КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра прикладної статистики

Кваліфікаційна робота
на здобуття ступеня бакалавра

за спеціальністю 124 Системний аналіз

на тему:
МОДЕЛЮВАННЯ МЕДИЧНОГО СОРТУВАННЯ

Виконав студент 4-го курсу
Царук Роман Михайлович

(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук
Пономарьов Вадим Дмитрович

(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент

(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри прикладної статистики
«05» червня 2023 р.,
протокол №11
Завідувач кафедри
І. В. Розора

(підпис)

Київ – 2023

ЗМІСТ

АНОТАЦІЯ

Дипломна робота складається із вступу, 4 розділів, висновків, джерел (17 найменувань) та додатку. Загальний обсяг роботи складає 49 сторінок без додатка, основний текст роботи викладено на 41 сторінці.

Ключові слова: тріаж, медичне сортування, моделювання, симуляція, UML, Python, програмна реалізація, стохастичний процес, імовірність, статистичний аналіз, оптимальне рішення.

Реферат. В роботі проаналізовано основні підходи до медичного сортування та розроблено програмне рішення для моделювання процесів пункту медичної допомоги. Розглянуті медичні алгоритми та теоретичні засоби для моделювання процесів, пов’язаних зі здоров’ям людини. За допомогою сучасних інструментів спроєктовано та побудовано на мові Python модель, що забезпечує виконання та аналіз результатів симуляційних досліджень для процесів надання невідкладної допомоги в умовах обмежених ресурсів. Зроблено висновки щодо можливостей кінцевої програми.

Keywords: triage, modeling, simulation, UML, Python, software implementation, stochastic process, probability, statistical analysis, optimal solution.

Abstract. The paper analyzes the main approaches to medical triage and develops a software solution for modeling the processes of a medical care center. Medical algorithms and theoretical tools for modeling processes related to human health are considered. The program ensures the implementation and analysis of the results of simulation studies for emergency care processes in conditions of limited resources. Conclusions are drawn about the capabilities of the final program.

ВСТУП

Медичне сортування являє собою процес розподілу й пріоритезації пацієнтів при умовах обмежених ресурсів. Своє основне застосування воно знаходить при масових нещасних випадках, катастрофах, стихійних лихах, а також при військових конфліктах, в умовах війни. Дослідження ефективності алгоритмів сортування відбувається методами комп'ютерної симуляції, позаяк постанова реального експерименту може вартувати життя людей. На сьогоднішній день майже неможливо знайти інструменти, що дозволили б моделювати та досліджувати ефективність таких алгоритмів у загальному випадку. Їх розробка могла б сприяти розв'язанню багатьох прикладних задач й, можливо, суттєво допомогти майбутнім дослідникам цієї сфери.

У цій праці проводиться ознайомлення із медичним сортуванням та методами для моделювання процесів у медичних установах. На основі дослідженої теорії розробляється теоретична модель та наводиться приклад її програмної реалізації. Імплементується ряд інструментів для статистичного аналізу й пошуку оптимальних рішень. Проводиться оцінка отриманої системи, визначаються можливості для її подальшого розвитку.

Метою роботи є розробка гнучкої комп'ютерної моделі медичного сортування з можливістю пошуку оптимальної стратегії.

РОЗДІЛ 1
АЛГОРИТМИ МЕДИЧНОГО СОРТУВАННЯ

1.1 Що таке медичне сортування та тріаж

Коли запит на медичну допомогу значно перевищує наявні ресурси, природно постає питання про розподіл цих ресурсів. При цьому необхідно прийняти той факт, що не всі негайні потреби будуть задоволені вчасно, а деякі можуть не бути задоволені взагалі. Рішення про розподіл обмежених ресурсів охорони здоров'я можуть прийматися на всіх рівнях – від суспільного вибору в рамках національної системи охорони здоров'я (макророзподіл) до рішень окремих осіб про надання негайної невідкладної медичної допомоги та транспортування численних пацієнтів у критичному стані на локальному рівні (мікророзподіл).

У цій науковій роботі мова піде про «тріаж» – термін, який найчастіше використовується для визначення пріоритетності лікування пацієнтів у відділеннях екстреної медичної допомоги (ЕМД), а також під час інцидентів з великою кількістю постраждалих, катастроф і в умовах бойових дій.

Медичне сортування або тріаж – це процес визначення того, які пацієнти потребують найбільш термінового медичного обслуговування на основі тяжкості їхнього стану та інших факторів. Під час медичного сортування лікар або інший медичний працівник оцінює стан пацієнта, дивлячись на його симптоми та інші медичні показники. Це дозволяє визначити, який рівень негайної медичної допомоги необхідний кожному пацієнту.

Хоча ці два терміни тут вжиті як взаємозамінні, між ними існують деякі відмінності. Медичне сортування описує розподіл як медичних, так і немедичних ресурсів і не обов'язково означає, що ресурс, який

розподіляється, є дефіцитним. Напротивагу сортуванню, термін «тріаж» має більш конкретні означення. Він походить від французького слова trier, що означає «сортувати», й історично використовувався в різних сферах людського життя, включаючи поділ земель та сільськогосподарської продукції. Зараз ж слово «тріаж» використовується майже виключно в специфічних контекстах охорони здоров'я, коли мова йде про застосування конкретних алгоритмів розподілу пацієнтів.

Для простоти будемо використовувати слова «тріаж», «медичне сортування» чи просто «сортування» як взаємозамінні.

1.2 Основні етапи та види

Виділяють два основні етапи медичного сортування:

I. Оцінка стану пацієнта. Проводиться швидка оцінка стану пацієнта, щоб визначити, чи потрібна термінова медична допомога. Вимірюються показники вітальних функцій, таких як пульс, артеріальний тиск, дихання, спостереження за поведінкою та інші симптоми.

II. Визначення пріоритетності. На другому етапі пацієнти розподіляються на групи відповідно до ступеня терміновості медичної допомоги, що їм потрібна, та визначається їх пріоритетність для отримання медичної допомоги. При цьому враховуються фактори, такі як тяжкість стану та терміновість надання допомоги, можливість прогресування стану без допомоги, доступність ресурсів та інші критерії.

При пошуку оптимальних стратегій моделюється другий етап, де на розсуд медичного працівника покладене завдання пріоритезації. Перший етап, на якому визначається категорія пацієнта, чітко формалізований в рамках відомих алгоритмів, що будуть розглянуті далі. Однак, щодо першого етапу, то може мати сенс відтворення неточності оцінок, а саме таких явищ, як медичне недосортування та пересортування, коли

відбувається недооцінка чи переоцінка важкості стану пацієнта. Тому має сенс враховувати, що такі характеристики, як тривалість життя пацієнта певної категорії, на практиці можуть відхилятись від очікуваних значень.

Основні види тріажу:

1. Сортування при наданні екстреної медичної допомоги.

2. Сортування в стаціонарі (у відділенні інтенсивної терапії).

3. Сортування при інцидентах з великою кількістю постраждалих.

4. Військове сортування (на полі бою).

5. Сортування при катастрофах чи застосуванні зброї масового ураження (найбільша кількість постраждалих).

Тут на початок списку покладені види, що відносяться до більш звичних для медичних працівників випадків з невеликою кількістю невідкладних пацієнтів й відносно достатнім рівнем ресурсів. Проте алгоритми тріажу були розроблені насамперед для випадків з великою кількістю постраждалих, коли неможливо допомогти всім і питання пріоритезації пацієнтів постає особливо гостро.

Так, перші два види сортування можна назвати плановими чи рутинними. Системи сортування для цих видів, як правило, призначені для визначення пацієнтів з найбільш ургентними (потенційно найсерйознішими) станами, щоб забезпечити їм першочергове лікування, а потім менш ургентні випадки за принципом «хто прийшов першим, той першим і отримав». При рутинному сортуванні є ресурси для лікування кожного пацієнта, хоча ті, хто має менш важкі захворювання або травми, змушені чекати довше.

У випадках масових уражень і катастроф, а також при військових діях, медичне сортування може значно відрізнятись від планового сортування. У таких ситуаціях зазвичай виникає потреба у швидкому та ефективному розподілі медичних ресурсів. При цьому раціонально надавати пріоритет не тільки найважчим хворим, але і пацієнтам, які

можуть бути успішно врятовані з найменшими витратами дефіцитних ресурсів. Також може братись до уваги можливість швидкого відновлення пацієнта, що актуально для військового сортування, коли є потреба швидше вилікувати й повернути солдата у стрій (такий підхід інколи називають зворотнім медичним сортуванням). Саме у випадках з великою кількістю постраждалих добре окреслюється задача максимізації корисності медичних заходів.

1.3 Основні алгоритми тріажу

Simple Triage And Rapid Treatment (START) – це загальновідома система категорій, яку рекомендується використовувати при масових нещасних випадках. Вона дозволяє оцінити стан потерпілого всього за 30 секунд, базуючись на показниках дихання, кровообігу та свідомості. Принцип роботи системи полягає в тому, щоб якомога швидше визначити ступінь тяжкості пошкоджень у кожного постраждалого, а потім розподілити їх на категорії пріоритетів в залежності від ступеня тяжкості. START використовує 4 кольори табличок для позначення пріоритету:

- червоний – негайні,
- жовтий – важливі,
- зелений – можливі та
- чорний – не врятовані й неврятовні (померлі й ті, що не підлягають порятунку).

Опис такого розподілу за кольорами інколи позначається, як METTAG. Для системи START існують різні модифікації, з поміж них окремо можна виділити алгоритм JumpSTART, що створений для оцінки важкості стану дітей.

SALT (Sort, Assess, Lifesaving interventions, Treatment/Transport). Розділяє пацієнтів на аналогічні категорії за кольоровим кодуванням. При цьому категорії характеризують наступним чином:

■ невідкладні, ■ відкладені, ■ мінімальні, ■ очікуючі.

Час оцінки залежить від ступеня важкості та розміру катастрофи або надзвичайної ситуації. Проте, загалом, цей алгоритм повинен допомогти здійснити оцінку пацієнта за 60 секунд або менше.

Canadian Triage and Acuity Scale (CTAS) – це канадська система пріоритетів, яка використовується в екстрених відділеннях лікарень та на догоспітальному етапі для розподілу пацієнтів на категорії пріоритетів. Виділяє 5 категорій за ступенем важкості:

■ рівень 1, сині – реанімація,

■ рівень 2, червоні – невідкладні,

■ рівень 3, жовті – ургентні,

■ рівень 4, зелені – менш ургентні,

■ рівень 5, білі – не ургентні.

Аналогічні позначення використовуються й в алгоритмі JTAS (Japanese Triage and Acuity Scale), а також KTAS (Korean Triage and Acuity Scale).

The Manchester Triage System (MTS). Є найпоширенішою системою сортування у Європі та деяких інших країнах, як наприклад Бразилія. Розподіляє пацієнтів на 5 категорій за невідкладністю й позначає їх кольорами за зменшенням ступеня невідкладності:

■ червоний, ■ оранжевий, ■ жовтий, ■ зелений та ■ синій.

Australasian Triage Scale (ATS) – це клінічний інструмент, який виділяє 5 категорій для визначення максимального часу очікування на медичне обстеження та лікування.

