilidade dos recursos. Em seguida, a função espera até que ambos os recursos estejam disponíveis utilizando. Após a espera, a função calcula o tempo de espera para a consulta com a diferença entre o tempo atual (env.now) e start_wait_consulta. Este tempo de espera é então adicionado à lista de tempos de espera para consulta no dicionário de estatísticas stats.

Após calcular e registar o tempo de espera para consulta, a função inicia o processo de consulta do paciente. Depois da consulta, há uma chance de trinta por cento de que o paciente precise de exames adicionais. Esta decisão é feita através do código if random.random() $<$ 0.3. Se o paciente precisar de exames, a função solicita uma sala de exames, com prioridade definida pela prioridade do paciente. A função então espera até que a sala de exames esteja disponível. Em seguida, a função inicia o processo de exames do paciente.

Após os exames, há uma chance de vinte por cento de que o paciente precise de tratamento adicional. Esta decisão é feita através do código if random.random() $<$ 0.2. Se o paciente precisar de tratamento, a função inicia o processo de tratamento.

Há uma chance de dez por cento de que o paciente precise de uma segunda cirurgia. Esta decisão é feita através do código if random.random() $<$ 0.1. Se o paciente precisar de uma segunda cirurgia, a função solicita uma sala de cirurgia, com prioridade definida pela prioridade do paciente. A função então espera até que a sala de cirurgia esteja disponível. Em seguida, a função inicia o processo de cirurgia. Após a cirurgia, a sala fica indisponível e por sua vez a função inicia o processo de limpeza da sala de cirurgia.

Após a segunda cirurgia e limpeza da sala ter sido realizada, há uma chance de vinte por cento de que o paciente precise ser observado. Esta decisão é feita através do código if random.random() $<$ 0.2. Se o paciente precisar de observação, a função inicia o processo de observação. Caso contrário, a função inicia o planeamento da alta.

Se o paciente não precisar de uma segunda cirurgia, a função inicia diretamente o processo de planeamento da alta.

```
1  def critical_patient(env, dados_paciente, hospital, stats):
2      with hospital.surgeons.request(priority=PRIORITY_CRITICAL) as
           request_surgeon, \
3           hospital.nurses.request(priority=PRIORITY_CRITICAL) as
                request_nurse_1, \
4           hospital.nurses.request(priority=PRIORITY_CRITICAL) as
                request_nurse_2, \
5           hospital.sala_cirurgia.request(priority=PRIORITY_CRITICAL) as
                request_surgery_room:
6
7          yield request_surgeon
8          yield request_nurse_1
9          yield request_nurse_2
10         yield request_surgery_room
11         start_time = env.now
12         yield env.process(hospital.cirurgia(dados_paciente))
13         stats['surgery_times'].append(env.now - start_time)
14         if random.random() < 0.05:
15             return
16         yield env.process(hospital.limpeza_sala())
17
18     with hospital.sala_consulta.request(priority=dados_paciente['Prioridade']) as
           request_consulta, \
19          hospital.doctors.request(priority=dados_paciente['Prioridade']) as
                request_doctor:
20         start_wait_consulta = env.now
21         yield request_consulta & request_doctor
22         if env.now > start_wait_consulta:
23             wait_time_consulta = env.now - start_wait_consulta
24         else:
25             wait_time_consulta = 0
26         stats['consulta_wait_times'].append(wait_time_consulta)
27         yield env.process(hospital.consulta(dados_paciente))
28         if random.random() < 0.3:
29             with hospital.sala_exames.request(priority=dados_paciente['Prioridade'])
                    as request_exames:
30                 yield request_exames
31                 yield env.process(hospital.exames(dados_paciente))
32         if random.random() < 0.2:
33             yield env.process(hospital.tratamento(dados_paciente))
34         if random.random() < 0.1:
35             with hospital.sala_cirurgia.request(priority=dados_paciente['Prioridade'])
                    as request_cirurgia:
36                 yield request_cirurgia
37                 yield env.process(hospital.cirurgia(dados_paciente))
38                 yield env.process(hospital.limpeza_sala())
39                 if random.random() < 0.2:
40                     yield env.process(hospital.observacao(dados_paciente))
41                 else:
42                     yield env.process(hospital.plano_alta(dados_paciente))
43         else:
44             yield env.process(hospital.plano_alta(dados_paciente))
```

Listagem 3.7: Definição das atividades do paciente crítico.