Emergency Severity Index (ESI). Як і попередні два алгоритми теж використовує 5 категорій, однак пронумерованих не тільки за важкістю

проблем зі здоров'ям пацієнта, а й за кількістю ресурсів, які, як очікується, знадобляться для його лікування. Поняття «ресурс» в ESI означає типи складних втручань або діагностичних інструментів, що виходять за рамки фізичного обстеження. Прикладами ресурсів є рентген, аналізи крові, шви, внутрішньовенні або внутрішньом'язові ліки. Рівні ESI пронумеровані від одного до п'яти, причому перші два рівні означають найбільшу терміновість, однак рівні 3, 4 і 5 визначаються не терміновістю, а кількістю ресурсів, які, як очікується, будуть використані.

Sacco Triage Method (STM). Цей метод, який розроблений на основі математичної моделі і є числовим методом сортування, що враховує ресурси. Пораненим виставляються бали від 0 до 12, які визначають одну із 3 категорій, куди потрапить пацієнт.

Військове сортування. Може використовувати аналогічні чи подібні системи оцінювання стану. Основною відмінністю військового сортування є врахування різних факторів, щодо можливості надання допомоги та повернення поранених солдатів на поле бою.

Варто зазначити, що не у всіх випадках використовуються строгі алгоритми. Так, історично такий розподіл впроваджувався суб'єктивно, методом «погляду», коли медик проводить дуже короткий огляд пацієнтів

Важливо звернути увагу, що хоча ці алгоритми описуються як алгоритми пріоритезації, в більшості випадків мова йде лишень про оцінку стану пацієнта і визначення його конкретної категорії. Тобто ці алгоритми працюють здебільшого тільки на першому основному етапі медичного сортування. Завдання визначення пріоритетності покладене на медиків, а згадані системи лишень створюють для цього умови, розділяючи пацієнтів на категорії. Інколи нумерація категорій говорить про очікувану першочерговість надання допомоги (як от алгоритм ESI), однак в більшості випадків кінцеве рішення власне про спосіб відбору має прийматись на місцях.

Мала кількість досліджень, малі вибірки та різноманітність методів оцінки успішності алгоритмів роблять порівняння алгоритмів важким. Рекомендується, щоб різні країни розробляли свою модель сортування під час надзвичайних ситуацій і катастроф, виходячи з місцевих умов і ресурсів.

START є хорошим прикладом алгоритму тріажу (рис. 1). Він є простою для застосування й прийнята в багатьох країнах, зокрема є затвердженим медичним стандартом в Україні. Йому було надано офіційний статус наказом Міністерства охорони здоров'я України від 24 лютого 2022 року № 368.

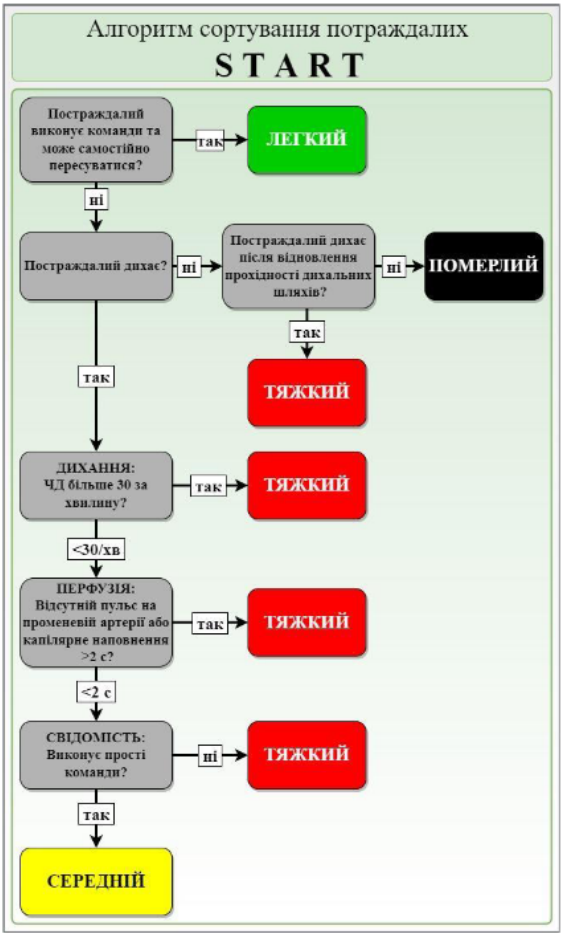

Рис. 1: Алгоритм сортування постраждалих START.

РОЗДІЛ 2
ОГЛЯД ТЕОРЕТИЧНИХ ПІДХОДІВ

2.1 Про моделювання та симуляцію

Модель – це спрощене представлення системи. Моделі намагаються звести світ до фундаментального набору елементів і законів, і на цій основі допомагають краще зрозуміти і передбачити ключові аспекти світу.

Розробка моделі зазвичай передбачає формулювання наукової гіпотези або виявлення певної структури чи динаміки. Часто ця гіпотеза ґрунтується на аналізі закономірностей, виявлених в емпіричних даних. Незалежно від того, чи базується гіпотеза на даних, чи на теорії, емпіричний набір даних повинен бути доступним для перевірки результатів моделі.

Моделі можна поділити на дескриптивні (описові) та нормативні. Дескриптивні моделі відповідають на питання «як це відбувається або як це може розвиватися в майбутньому?». Іншими словами, вони лише пояснюють або передбачають спостережувані факти. Нормативні моделі відповідають на питання «як має бути», пропонують рішення.

За характером причинно-наслідкових зв’язків розрізняють моделі жорстко детерміновані і моделі, що враховують випадковість і невизначеність. Щодо останніх, то треба розрізняти ймовірнісну невизначеність, та невизначеність, для опису якої закони теорії ймовірностей застосовувати не можна. Прикладом другого типу є теорія нечітких множин та нечітка логіка. Детерміновані моделі описують об'єкти або явища, поведінка яких повністю визначається їх початковим станом та вхідними даними. У детермінованих моделях заданий вхід завжди призводить до одного і того ж результату. Прикладами їх є фізичні закони, наприклад, закони Ньютона, які можна використовувати для опису і

прогнозування руху фізичних тіл. Стохастичні (також звані імовірнісними) моделі дають змогу передбачити поведінку об'єкта або явища, якщо вплив невідомих факторів є значним. Вони не можуть передбачити точну поведінку, але допомагають передбачити ймовірність того, що певне значення буде спостерігатися в певний момент часу в межах відомого довірчого інтервалу. Діапазони значень (у вигляді розподілу ймовірностей) використовуються для опису кожної змінної моделі.

За способом вираження фактора часу математичні моделі поділяються на статичні й динамічні. Динамічні моделі описують зміни процесів у часі. При цьому час може змінюватися стрибкоподібно (дискретно) або неперервно.

Класифікація типів математичних моделей також може бути здійснена на основі критеріїв аналітичного та комп'ютерного моделювання (рис. 2).

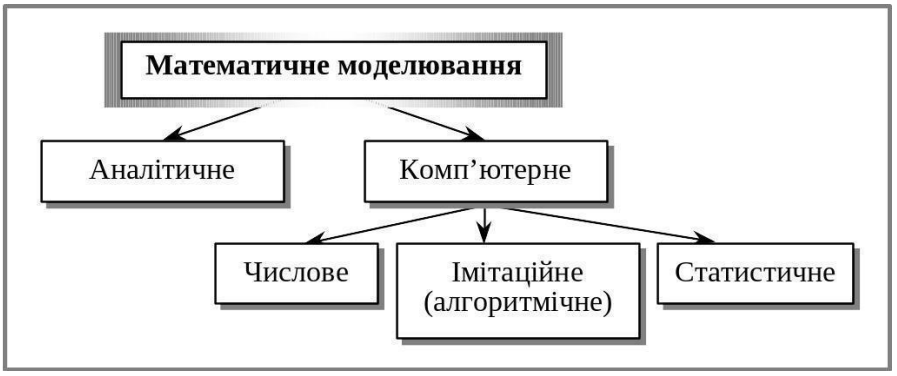

Рис. 2: Аналітичне та комп'ютерне моделювання.

Аналітичне моделювання характеризується тим, що функціональні процеси елементів системи записуються у вигляді деяких математичних співвідношень (алгебраїчних, диференціальних тощо) або логічних умов.

Аналітичне моделювання можна проводити наступними способами:

а) Власне аналітичне чи математичне, коли намагаються отримати залежності в загальному вигляді, вивести формулу;

б) Чисельне, коли виводяться розв’язки, які передбачають проведення певної кількості розрахунків для досягнення потрібної точності;

в) Якісне, що означає знаходження властивостей розв'язку, не маючи явного розв'язку.

Комп'ютерне моделювання характеризується тим, що математична модель системи (не без використання основних співвідношень аналітичного моделювання) представляється у вигляді певного алгоритму і програми, придатної для реалізації на комп'ютері, що дозволяє проводити з нею обчислювальні експерименти. Залежно від математичного інструментарію, що використовується для побудови моделі, і способу організації обчислювальних експериментів, можна виділити три взаємопов'язані види моделювання: чисельне, алгоритмічне (імітаційне) і статистичне.

При чисельному моделюванні комп'ютерні моделі будуються за допомогою обчислювальних математичних методів, а обчислювальні експерименти полягають у чисельному розв'язанні конкретного математичного рівняння при заданих значеннях параметрів і початкових умовах, фактично відбувається реалізація аналітичного чисельного методу.

Алгоритмічне (імітаційне) моделювання (детерміноване або стохастичне) – вид комп'ютерного моделювання, що характеризується комп'ютерною імітацією функціонування досліджуваної складної системи. Воно імітує елементарні явища, що складають процес, зі збереженням їх логіко-смислової структури та порядку протікання в часі.

Статистичне моделювання – вид комп'ютерного моделювання, що дозволяє отримувати статистичні дані про процеси в модельованій системі. Стохастичне імітаційне моделювання також можна віднести і до статистичного моделювання.

Модель можна переконфігурувати та експериментувати з нею; зазвичай це неможливо, занадто дорого або недоцільно робити в системі, яку вона представляє.

Щодо **симуляції**, то за визначенням Роберта Е. Шеннона, симуляція – це процес конструювання моделі реальної системи та проведення експериментів з цією моделлю з метою зрозуміти поведінку системи або відтворити певні стратегії, змінюючи критерії системи. Симуляція є процесом відтворення реальних процесів або систем у віртуальному середовищі з використанням моделей. Вона охоплює запуск моделі й спостереження за її поведінкою та реакцією на різні вхідні дані або умови. Симуляція може бути використана для перевірки або тестування моделі, вивчення впливу різних факторів на систему, навчання, дослідження або тренування людей на взаємодію з певними сценаріями.

Дослідження через симуляцію можна віднести до методів моделювання динамічних систем, що є найбільш актуальним при імітації складних стохастичних систем. Надалі будемо використовувати термін симуляція саме як процес відтворення роботи досліджуваних систем-моделей.

Симуляцію також можна визначити, як дискретно-подієве моделювання. Хоча існують різні підходи та парадигми до дискретно-подієвого моделювання, склалася базова структура, яка використовується більшістю готових інструментів для проведення симуляційних досліджень. Структурні компоненти симуляцій зазвичай включають сутності, дії, події, ресурси, глобальні змінні, генератор випадкових чисел, змінні стану системи та збирачі статистики.