3.2 Implementação da interface

Na Listagem 3.8 é definido o título da aplicação no Streamlit como “Simulação de Hospital”.

```
1 st.title('Simulação de Hospital')
```

Listagem 3.8: Titulo da simulação.

Na Listagem 3.9 é criado um controle deslizante na barra lateral para o utilizador selecionar a duração da simulação em dias, com um valor padrão de 7 dias.

```
1 sim_duration = st.sidebar.slider('Duração da Simulação (dias)', 1, 30, 7)
```

Listagem 3.9: Titulo da simulação.

Na Listagem 3.10 é criado vários controles deslizantes na barra lateral para o utilizador configurar as capacidades e o número de profissionais disponíveis no hospital.

```
1  recursos = {
2      'recepcao': st.sidebar.slider('Capacidade da Recepção', 1, 10, 4),
3      'consulta': st.sidebar.slider('Capacidade da Sala de Consulta', 1, 10, 4),
4      'exames': st.sidebar.slider('Capacidade da Sala de Exames', 1, 10, 4),
5      'medicos': st.sidebar.slider('Número de Médicos', 1, 10, 4),
6      'cirurgioes': st.sidebar.slider('Número de Cirurgiões', 1, 10, 4),
7      'enfermeiros': st.sidebar.slider('Número de Enfermeiros', 1, 20, 10),
8      'cirurgia': st.sidebar.slider('Capacidade da Sala de Cirurgia', 1, 5, 3),
9      'uti': st.sidebar.slider('Capacidade da UTI', 1, 10, 4),
10     'limpeza': st.sidebar.slider('Capacidade da Sala de Limpeza', 1, 10, 4)
11 }
```

Listagem 3.10: Titulo da simulação.

Na Listagem 3.11 é apresentado o código executado quando o botão “Iniciar Simulação” é pressionado, um ambiente SimPy é criado, e a simulação começa. A função gerar_pacientes gera pacientes que serão processados pelo hospital e por sua vez é definido o tempo de simulação nomeadamente 24*60*sim_duration minutos (equivalente ao número de dias selecionados).

```
1 if st.button('Iniciar Simulação'):
2     env = simpy.Environment()
3     hospital = Hospital(env, recursos)
4     records = []
5     critical_stats = {'sobreviventes': 0, 'mortes': 0}
6     env.process(gerar_pacientes(env, hospital, records, critical_stats))
7     env.run(until=24*60*sim_duration)
```

Listagem 3.11: Titulo da simulação.

3.2.1 Estatística apresentada

Na Figura 3.1 é apresentado o botão que pode ser utilizado para se obter o ficheiro da simulação que apresenta todas as operações.

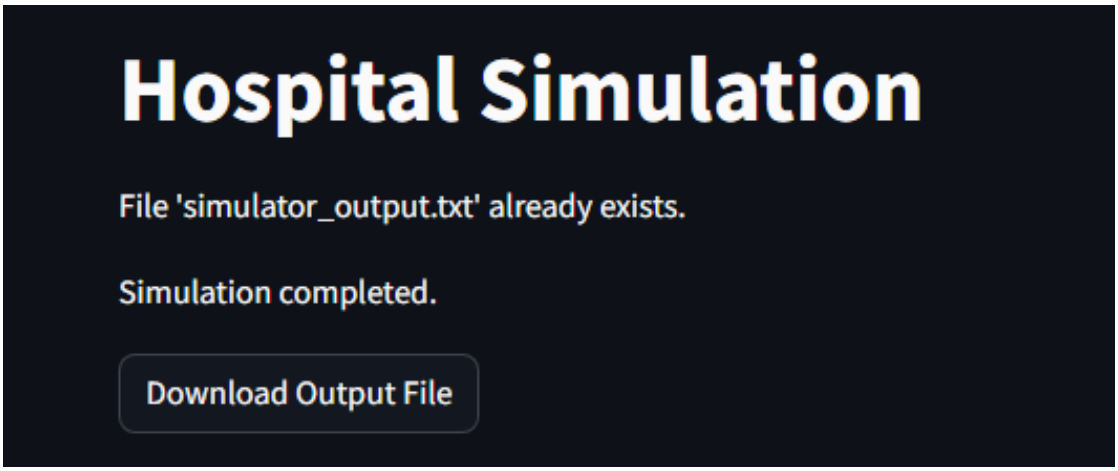

Figura 3.1: Ficheiro da simulação

Na Figura 3.2 é apresentada uma tabela com os valores médios de espera.

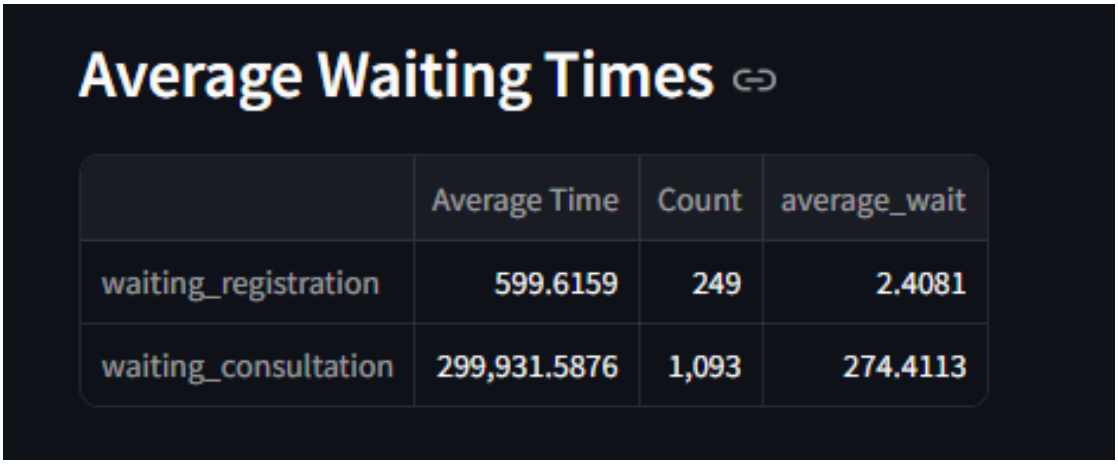

Figura 3.2: Tabela com os valores médios de espera

Na Figura 3.3 é apresentado um gráfico com os valores médios de espera.