Сутності – об’єкти імітаційної моделі. Сутності мають атрибути – характеристики даної сутності, які є унікальними для цієї сутності.

Дії – це процеси та логіка в симуляції. Події – це умови, які відбуваються в певний момент часу і викликають зміну стану системи.

Сутності, взаємодіючи з діями, створюють події або події впливають на сутності, приводячи їх до дій.

Ресурси – це все, що має обмежену кількість чи ємність. Поширеними прикладами ресурсів є робітники, машини, вузли в комунікаційній мережі тощо.

Глобальна змінна – це змінна, яка доступна для всієї моделі в будь-який час.

Кожна стохастична симуляція потребує генератор випадкових чисел (технічна назва – генератор псевдовипадкових чисел), що використовується для створення невизначеності в симуляції, генерування даних за деякими законами розподілу.

Змінні стану системи вказують на стан, в якому перебуває модель. До них відносять і змінну поточного часу системи.

Збирачі статистики – це частина симуляції, яка збирає статистику про певні стани (наприклад, стан ресурсів або значення глобальних змінних, чи певну статистику продуктивності на основі дій чи атрибутів об'єкта). Існує три різні типи статистики, які збираються: лічильники, підрахунки, постійні в часі та підсумкові статистики. Лічильники дуже прості, вони просто рахують. Постійні в часі статистичні колектори дають зважені в часі значення різних змінних у симуляції. Підсумкові статистики збираються по одному спостереженню за раз без урахування часу між спостереженнями.

2.2 Математичні моделі медичних процесів

У цьому розділі описані дві математичні моделі, що їх часто можна зустріти у дослідженнях медичних процесів. При потребі будуть використані відповідні математичні рішення. Оглянемо ці моделі:

2.2.1 Теорія черг

Одним із популярних теоретичних підходів, що використовується для розв’язання прикладних задач у різних сферах людського життя, зокрема у медицині, є теорія масового обслуговування або теорія черг (Queueing theory).

Систему черги у випадку медичних закладів можна описати як пацієнтів, які певним чином прибувають на обслуговування, очікують на обслуговування, користуються послугою і залишають систему після її отримання.

Процеси або моделі черги описуються за допомогою серії символів і косих рисок A/S/c/K/N/D (Kendall's notation). Букви позначають наступне:

A – arrival process, прибуття пацієнтів. Розподіли, які визначають моменти прибуття нових пацієнтів можуть бути досить різними. Найпопулярніші значення, які набуває змінна A:

- D – детерміноване прибуття чи фіксоване в часі
- M – прибуття, що характеризується марківським процесом (випадкові прибуття що реалізуються в Пуассонівському процесі з постійною інтенсивністю λ). Утворює розподіл Пуассона.
- G – загальний розподіл або деякі правила, про які є мало інформації для визначення форми розподілу.

S – service time distribution, розподіл часу обслуговування пацієнтів. Більшість можливих позначень аналогічні, як і в пункті A. Позначення «M» у цьому випадку відповідатиме експоненціальному розподілу тривалості часу надання допомоги.

c – number of servers, кількість серверів обслуговування (зазвичай 1).

K – number of places in the queue, пропускна здатність системи або скільки пацієнтів може прийняти сервер, ∞ за замовчуванням.

N – calling population, величина популяції з якої прибуває черга, ∞ за замовчуванням. Інколи цей параметр викидається із загального позначення.

D – queue's discipline, порядок обслуговування черги. Можливі значення:

- FIFO/FCFS – First In First Out/First Come First Served, «перший ввійшов – перший вийшов», пацієнти обслуговуються за порядком поступання.
- LIFO/LCFS – Last in First Out/Last Come First Served, «останній ввійшов – перший вийшов», обслуговування в зворотному порядку.
- SIRO – Service In Random Order, випадковий порядок надання допомоги.
- PQ – Priority Queuing, пріоритетне надання допомоги.

Виходячи з цієї нотації, очевидно, що теорія масового обслуговування має можливість працювати зі складними і досить витонченими моделями. Але в більшості випадків прості моделі є адекватними для надання всієї інформації, необхідної для прийняття правильних рішень.

Однією із найпоширеніших моделей є M/M/1/∞/∞/FCFS (останні параметри зазвичай залишають поза позначенням, тому модель зазвичай записується як M/M/1). Як зазначалось вище, M означає марковський процес, який передбачає, що швидкість прибуття або обслуговування відповідає пуассонівському розподілу, а час між прибуттями або час обслуговування відповідає експоненціальному розподілу. Для цієї системи пропускна здатність системи є нескінченною, а порядок обслуговування черги – FCFS (First Come First Served).

Очевидно, що моделювання обслуговування людей, зокрема надання медичної допомоги, не обходиться без визначення черг. Ця робота не буде виключенням – будуть змодельовані гнучкі черги, що можуть бути змінені у залежності до налаштувань.

2.2.2 Моделі марківських процесів

Марковський процес – це випадковий процес без пам'яті, в якому ми беремо послідовність випадкових станів, що задовольняють вимогам властивості Маркова:

$$P(X_2 = s_2 | X_0 = s_0, X_1 = s_1) = P(X_2 = s_2 | X_1 = s_1)$$

Тобто ймовірність будь-якого стану системи в майбутньому обумовлюється тільки її теперішнім станом і не залежить від того, яким чином система набула цього стану.

Марковським процесом або **ланцюгом маркова** називається кортеж <S, P>, де

S – (скінченна) множина станів,

$P_{ss'}$ – матриця ймовірностей переходу в інший стан,

$$P_{ss'} = P\left[S_{t+1} = s' \mid S_t = s\right].$$

Марковський процес з винагородою (Markov Reward Process – MRP)

Марковський процес з винагородою є розширенням оригінального процесу Маркова, але з додаванням винагороди, визначається через

кортеж <S, P, R, γ>, де додаються

R_s – функція винагороди для кожного стану,

γ – коефіцієнт дисконтування, $\gamma \in [0, 1]$. Цей фактор зменшує винагороду, яку ми отримуємо від однієї й тієї ж дії з плином часу.

Марковський процес прийняття рішень (Markov Decision Process – MDP)

Розширення марковського процесу з винагородою, визначається через кортеж <S, A, P, R, γ>. Вводяться наступні зміни:

A – додана скінченна множина дій,

$P_{ass'}$ – в матриці ймовірностей переходу у новий стан враховуються дії,

$$P_{ass'} = P\left[S_{t+1} = s' \mid S_t = s, A_t = a\right].$$

Згадані моделі надають можливість знаходити розв’язки багатьох задач у медицині за допомогою симуляцій, а інколи й аналітичним шляхом. Саме вони були використані як математичний апарат для моделювання відбору пацієнтів при евакуації й надання їм місць у госпіталі в деяких відкритих дослідженнях. Планується, що розроблювана у цьому дослідженні система буде досить складною й в багатьох місцях потенційно залежною від попередніх станів системи, тому інструменти, що використовуються виключно для аналізу моделей марківських процесів реалізовуватись не будуть.

РОЗДІЛ 3
МАТЕМАТИЧНА МОДЕЛЬ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Побудова математичної моделі

Буде змодельована робота пункту невідкладної медичної допомоги, що приймає пацієнтів та виконує з ними такі дії a кількістю A що дорівнює трьом, як маркування, транспортування з сортуванням, лікування – основні процеси (дії) системи. Також називатимемо їх етапи, тому вони проводитимуться послідовно для кожного пацієнта.

Утворена система буде умовно поділена на дві частини: стійка структурна частина або основа системи та перемінна частина або налаштування.

Налаштування складатимуть важливі для симуляції початкові дані та правила у вигляді змінних і функцій, що задаються дослідником.

Стандартний набір налаштувань міститиме наступні елементи:

$n \in N$ – початкова кількість пацієнтів,

$L \in N$ – кількість категорій для маркування (категорії визначатимуться числом),

$Dm_{L \times A}$ – очікувана ймовірність смерті відповідно до категорії та етапу, виражена у формі матриці; під ймовірністю смерті будемо мати на увазі інтенсивність смертності – деяку величину μ_x, для якої величина $\mu_x t$ наближено дорівнює ймовірності померти на інтервалі $(x, x + t)$,

$Dd_{L \times A}$ – очікуване відхилення очікуваної ймовірності смерті відповідно до категорії та етапу, також виражене у формі матриці,

$Prob(l)$ – вектор або функція відносної імовірності отримати пацієнта певної категорії,

$g()$ – функція генерування ймовірності смерті,

$\lambda \in N$ – змінна або функція очікуваної кількості часу, що необхідна для стабілізації (надання допомоги) пацієнта,

$\delta()$ – функція зміни інтенсивності смертності для пацієнта,

$l()$ – функція маркування, визначає категорію пацієнта за доступною інформацією про нього із можливою похибкою,

$s()$ – функція сортування пацієнтів для надання допомоги,

r() – функція визначення результату експерименту,

$\{r\}$ – множина ресурсів, структура яких визначена нижче,

$G()$ – функція, що конструює час та основну динаміку процесів,

$I \in N$ – максимальна тривалість кожного експерименту,

$J \in N$ – кількість повторень експерименту.

Ці та інші налаштування будуть вбудовані у загальну систему.

Опис **основи системи** почнемо з базових структурних одиниць системи:

$r \in \{r\}$ – ресурс системи; кожен елемент цієї множини містить дані про загальну кількість ресурсу R відповідного типу, його відновлюваність чи невідновлюваність, можливість паралельного використання u та функції $\alpha()$, $\beta()$, $\theta()$, що визначають відповідно кількість, що виділяється на відповідний етап із загальної кількості ресурсів та кількість і час використання цього ресурсу для одного пацієнта,

$p \in \{p\}$ – пацієнт; складається із наступних даних: інтенсивність смертності d_a, потрібний час для надання допомоги λ, категорія l, етап a, на якому перебуває пацієнт та змінна стану пацієнта (визначає, відноситься пацієнт до активних, померлих чи врятованих).

Система функціонуватиме завдяки вкладеним циклам. Розглянемо детальніше що відбувається в циклах, починаючи від циклів загального рівня.

1) Перебір налаштувань, якщо їх є декілька. На цьому рівні відбувається ітерація всього алгоритму для різних наборів вхідних даних $k \in \{k\}$.
2) Повторення експерименту для одного налаштування. Оскільки результат стохастичного експерименту не визначений, то відбувається J повторень симуляції з накопиченням результатів від кожного перезапуску.
3) Реалізація плину часу й запланованих процесів. Викликається функція $G()$, що відтворює час $i \in 1, 2 \dots I$ й у потрібні моменти запускає нові або відновлює процеси надання допомоги, а також викликає ймовірнісну функцію, що визначає стан пацієнта й оновлює його інтенсивність смертності d_a за результатом функції $\delta(d_a)$. На цьому рівні визначений достроковий вихід з циклу при досягненні певних умов, наприклад відсутності активних пацієнтів у системі. Результат експерименту визначається через функцію $r(\{p\})$ як деяка метрика множини пацієнтів для моменту часу.
4) Ітерація пацієнтів для виконання процесу a'. На цьому рівні для кожної нової ітерації циклу ініціалізується змінна використаного часу t та визначається множина потрібних ресурсів $\{r_{ai}\}$ за значенням функції $\alpha_{ra}(\{p\}, R)$. Далі для кожного пацієнта $p \in \{p \mid a_p = a'\}$ запускається цикл рівня 5 і при його успішному завершенні змінюється значення етапу a пацієнта, у випадку кінцевого процесу (надання допомоги)

зменшується тривалість лікування пацієнта λ й коли вона дорівнює 0 пацієнт змінює стан і стає врятованим.