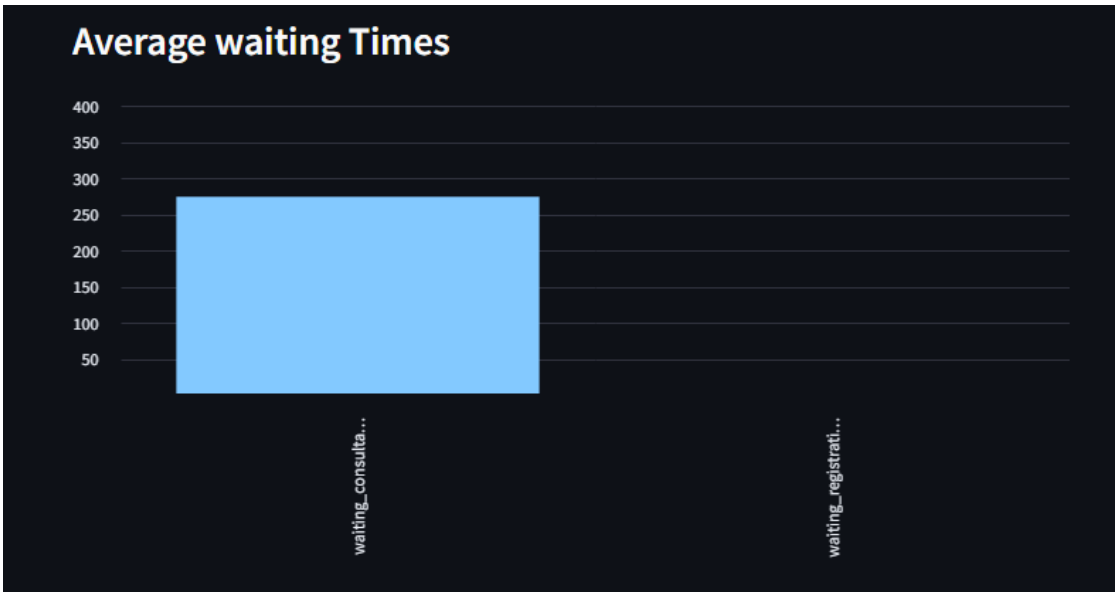

Figura 3.3: Gráfico com os valores médios de espera

Na Figura 3.4 é apresentada uma tabela com o número de vezes que um certo evento aconteceu.

Action Counts

	Action	Count
0	chegada	2,128
1	registo	1,876
2	espera_registo	249
3	espera_consulta	1,093
4	consulta	1,215
5	exames	386
6	tratamentos	255
7	observação	18
8	plano_alta	1,193
9	saida	1,082

Figura 3.4: Tabela com o número de vezes que um evento acontece

Na Figura 3.5 é apresentado um diagrama com a distribuição dos vários níveis de prioridade atribuídos aos pacientes.

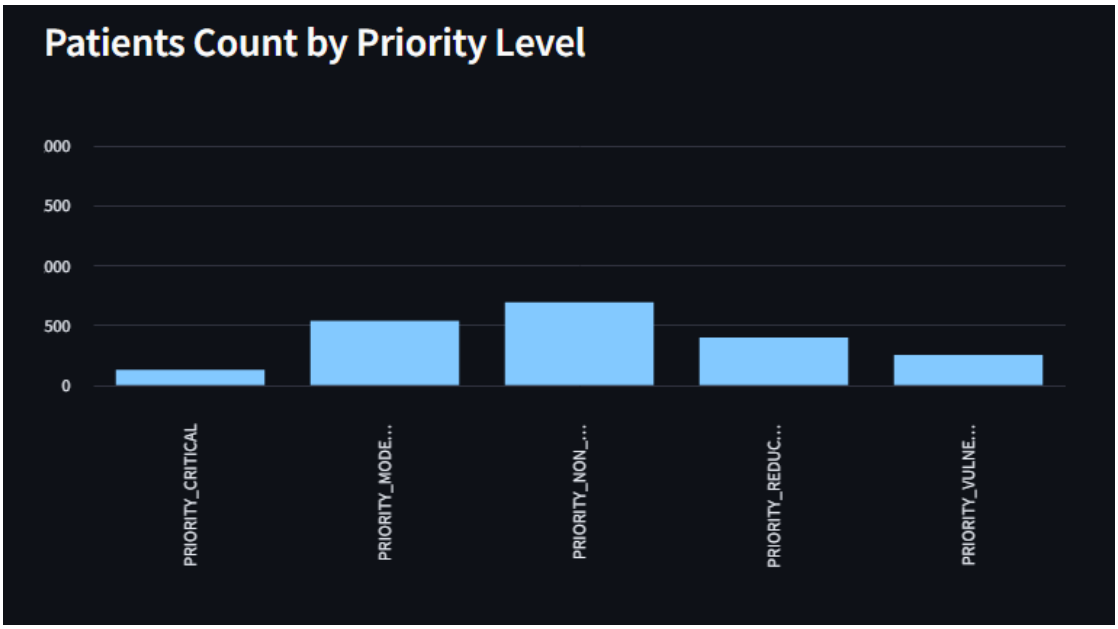

Figura 3.5: Distribuição das prioridades pelos pacientes

Na Figura 3.6 é apresentado um gráfico com a distribuição dos vários eventos.

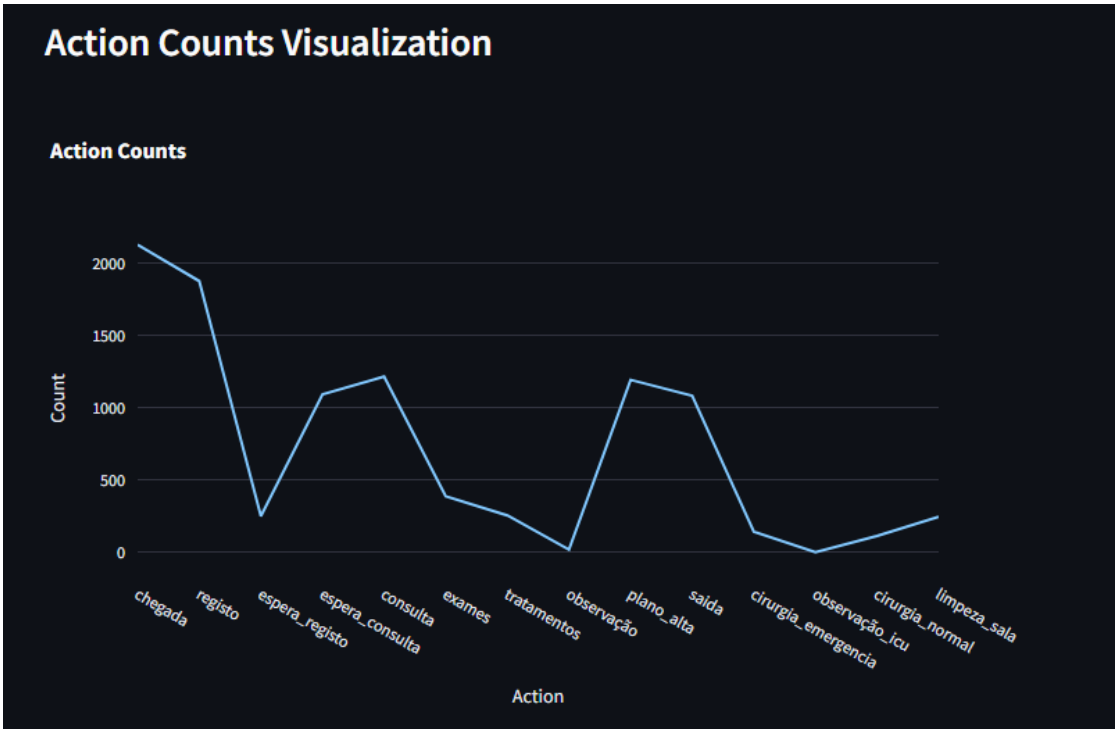

Figura 3.6: Gráfico com a distribuição dos vários eventos

Na Figura 3.7 é apresentado um gráfico com a distribuição do estado dos pacientes críticos.