5) Витрати ресурсів на певного пацієнта. Для кожного ресурсу $r \in \{r_{ai}\}$ робляться наступні обчислення:

а) якщо $R_{ai} - \beta(p) < 0$, то виконується вихід з циклу (недостатньо потрібного ресурсу),

б) обчислюється $t + \theta_{ra}(p)$ або $t + \theta_{ra}(p)/R_{ai}$ в залежності від змінної u і якщо результат більший за 1, то відбувається вихід з циклу при збереженні стану (вийшов час симуляції),

в) обчислюється

$R_{a,i+1} = R_{ai} - \beta(p)$ (нова кількість певного ресурсу)

$t' = t + \theta_{ra}(p)$ або $t + \theta_{ra}(p)/R_{ai}$ в залежності від змінної u, де t' – нове значення витрат часу.

В кінці отримуємо множину результатів повторень експерименту для кожного налаштування системи. Система не виконуватиме пошук оптимальних рішень, однак для порівняння результатів будуть реалізовані різні інструменти статистичного аналізу.

Створена модель утворює чергу із пацієнтів типу D/M/1/∞/∞/PQ, в припущенні що час лікування матиме розподіл Пуассона. Загалом характеристики цієї черги залежатимуть від початкових налаштувань.

Ймовірність смерті у системі може змінюватись динамічно й час перебування на етапі надання допомоги має визначену тривалість, тому процеси системи не є марківськими.

3.2 Проєктування розробки. Засоби та підходи

Тепер, коли сформульована алгоритмічна модель для імітаційного стохастичного моделювання, можна визначитись із засобами та способами розробки.

Основним **засобом** розробки обрано мову програмування Python, що є сучасною високорівневою мовою і дозволяє лаконічно та швидко створювати код, не відволікаючись на такі технічні аспекти, як перетворення програми на машинний код, менеджмент пам’яті тощо. Python є однією із найпопулярніших мов для вирішення широкого кола задач, зокрема для моделювання у наукових дослідженнях.

Будуть використані такі бібліотеки (готові програмні рішення), як

- numpy – генерація псевдовипадкових чисел, обчислення,
- scipy – великий набір наукових методів, зокрема тестів статистичного аналізу,
- plotly – створення різноманітних графіків та діаграм,
- а також інші, вбудовані бібліотеки.

Ефективне програмування забезпечуватимуть середовища розробки, наприклад Jypiter Notebook, що рекомендується використовувати й при використанні програми.

Безпосередньо перед розробкою рекомендується визначитись із **підходами** до проєктування, що сильно впливає на кінцевий продукт. В загальному випадку хороше проєктування має деякі ключові властивості, які сприяють ефективності, якості та розширюваності системи. Ось декілька важливих властивостей або принципів хорошого проєктування:

- Модульність: Система повинна бути розбита на незалежні модулі або компоненти, які можуть працювати окремо із заданим інтерфейсом. Це сприяє зручності управління системою, розподілу роботи між розробниками та полегшує тестування та підтримку.

- Зрозумілість: Проєктування повинно бути зрозумілим для розробників, архітекторів та інших зацікавлених сторін. Чітка структура, документація та коментарі допомагають зрозуміти логіку та функціональність системи.
- Розширюваність: Система повинна бути здатною до легкого розширення або модифікації без необхідності внесення великих змін. Гнучка архітектура, використання добре визначених інтерфейсів та розділення обов'язків сприяють розширюваності.
- Повторне використання: Проєктування повинно сприяти повторному використанню компонентів або модулів. Це зменшує зусилля та час, необхідні для розробки нових систем або функцій, і сприяє створенню більш ефективних та стабільних рішень.
- Ефективність: Проєктування повинно забезпечувати оптимальне використання ресурсів, таких як час виконання, пам'ять, мережеві з'єднання тощо.

Реалізація сучасних моделей методами програмування лишень підкреслила важливість дотримання принципів хорошого проєктування. В контексті цього були вироблені методики для ефективної розробки комп’ютерних програм, однією з яких є методологія розробки, що керується функціональністю (Feature Driven Development, FDD).

У традиційному підході до розробки програмного забезпечення за моделлю водоспаду весь проєкт ділиться на кілька етапів: збір вимог користувачів, проєктування та документування, розробка, тестування та розгортання. У цьому підході передбачається, що кожен етап є завершеним до початку наступного етапу. Одним з основних недоліків цього підходу є те, що помилки в дизайні можуть бути виявлені лишень при застосуванні

програми. В цей час проєкт майже завершений і виправлення помилок часто коштує дорого. Гнучкі методи намагаються уникнути цієї слабкості «водоспаду», виконуючи ітеративну розробку. Це допомагає виявити помилки на ранніх стадіях розробки.

Для створення схеми програми заведено використовувати такий інструмент, як UML. Уніфікована мова моделювання (Unified Modeling Language, UML) – це графічна мова для візуалізації, специфікації, конструювання та документування артефактів складної системи, що стала стандартом де-факто для створення об'єктноорієнтованого програмного забезпечення. Специфікація UML 2 визначає 13 основних типів діаграм, розділених на два великих класи: структурні та поведінкові діаграми. Використаємо її для проєктування та представлення розробленої системи.

3.3 Опис структури програми та її загального функціонала

На основі сучасних підходів до моделювання була розроблена і програмно реалізована модель для симуляції роботи відділення невідкладної медичної допомоги із медичним сортуванням (її код міститься у Додатку). Слідуючи підходам Feature Driven Development була створена гнучка загальна структура і функції, що ускладнюються і доробляються в міру розробки.

Модель реалізовано в рамках об’єктоорієнтованого програмування (ООП), з використанням класів — інструментів розділення програмної логіки на шматки, що відповідають за окремі сутності моделі, що утворюють певні об’єкти зі своїми атрибутами та методами. Саме тому візуалізацію даної моделі структурною діаграмою UML можна провести за допомогою діаграми класів (рис. 3):

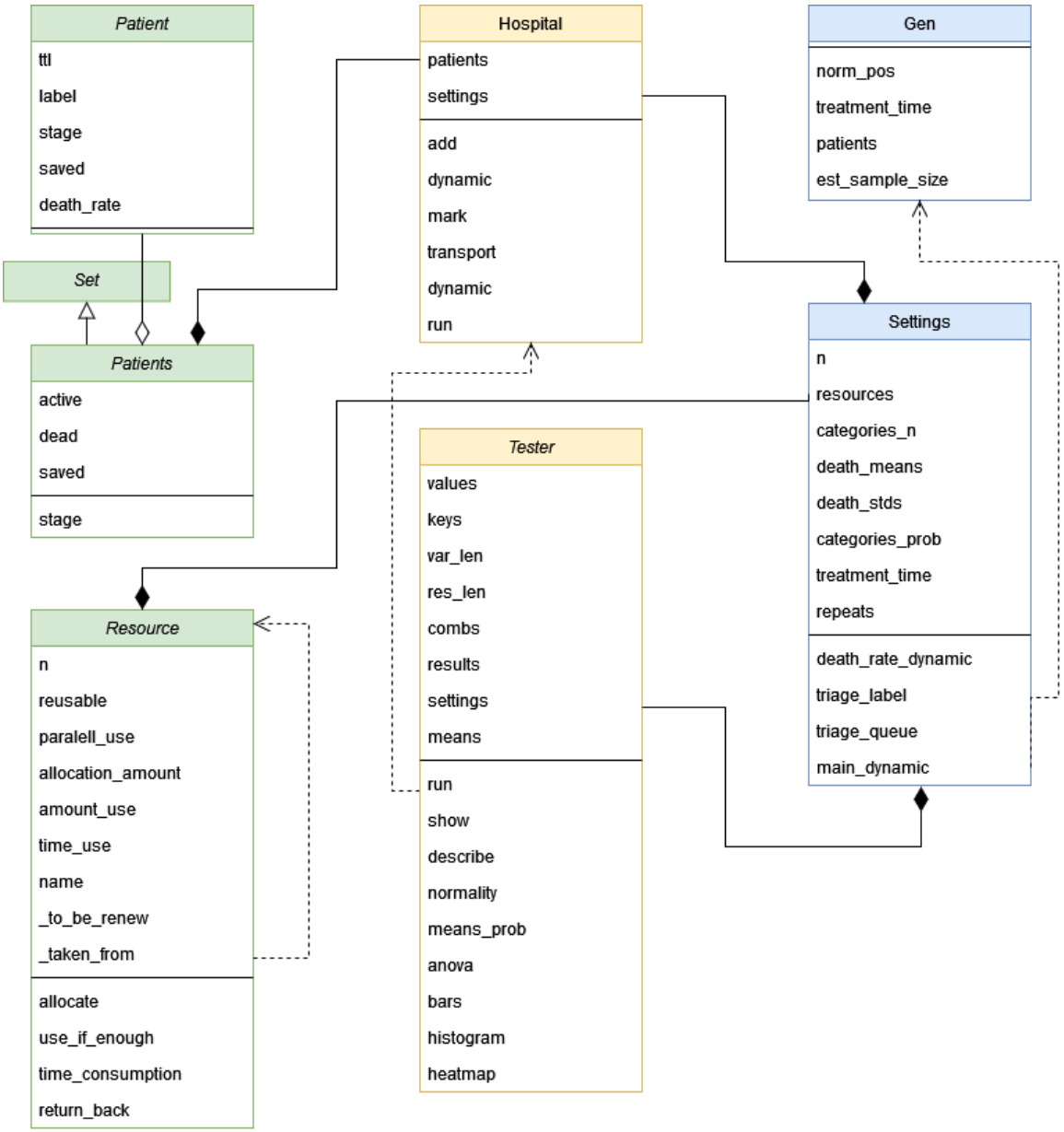

Рис. 3: Діаграма класів програми.

Як можна бачити програма складається із семи класів. Для кожного класу зазначена його назва, список атрибутів (даних чи параметрів класу), список методів класу (функцій-підпрограм що можуть бути виконані звертаючись до класу прямо чи через згенеровані ним об’єкти) та показані зв’язки між класами. Типи даних вказані у коді у вигляді підказок (type hints).

Зеленим кольором виділені класи, що є конструкторами унікальних об’єктів, таких як предмети, жовтим — рольові класи, що містять ряд методів для виконання функцій над об’єктами. Голубим кольором позначені описові класи, що зберігають налаштування. Вони не містять конструкторів об’єктів класу і виконують роль простих контейнерів змінних та функцій, що можуть бути легко відредаговані.

Програма не має інтерфейсу користувача, тобто досліднику, який нею користується потрібно взаємодіяти з кодом програми. Це вимагає деяких знань, однак дає свободу — робота ведеться не лишень з деякими змінними, а з багатьма складними об’єктами; за допомогою програмних функцій реалізовуються довільні функціонально-ймовірнісні залежності.

Для базової взаємодії із програмою потрібно вміти редагувати вміст класу Settings та знати основні методи класів Tester та Hospital. Приклади використання програми будуть надані в наступному розділі.

Схарактеризуємо побудовані класи:

Patient. Простий, але важливий для розуміння можливостей програми клас, що зберігає дані про пацієнта. Його атрибути містять такі характеристики:

- _death_rates – кортеж зі значень ймовірності смерті пацієнта для кожної стадії, умовно прихований атрибут,
- ttl – час до стабілізації чи одужання,
- label – маркування (номер категорії),
- stage – номер етапу пацієнта: 0 — новий, 1 — оглянутий, промаркований і очікує, 2 — лікується,
- saved – атрибут зі значенням за замовчуванням None, змінюється на True або False відповідно до того, чи був пацієнт стабілізований, чи помер,
- death_rate – ймовірність смерті у залежності від того, на якому етапі знаходиться пацієнт. Отримання значення цього

атрибуту реалізується через спеціальну функцію — «ґетер» — що використовує атрибути _death_rates та stage, тому фактично цей атрибут є і методом класу.

Patients. Множина пацієнтів з деякими методами, що фільтрують пацієнтів за певними ознаками.

Resource. Формує ресурс певного типу, точніше деяку кількість певного ресурсу.

Hospital. Пункт надання допомоги, що містить множину пацієнтів, набір налаштувань та методи або процеси, що проводяться над пацієнтами (маркування, транспортування, надання допомоги), а також функцію постійних процесів (смертність).

Tester. Редагує налаштування і запускає каскад симуляційних дій. Містить ряд методів для статистичного аналізу результатів відповідно до наборів параметрів налаштувань (можна порівняти результати для різних вхідних параметрів):

- show – виводить отримані значення
- describe – середнє значення та середньоквадратичне відхилення результатів повторних симуляцій для кожного набору параметрів,
- normality – перевірка на нормальність розподілу результатів критеріями Д’Агостіно-Пірсона та Шапіро-Вілка,
- means_prob – ймовірності середніх значить бути у межах заданого відхилення у припущенні, що результати розподілені нормально,
- anova – дисперсійний аналіз результатів, попарне порівняння,
- bars – стовпчикова діаграма,
- histogram – гістограми розподілів,
- heatmap – теплова карта при дослідженні комбінацій однієї пари параметрів.

Gen. Генератори розподілів та елементів із використанням статистичних функцій. Ці функції редагуються у залежності від статистичних припущень.

Settings. Загальний набір параметрів і функцій. Основний клас, куди потрібно вносити зміни при дослідженні різних конфігурацій. Містить дані, що використовуються у інших класах через передачу копії класу.

Дослідимо зв’язки між утвореними класами, розглядаючи їх за типами. На рисунку були зазначені зв’язки наступних типів:

- Наслідування. Незафарбована стрілка позначає, що клас Patients наслідує функції у базового класу Set (множина). Це дозволяє поводитись з елементами Patients, як з множинами, виконуючи з нами функції додавання чи видалення елементів.
- Агрегація. Цей зв’язок реалізований також лишень між класами Patient та Patients. Він означає, що елементи (об’єкти) першого агрегуються в елементах другого. Легко зрозуміти, що ці класи моделюють пацієнта та множину пацієнтів відповідно. При видаленні об’єкта множини видаляються також всі об’єкти що вона у собі містила.
- Композиція. Утворена модель містить декілька композицій. Так, атрибут settings класу Tester і аналогічний атрибут класу Hospital посилаються на копії класу Settings. Такий самий зв’язок між Hospital та Patients. У випадку композиції між Settings та Resource присутній композиційний зв’язок типу «один до багатьох» — об’єкт Settings може мати будь-яку кількість об’єктів Resources. Композиції позначаються лінією із зафарбованим робом на кінці.
- Залежність. Пунктирними лініями позначені деякі залежності, що говорять про те, що для деяких методів одного класу потрібний доступ до іншого класу. При потребі залежності можуть бути нівельовані прямою передачею копій потрібних класів чи методів.

Рефлексивною залежністю такого типу позначений і клас Resource, тому що його метод allocate продукує нові об’єкти цього ж класу. Це створено для зручної взаємодії із ресурсами моделі — ресурси можна розділити, частково використати, а потім повернути їх до набору основних об’єктів класу Resource, доступ до яких здійснюється через атрибут resources в Hospital.

Класи мають й інші, не показані на рисунку 3 атрибути та методи, які виконують допоміжні функції та не передбачені для доступу ззовні класу (інкапсуляція). Їх назва починається із нижнього підкреслення.

Основні функції симуляції працюють завдяки викликам на різних рівнях вкладених циклів. Загалом поведінку програми можна пояснити на діаграмі основних циклів, спрощено зображеній на рис. 4:

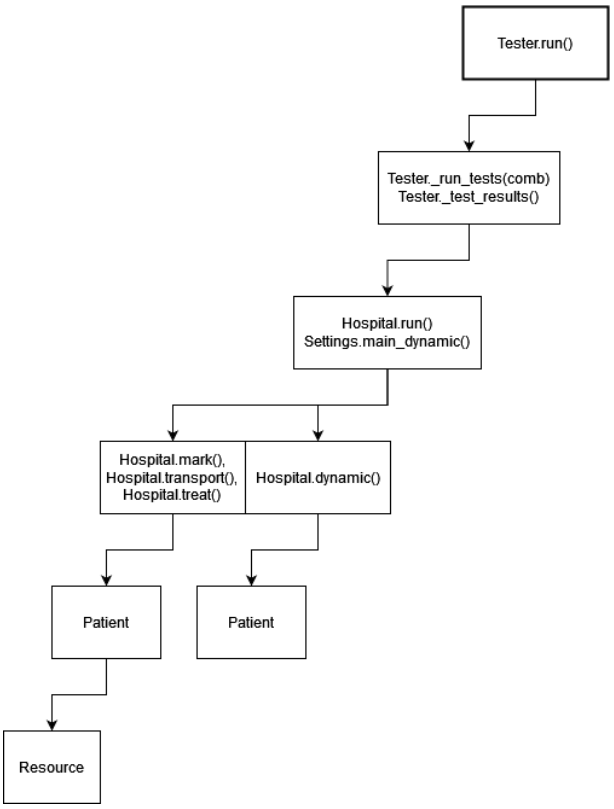

Рис. 4: Представлення зв’язків між основними циклами.

Отже, програма функціонує наступним чином. Після утворення об'єкта типу (класу) Tester із необхідними налаштуваннями, можна викликати його метод run, що запускає процеси виконання симуляції. На цьому рівні працює перший цикл, під час роботи якого завантажуються різні комбінації налаштувань, задані дослідником при утворенні об'єкта класу Tester.

На кожному кроці першого циклу запускається другий цикл, кількість ітерацій якого визначена через змінну repeats, що означає кількість повторень експерименту для одного набору налаштувань. В ньому відбувається ініціалізація (утворення) об’єкту типу Hospital і запускається симуляція через метод Hospital.run() (метод run для об’єктів класу Hospital, без передачі додаткових аргументів — порожні дужки), що запускає Settings.main_dynamic(). Остання функція викликає цикл третього рівня, що відповідає за моменти часу симуляції.

Час є дискретним і просто означає ітерацію даного циклу, його цілочислове значення зберігається у змінній _time в актуальній копії класу Settings. Це значення може бути корисним, якщо потрібно знати як довго система містила пацієнтів або для реалізації функцій, що залежні від значення часу (однак це додаткова залежність, яких слід уникати). Кожної ітерації в цьому «часовому» циклі відбуваються виклики mark, transport, treat, а також dynamic об’єкту типу Hospital, що означають процеси маркування, транспортування, лікування пацієнтів та відтворення смертності. Ці функції запускають деякі підрахування для кожного пацієнта, тому тут працює новий цикл — перебір пацієнтів.

Останнім рівнем у цій функціональній структурі із явно заданих вкладених циклів симуляції є перебір ресурсів для кожного пацієнта. На цьому етапі реалізується використання заданих ресурсів. Обов’язковим ресурсом можна вважати й час, заданий окремо.

Якщо для деякого пацієнта не вистачає потрібного ресурсу, робота процесу зупиняється, а залишки ресурсів повертаються до загального набору. Якщо ж під час виконання процесу, сума використаного часу має перевищити одиницю, то виконання симуляції цього процесу призупиняється — ітерації будуть продовжені при новому часовому циклі. Цей важливий момент робить симуляцію асинхронною, шляхом введення конкурентності між процесами. Таким чином імітується паралельність виконання задач — якийсь процес може тривати більше чим одна ітерація часового циклу, однак це не буде заважати іншим запланованим процесам початись у потрібний момент.

В програмі присутнє логування, що допомагає відстежувати основні процеси. Функція логування що виводить повідомлення на екран та деякі технічні змінні розташовані в глобальному полі програми, тобто поза класами. Підраховується також час роботи програми.

Методи, що виконують складні статистичні розрахунки чи генерують діаграми займають всього декілька рядків коду, не дивлячись потужну функціональність, що в них закладена. Це можливо завдяки зручним готовим рішенням — бібліотекам, що імпортуються на початку програми й відкривають доступ до своїх методів. Для мови Python розроблено багато бібліотек, що неабияк полегшує роботу розробникам і робить мову багатофункціональною і доступною для роботи в різних сферах.

3.4 Уточнення моделі на основі емпіричних даних

Створена модель є майже завершеною структурно, однак запрограмовані функції реалізовані не остаточно. Реалізація цих функцій передбачає отримання відповідей на ряд запитань, які стосуються безпосередньо предмету дослідження.

Необхідно визначити характер, базові параметри й типи розподілів для ймовірнісних функцій, обрати базові налаштування та пересвідчитись, що наявна структура здатна стати основою для реалізації потрібних функціональних зв’язків і взаємодій.

При цьому потрібно опиратись на статистичні набори й наукові дослідження. У протилежному випадку навіть не дивлячись на деталізованість моделі результати будуть нереалістичними та відірваними від реальності.

На жаль, доступних баз даних із зібраною статистикою, що стосується медичного сортування є мало, а ті що є фокусуються на виключно медичних аспектах, збираючи інформацію про симптоми й лабораторні показники, а не про розподіл ресурсів, час очікування й надання допомоги, пріоритети, та інші дані, що є чутливою для медичних закладів інформацією. Та все-таки деяка інформація доступна.

Дослідження під назвою «Triage accuracy and causes of mistriage using the Korean Triage and Acuity Scale» проведено зі збором даних про 1267 випадків пацієнтів, що прибули для надання невідкладної медичної допомоги в Кореї. Ці дані допоможуть визначитись із деякими функціями. Дане дослідження підраховує випадки неточного початкового тріажу. Так, помилкове встановлення категорії визначено для близько 15% пацієнтів. При цьому з них у 70% випадків відбулась недооцінка важкості стану пацієнта, а у 30% відповідно переоцінка. Використаємо ці дані для розробки тіла додаткової функції _triage_inacuracy в налаштуваннях.