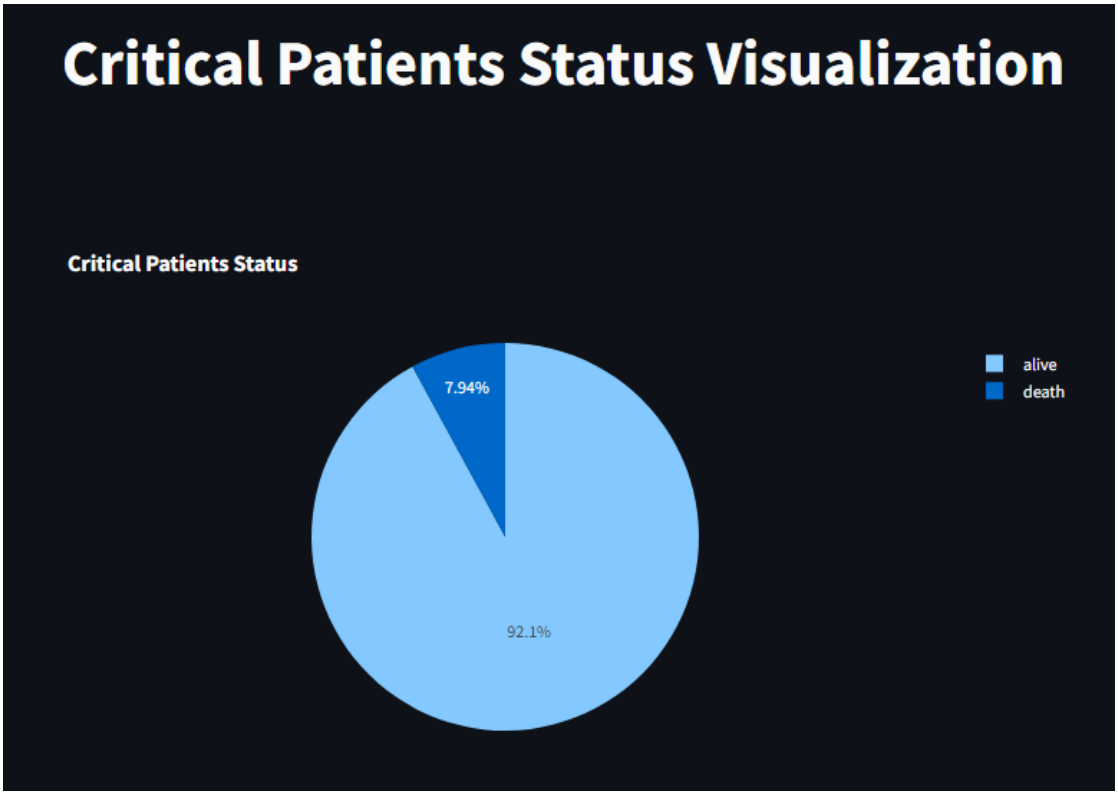

Figura 3.7: Gráfico com a distribuição do estado dos pacientes críticos

3.3 Tecnologias utilizadas

Na realização deste sistema foram utilizadas as seguintes tecnologias e ferramentas:

- Streamlit: O Streamlit é uma biblioteca de código aberto em Python utilizada para a criação de aplicações web interativas e dinâmicas de forma rápida e fácil. Destina-se principalmente a cientistas de dados e engenheiros, permitindo-lhes transformar scripts de dados em aplicações web interativas com apenas algumas linhas de código. Com o Streamlit, é possível adicionar componentes como gráficos, tabelas, controles deslizantes e botões, facilitando a visualização e a análise de dados em tempo real. A simplicidade de uso e a integração perfeita com o ecossistema Python fazem do Streamlit uma ferramenta poderosa para prototipagem e partilha de análises de dados.
- Simpy: O Simpy é uma biblioteca em Python para a simulação de eventos discretos. É frequentemente utilizada para modelar e simular processos complexos, como filas de espera, redes de computadores, sistemas logísticos e outras aplicações onde eventos ocorrem em intervalos discretos de tempo. O Simpy fornece um ambiente

flexível para criar e manipular processos simultâneos, permitindo aos utilizadores definir eventos, recursos e interações intuitivamente. É uma ferramenta valiosa em pesquisas operacionais, engenharia de sistemas e outros campos que necessitam de simulação para análise e otimização.

- Python: O Python é uma linguagem de programação de alto nível, interpretada e de propósito geral, conhecida pela sua sintaxe clara e legível. Desenvolvida por Guido van Rossum e lançada pela primeira vez em 1991, o Python é amplamente utilizado em diversas áreas, incluindo desenvolvimento web, ciência de dados, inteligência artificial, automação, e mais. A sua vasta biblioteca padrão e a abundância de bibliotecas de terceiros permitem aos programadores desenvolver soluções eficientes e rápidas para uma ampla gama de problemas. A comunidade ativa e o suporte extenso tornam o Python uma das linguagens de programação mais populares e acessíveis atualmente.