В дослідженні було використано алгоритм KTAS. Розглянемо розподіл категорій для кожного пацієнта (рис 5). Ця інформація може бути використана для визначення ймовірності отримати пацієнта певної категорії.

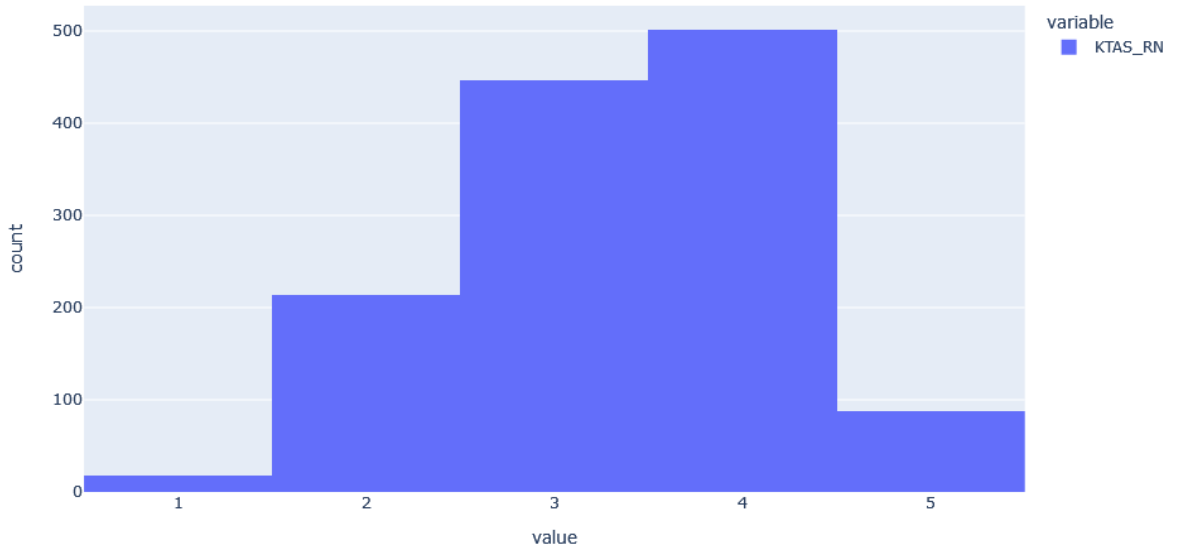

Рис. 5: Розподіл категорій важкості стану пацієнта.

Гістограма тривалості (у хвилинах) перебування пацієнтів на станції невідкладної медичної допомоги має розподіл Пуассона (рис. 6). За таким розподілом буде генеруватись змінна ttl, що відповідає за потрібний час надання допомоги пацієнту.

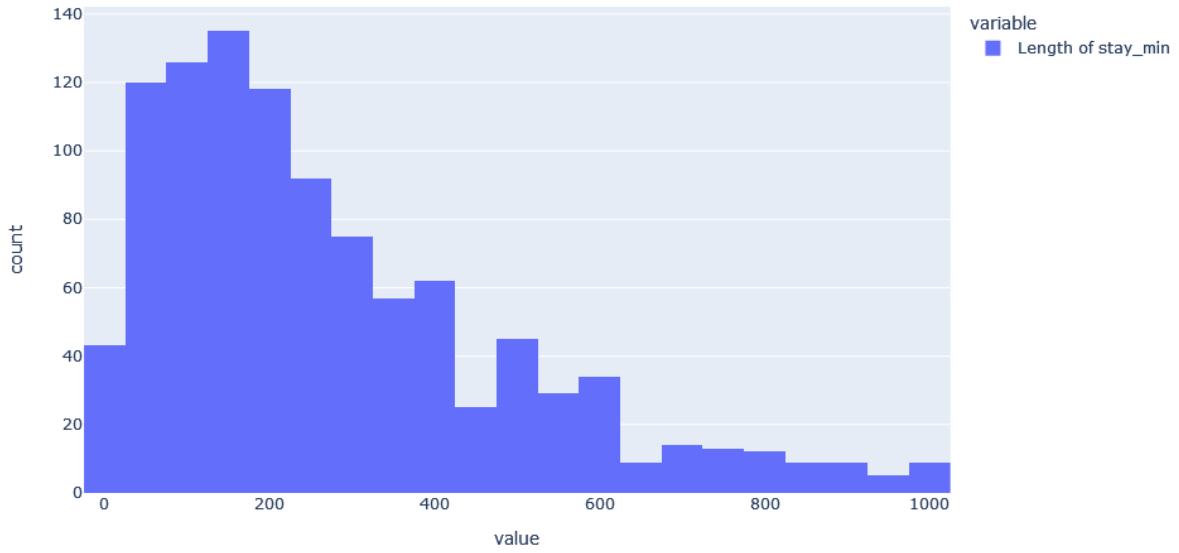

Рис. 6: Розподіл категорій важкості стану пацієнта.

Середня тривалість маркування складає понад п'ять хвилин і має слабку, але статистично надійну кореляцію із номером маркування пацієнта, що може бути використане для функції time_use ресурсу «персонал» для процесу «mark» (медичні працівники представлені у моделі як відновний ресурс). Кореляція між номером категорії пацієнта та тривалістю перебування в лікарні відсутня.

Рівні та розподіли смертності не можна визначити із даних цього дослідження. Скористаємось результатами інших наукових робіт. Під смертністю будемо розуміти функцію інтенсивності смертності, яка для програми з дискретним часом є кусково-неперервною. Як видно на гістограмі (рис. 7), кількість випадків смертей за часом після прибуття до пункту медичної допомоги очікувано також має Пуассонівський розподіл, що тут може бути добре наближений функцією із від'ємним логарифмом.

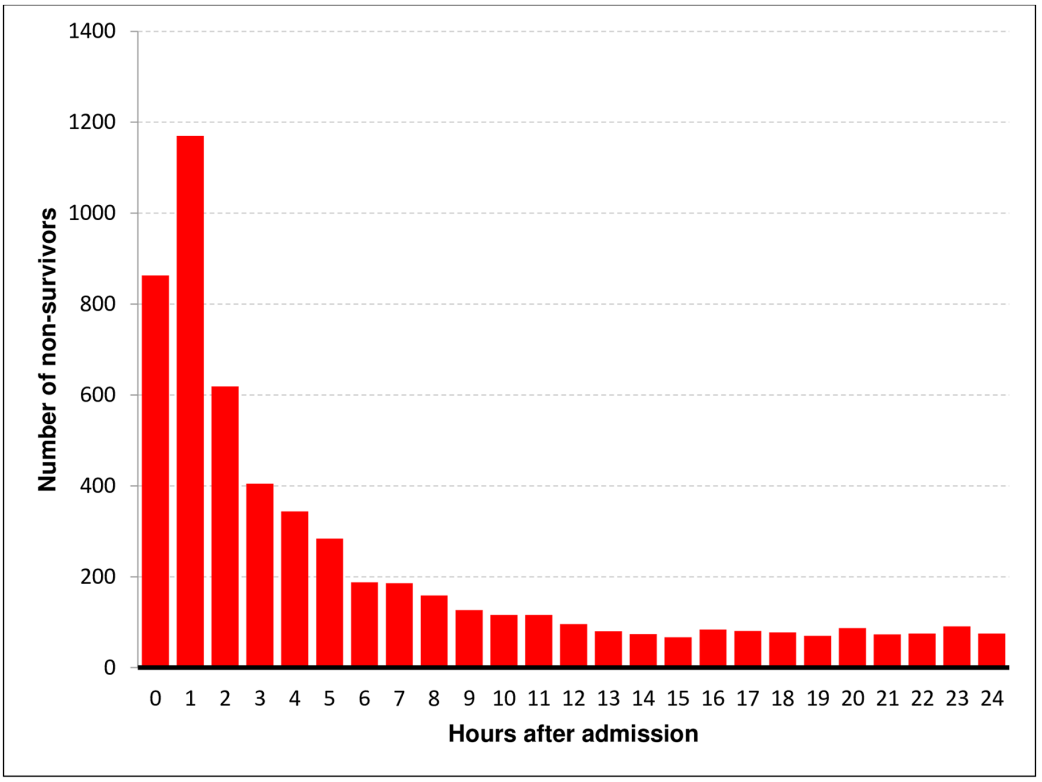

Рис. 7: Розподіл категорій важкості стану пацієнта.

Можна бачити, що виражати ймовірність смерті через константу було б надто недоречно, особливо в перші години прибуття пацієнта. Саме тому існує функція death_rate_dynamic, що покликана відобразити зміни для інтенсивності смертності. В ній реалізуємо спадання рівня смертності, щоб результат був схожий до емпіричного.

Попередні значення смертності та очікуваного часу надання допомоги визначимо приблизно, базуючись на деяких даних із медичних досліджень. При цьому необхідно вказати, яке значення часу обрано за одиницю в програмі (до прикладу, це буде година).

Кількість повторень тесту за замовчуванням покладемо рівними тридцяти, що є рекомендованим розміром вибірки для дослідження центральних моментів у багатьох випадках. Клас Gen містить допоміжну функцію est_sample_size, що приймає значення стандартного відхилення, відхилення, що задає інтервал, та ймовірність та повертає два значення, що є певною оцінкою потрібної кількості повторів, щоб середнє значення вибірки потрапляло у вказаний інтервал із заданою імовірністю. Перше число визначається на основі Центральної граничної теореми, а друге на основі нерівності Чебишева, що на відміну від першого випадку добре працює для значно ширшої родини розподілів, навіть коли вони суттєво відмінні від нормального.

РОЗДІЛ 4
ПРИКЛАДИ ВИКОРИСТАННЯ, ОЦІНКА І ПОДАЛЬШИЙ РОЗВИТОК

4.1 Приклади використання

Перед запуском симуляцій необхідно провести ряд налаштувань системи. Припустимо, що дослідник згідний із базовими налаштуваннями функцій, що містяться в класах Gen та Settings.

Будемо моделювати ситуації, коли до медичного пункту одноразово прибуває певна кількість пацієнтів у важкому стані (3 категорії). Далі пацієнти маркуються та переводяться для надання допомоги. Кінцевий результат буде оцінюватись як відношення кількості стабілізованих пацієнтів до кількості всіх пацієнтів на той момент, коли активних пацієнтів не залишиться у системі (кожному, хто вижив, надана допомога). Маркування буде здійснюватись відповідно до рівня смертності пацієнта, враховуючи випадки неточного тріажу. Сортування пацієнтів проводитиметься за їх категорією — від найбільшого до найменшого номера, тобто найневідкладніші пацієнти. Єдиний ресурс — медичний персонал. Пацієнти генеруються відповідно до частоти появи тієї чи іншої категорії. Кожна категорія має свій очікуваний рівень смертності для кожного етапу та стандартне відхилення для кожного рівня смертності (що генерується з використанням нормального розподілу). Така базова модель уже відтворена у налаштуваннях, тому можна переходити до уточнення базових параметрів.

Дослідник оцінює скільки пацієнтів в тих чи інших станах має бути в його моделі. Маючи певні емпіричні дані, він може знайти наближені значення параметрів смертності для його моделі. При цьому треба обережно переводити середні рівні смертності для різних часових

інтервалів, враховуючи їх динамічність у даній моделі. Так, знаючи, що очікувана ймовірність померти за першу годину пацієнта в певному стані складає 0.5 (50%), вказуючи параметри для 30 хвилинних проміжків, можна задати імовірність для першого інтервалу рівній 0.25, однак це викличе неточність, бо вже у другому інтервалі (на новій часовій ітерації) ця імовірність зменшиться. Більш точне значення можна підібрати, якщо враховувати зміни, що реалізуються через death_rate_dynamic.

Завершивши налаштування базової моделі, можна створити об’єкт класу Tester, передавши параметром клас налаштувань, викликати його метод run, а тоді вивести результати за допомогою методу show. Якщо все працює і середній результат тестувань підраховано, можна переходити власне до тестування різних наборів налаштувань параметрів.

Нехай розподіл персоналу (загальна кількість дорівнює 20) відбувається із використанням певних параметрів що позначають яка максимальна частка цього ресурсу відповідно до доступної кількості буде передана для того чи іншого процесу. Нехай параметр «a» позначає частку персоналу яка буде обрана для першого процесу. Перевіримо, які результати дасть симуляція при різних значеннях «a». Для цього при ініціалізації об'єкта типу Tester, крім Settings, передамо туди аргумент «a» разом зі списком значень, яких він має набути, та запустимо симуляцію (це відтворено в кінці Додатка).

Після цього можна відобразити результати роботи у вигляді стовпчикової діаграми, здійснивши виклик tester.bars() (рис. 8):

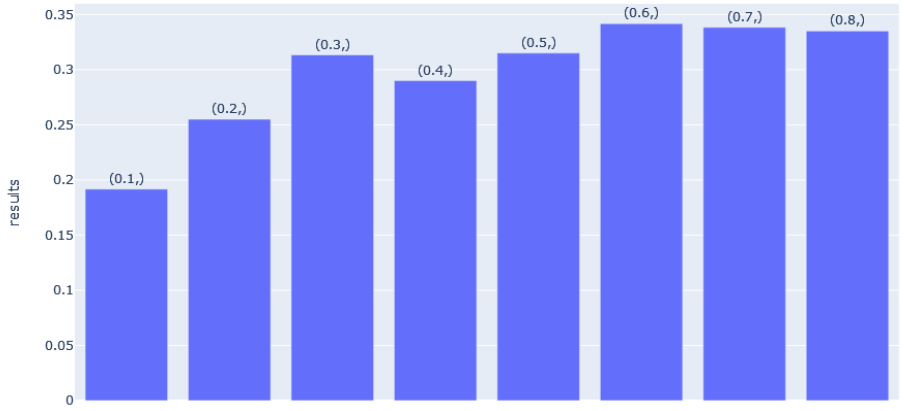

Рис. 8. Стовпчикова діаграма результатів для розміру вибірки за замовчуванням.

Пересвідчимось, що параметр впливає на результат, збільшивши кількість повторів (змінна repeats) в десять разів (рис. 9):

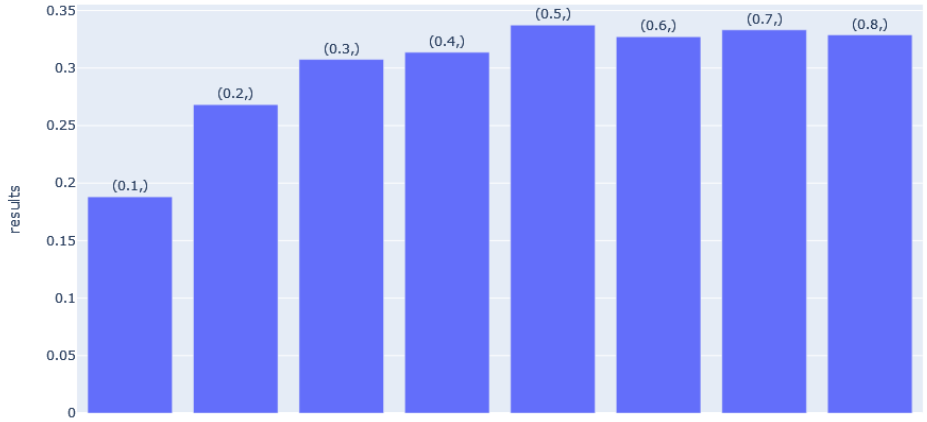

Рис. 9. Стовпчикова діаграма результатів при більшій вибірці.

Стає зрозуміло, що для досягнення оптимальних результатів даний коефіцієнт має бути приблизно рівним 0.5 або більше. Але що буде, якщо початкова кількість пацієнтів буде іншою? Дослідимо, як змінюються

результати в залежності від значень даного коефіцієнта та кількості пацієнтів одночасно, за допомогою інструменту «теплова карта» (рис. 10).

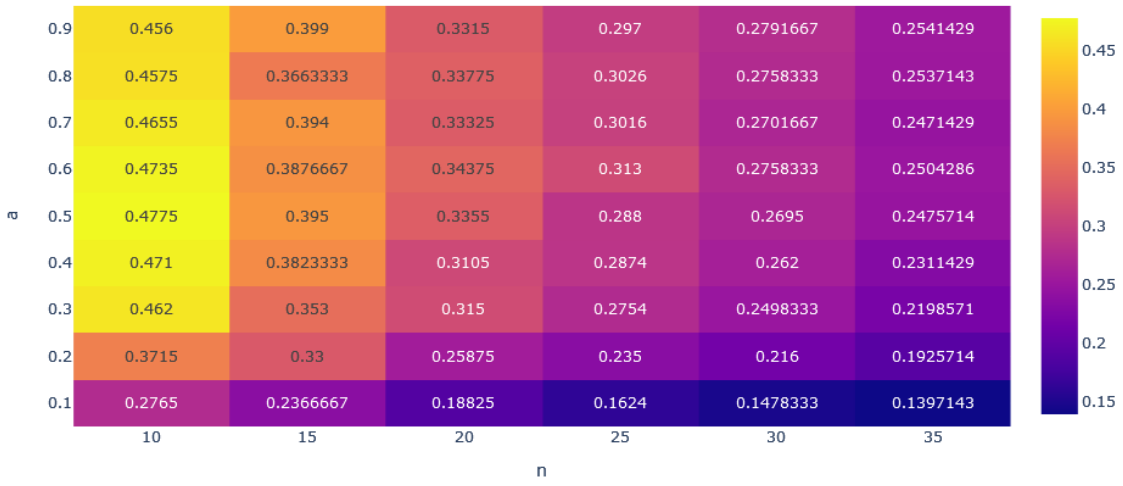

Рис. 10. Приклад теплової карти.

Тепер можна побачити, що найкращим варіантом досліджуваного параметра при кількості пацієнтів від 10 до 40 орієнтовно буде значення 0.6.

Нехай було досліджено що друга категорія пацієнтів має високий ризик смерті на другому етапі й пропонується надати їм пріоритет перед третьою категорією. Стовпчикові діаграми та гістограми (рис. 11) можуть показати деяку різницю розподілів вибірок.

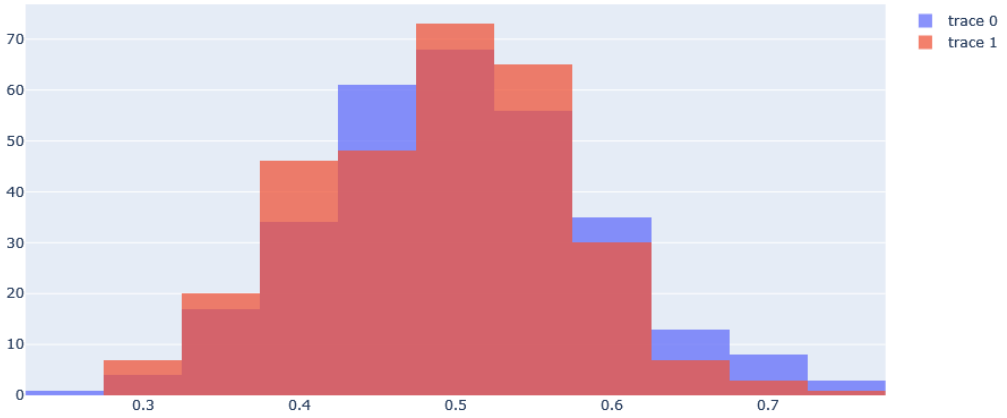

Рис. 10. Приклад гістограми.

Однак викликавши метод anova визначимо, що ця різниця не суттєва. Збільшивши розмір вибірки, пересвідчуємось, що така зміна пріоритету відповідно не впливає на кінцевий результат.

4.2 Сильні та слабкі сторони створеної системи

Отже, базуючись на доступній інформації вдалось розробити функціональне програмне рішення для моделювання медичного сортування. На основі кінцевого продукту можна виконувати симуляцію складних стохастичних процесів з можливістю аналізу за допомогою вбудованих статистичних методів.

Основною перевагою отриманої програми є її функціональність та масштабованість. Утворена система частково є фреймворком, що об’єднує інструментарій, і метамоделлю, що задає правила для побудови моделей в ній самій.

Розробка велась з розрахунку на дотримання таких принципів, як розширюваність, повторне використання та лаконічність. Були використані

підходи об’єктоорієнтованого програмування, особливо абстракція та поліморфізм, перш за все завдяки відділеній, але схожій внутрішній логіці представлених сутностей.

Слабким місцем програми є її крихкість (в розумінні неочікуваних випадків тривалої роботи або помилок при редагуванні налаштувань) внаслідок гнучкої або майже відсутньої типізації, відсутності засобів тестування функцій системи та складних функціональних зв’язків.

Швидкість програми є також її слабкою стороною, що типово для програм на мові Python. При деяких умовах сумарна кількість ітерацій циклів програми може стати дуже великою. Також при додаванні нових варіантів до списку параметрів для тестування асимптотична складність росте блискавично швидко.

4.3 Варіанти подальшого розвитку

Щодо способів варіантів подальшого розвитку системи, то тут можна виділити наступні вектори:

- Збільшення кількості параметрів та функціональних зв’язків моделі. Наприклад, можна додати елементи, що відповідатимуть за діагнози, ускладнення, випадкові витрати, розширити список процесів.
- Розширення цільової функції, кінцевої оцінки симуляції (частковий випадок попереднього вектора). Логічним кроком у розвитку системи буде додавання показника функціонального відновлення пацієнтів та різноманітних оцінок, які потрібно оптимізувати. До слова, така оптимізація медичної допомоги, де поряд із мінімізацією смертності ставиться ціль мінімізації витрат буде вважатись неетичною. При цьому дана програма уже дає можливість працювати із функцією витрат завдяки гнучкості класу ресурсів.

- Вплив на реалістичність системи через її узгодження із емпіричними даними. Відсіювання неможливих в реальності випадків.
- Збільшення стійкості системи. Накладання обмежень на дії, відмова від залежностей, розробка та автоматизація тестування.
- Розробка інструментарію для статистичного аналізу. Нові способи візуалізації, розумний пошук оптимального рішення.
- Збільшення швидкості. Дострокові виходи із циклів, векторизація математичних дій для великих об’ємів та при потребі використання інших мов програмування, наприклад перенесення деяких функцій на CPython чи C++.
- Зрозумілість. Додавання коментарів, покращення логування.