3.4 Validação do modelo

A validação do modelo foi realizada para assegurar que os resultados da simulação refletissem adequadamente o funcionamento esperado do hospital. Utilizamos uma interface desenvolvida especificamente para analisar os resultados obtidos e compará-los com as expectativas baseadas nos recursos disponíveis antes do início da simulação.

Inicialmente, configuramos os parâmetros do modelo conforme os recursos reais do hospital, como o número de rececionistas, salas de consulta, salas de exames, médicos, enfermeiros, e outras capacidades operacionais. Em seguida, executamos múltiplas simulações para avaliar diferentes cenários e condições operacionais.

Durante a análise dos resultados, foi possível observar como a alocação de recursos afetou diretamente os tempos de espera dos pacientes, a utilização das instalações hospitalares e a eficiência geral do fluxo de trabalho. Por exemplo, cenários com um número maior de médicos e enfermeiros resultaram em tempos de consulta reduzidos e uma capacidade maior de atendimento, enquanto uma falta de salas de cirurgia disponíveis impacta negativamente os tempos de espera para cirurgias urgentes.

Em resumo, a validação do modelo confirmou a sua capacidade de simular com precisão o comportamento operacional do hospital sob diferentes condições. Os insights obtidos são fundamentais para a otimização contínua dos processos hospitalares e para o desenvolvimento de estratégias que visem melhorar a eficiência e a qualidade do atendimento prestado aos pacientes.

Capítulo 4

Conclusão

Neste capítulo, exploramos a simulação detalhada de um hospital e os seus recursos através do uso da biblioteca SimPy. O objetivo principal foi modelar o fluxo de pacientes dentro do hospital, desde a chegada até a alta, considerando a dinâmica dos recursos disponíveis, tais como rececionistas, salas de consulta, salas de exames, médicos, enfermeiros, salas de cirurgia, entre outros.

4.1 Aspetos principais abordados

Ao longo deste trabalho, foi possível atingir os seguintes pontos:

- **Modelagem do Hospital:** Desenvolvemos um modelo conceitual que reflete as características e estrutura do hospital, incluindo a definição de recursos e processos fundamentais.
- **Implementação da Simulação:** Utilizamos a biblioteca SimPy para implementar o modelo de simulação, definindo eventos, processos e recursos conforme necessário para replicar o funcionamento realista do hospital.
- **Validação e Verificação:** Realizamos um processo de validação para assegurar que a simulação produza resultados precisos e consistentes. Isso incluiu comparações com dados reais sempre que possível e análise estatística dos resultados gerados.
- **Resultados Observados:** Os resultados da simulação forneceram insights valiosos sobre o desempenho do hospital em diferentes cenários. Observamos tempos de espera médios, utilização de recursos críticos e o impacto das prioridades dos pacientes na eficiência operacional.

4.2 Contribuições e Implicações

A simulação demonstrou a sua utilidade como uma ferramenta estratégica para a gestão hospitalar. As informações geradas podem ser utilizadas para otimizar o planeamento de recursos, melhorar os tempos de espera dos pacientes e aumentar a capacidade de resposta a emergências.

4.3 Limitações e Trabalho futuro

Embora a simulação tenha sido eficaz na modelagem do hospital, algumas simplificações foram necessárias devido à complexidade do sistema real. Para futuros trabalhos, recomenda-se explorar:

- **Especificação de Subprocessos:** Incorporar mais detalhes em subprocessos específicos, como protocolos clínicos, fluxos de trabalho específicos de unidades médicas e variabilidades no tempo de atendimento.
- **Integração de Dados Reais:** Expandir a validação da simulação com dados operacionais reais do hospital para uma validação mais robusta.
- **Análise de Sensibilidade:** Realizar análises mais profundas de sensibilidade para identificar os principais drivers de desempenho e eficiência do hospital.