ВИСНОВКИ

Ця робота зосереджувалося на розробці методів моделювання медичного сортування. Були описані й проаналізовані підходи та алгоритми до медичного сортування, оглянуто теоретичний матеріал та обрано основний підхід до розв’язання задачі. Теоретичні підходи та вихідні компоненти моделі були сформульовані із урахуванням доступних емпіричних даних. Зрештою було розроблене комплексне програмне рішення з поясненнями та прикладами його застосування та аналізом його сильних і слабких сторін.

В кінцевому результаті була створена програма для симуляції роботи пункту медичної допомоги, що здатна за один цикл роботи провести ряд симуляцій для різних вхідних налаштувань із аналізом результатів кожної симуляції. Результат кожної симуляції базується на численних параметрах та функціях моделі. Програма передбачає моделювання багатьох процесів медичних установ, серед яких приймання пацієнтів, оцінка їх стану та маркування, транспортування, надання допомоги, що моделюються в рамках кожної симуляції. Кінцева система має досить складну, але гнучку взаємодію із ресурсами моделі, що задаються дослідником.

Кінцевий результат покликаний закрити прогалину в дослідженнях медичного сортування і дати дослідникам цієї сфери новий інструмент. Робота має потенціал для практичного застосування та може сприяти знаходженню оптимальних рішень при дослідженні складних ситуацій та систем, де вони виникають, якщо їх відтворення в рамках експерименту неможливе. Загалом, отриманий інструмент має допомогти досліджувати широкий спектр складних стохастичних моделей, що виникають на стику різних дисциплін.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Triage in Medicine, Part I: Concept, History, and Types Annals of Emergency Medicine, Volume 49 — 2007.
2. Kenneth V. Iserson, John C. Moskop. Triage in Medicine, Part I: Concept, History, and Types — Annals of Emergency Medicine, Volume 49 — 2007, ISSN 0196-0644, doi:10.1016/j.annemergmed.2006.05.019.
3. Bazyar J, Farrokhi M, Khankeh H. Triage Systems in Mass Casualty Incidents and Disasters: A Review Study with A Worldwide Approach — Open Access Maced J Med Sci — 2019. doi:10.3889/oamjms.2019.119.
4. Enrico Dippenaar. Triage systems around the world: a historical evolution — International Paramedic Practice — 2019.
5. Leigha Clarkson, Mollie Williams. EMS Mass Casualty Triage — StatPearls Publishing — 2023.
6. Zachariasse, Joany & Hagen, Vera & Seiger, Nienke & Mackway-Jones, Kevin & Veen, Mirjam & Moll, Henriette. Performance of triage systems in emergency care: A systematic review and meta-analysis — BMJ Open. 9(5) —2019. doi:10.1136/bmjopen-2018-026471.
7. Стандарт екстреної медичної допомоги «Медичне сортування при масовому надходженні постраждалих на ранньому госпітальному етапі»: ГС 2022-368 — Офіц. вид. — Міністерство охорони здоров’я України, 2022. — (Нормативний документ Міністерство охорони здоров’я України. Стандарт).

8. Гориславець П.А. Моделювання діяльності страхових компаній на фінансовому ринку. Конспект лекцій — Національний університет «Львівська Політехніка» — 2016.
9. Borner, Katy & Boyack, Kevin & Milojevic, Stasa & Morris, Steven. An Introduction to Modeling Science: Basic Model Types, Key Definitions, and a General Framework for the Comparison of Process Models — Understanding Complex Systems — 2012. doi:10.1007/978-3-642-23068-4_1.
10. Ingalls, Ricki. (2001). Introduction to simulation. — Conference: Simulation Conference, 2002. Proceedings of the Winter. Volume 1 — 2001. doi:10.1109/WSC.2002.1172861.
11. Mayhew, Les & Smith, David. Using queuing theory to analyse the Government's 4-h completion time target in Accident and Emergency departments — Health care management science — 2008. doi:10.1007/s10729-007-9033-8.
12. Zonderland, M. E.; Boucherie, R. J. Queuing Networks in Health Care Systems — Handbook of Healthcare System Scheduling. International Series in Operations Research & Management Science. Vol. 168 — 2012. doi:10.1007/978-1-4614-1734-7_9. ISBN 978-1-4614-1733-0.
13. Begen, Mehmet & Patrick, Jonathan. Markov Decision Processes and Its Applications in Healthcare — Handbook of Healthcare Delivery Systems — 2011.
14. David G. Luenberger, Yinyu Ye. Linear and Nonlinear Programming — International Series in Operations Research & Management Science (ISOR, volume 116) — 2008. doi:10.1007/978-3-319-18842-3.
15. Авраменко В.С., Авраменко А.С. Проєктування інформаційних систем. Навчальний посібник — Черкаський національний

університет імені Богдана Хмельницького — 2017. ISBN 978-966-920-208-6.

16. Moon SH, Shim JL, Park KS, Park CS. Triage accuracy and causes of mistriage using the Korean Triage and Acuity Scale — PLoS One. — 2019. doi: 10.1371/journal.pone.0216972. PMID: 31490937; PMCID: PMC6730846.
17. Rauf R, von Matthey F, Croenlein M, Zyskowski M, van Griensven M. Changes in the temporal distribution of in-hospital mortality in severely injured patients — An analysis of the TraumaRegister DGU. PLOS ONE 14(2) — 2019. doi:10.1371/journal.pone.0212095

ДОДАТОК

```
from dataclasses import dataclass
from typing import Callable, Iterator, Optional, Any
import numpy as np
import plotly.express as px
import plotly.graph_objects as go
from scipy.stats import normaltest, shapiro, kstest, norm, t, f_oneway
import itertools
import time


MAX_ITER = 999  # maximum iteration amount for some loops
NUM_LEN = 7  # maximum length of number
ROUND = 4  # logging settings >
LOG_LEVEL = 1
def log(text, level=0):
    if level <= LOG_LEVEL:
        tabs = level * '\t'
        print(f'{tabs}{text}')


class Gen:
    def norm_pos(mean: float, std: float) -> float:
        assert 0 < mean and 0 < std, 'incorect mean or std values'
        for _ in range(MAX_ITER):
            p = np.random.normal(mean, std)
            if p > 0:
                return p
        return -p

    def treatment_time(expected_tt: int) -> int:
        return np.random.poisson(expected_tt)

    def patients(n: int, categories_n: int, prob: list[float], means: list[list[float]], stds: list[list[float]], tt: int) -> list['Patient']:
        norm_prob, categories = [i/sum(prob) for i in prob], [i for i in range(categories_n)]
        ps = []
        for _ in range(n):
            category = np.random.choice(categories, p=norm_prob)
            death_rates = tuple(Gen.norm_pos(means[category][i], stds[category][i]) for i in range(len(means[0])))
            treatment_time = Gen.treatment_time(tt)
            ps.append(Patient(death_rates, treatment_time))
        return ps

    def est_sample_size(std_dev: float, deviation: float = 0.1, probability: float = 0.95) -> tuple[int, int]:
        z = norm.ppf(1 - (1 - probability) / 2)  # Z-score corresponding to the desired probability
        return int(np.ceil(((z * std_dev) / deviation) ** 2)), int(np.ceil(std_dev ** 2 / deviation ** 2 / (1 - probability)))
```

```
class Patient:
    def __init__(self, death_rates: tuple[float, float, float], ttl: int) -> None:
        self._death_rates = death_rates
        self.ttl = ttl  # treatment time left

        self.label = 0
        self.stage = 0
        self.saved = None

    @property
    def death_rate(self) -> float:
        return self._death_rates[self.stage]

    @death_rate.setter
    def death_rate(self, new: float) -> None:
        new_rates = list(self._death_rates)
        new_rates[self.stage] = new
        self._death_rates = new_rates

    def __repr__(self) -> str:
            return  f'Patient(death_rate={self.death_rate:.{ROUND}}, ttl={self.label}, label={self.label}, stage={self.stage},
saved={self.saved})'


class Patients(set):
    def __init__(self, existing_set: set = set()) -> None:
        super().__init__()
        super().update(existing_set)

    @property
    def active(self) -> 'Patients':
        return Patients({patient for patient in self if patient.saved is None})

    @property
    def dead(self) -> 'Patients':
        return Patients({patient for patient in self if patient.saved is False})

    @property
    def saved(self) -> 'Patients':
        return Patients({patient for patient in self if patient.saved})

    def stage(self, stage) -> 'Patients':
        return Patients({patient for patient in self.active if patient.stage == stage})

    def __bool__(self) -> bool:
        return len(self) > 0


@dataclass
class Resource:
```

```
        n: int
        reusable: bool
        parralel_use: bool
        allocation_amount: dict[Callable[[Patients, int], int], Callable[[Patients, int], int], Callable[[Patients, int], int]]
        amount_use: dict[Callable[[Patient], int], Callable[[Patient], int], Callable[[Patient], int]]
        time_use: dict[Callable[[Patient], float], Callable[[Patient], float], Callable[[Patient], float]]
        name: str = ''
        _to_be_renew: int = 0
        _taken_from: object = None

        def _take_some(self, n: int) -> 'Resource':
            awailable = min(self.n, n)
            self.n -= awailable
                return Resource(awailable, self.reusable, self.parralel_use, self.allocation_amount, self.amount_use, self.time_use,
self.name, 0, self)

        def allocate(self, patient_list: list | Patients, for_what: str) -> Optional['Resource']:
            patients = Patients(patient_list) if isinstance(patient_list, list) else patient_list
            n = self.allocation_amount[for_what](patients, self.n)
            if not n is None:
                return self._take_some(n)

        def use_if_enough(self, patient: Patient, for_what: str) -> bool:
            n = self.amount_use[for_what](patient)
            if self.n >= n:
                self.n -= n
                if self.reusable:
                    self._to_be_renew += n
                return True

        def time_consumption(self, patient: Patient, for_what: str) -> float:
            return self.time_use[for_what](patient) / self.n if self.parralel_use and self.n else self.time_use[for_what](patient)

        def return_back(self) -> None:
            if self._taken_from:
                self._taken_from.n += self.n + self._to_be_renew
            self.n = 0

        def __repr__(self) -> str:
            return f'Resource(name={self.name}, n={self.n})'


    class Hospital:
        def __init__(self, settings: 'Settings') -> None:
            self.patients = Patients()
            self.settings = settings
            self._mark, self._transport, self._treat = None, None, None

        def add(self, patients: [Patients, Patient]) -> None:
            if isinstance(patients, Patient):
```

```
        patients = [patients, ]
    for patient in patients:
        self.patients.add(patient)

def _action(self, _f: Callable[[Patient], None], name: str, patient_list: list | Patients) -> Iterator[None]:
    allocated = [r.allocate(patient_list, name) for r in self.settings.resources]
    log(f'New action: {name}, resources on base: {self.settings.resources}', 3)
    log(f'Allocated resources: {allocated}', 3)
    time = 0
    break_ = False
    for patient in patient_list:
        log(f'{name} action for {patient}', 4)
        for r in allocated:
            time += r.time_consumption(patient, name)
            if not r.use_if_enough(patient, name):
                log(f'Not enough required {r}', 5)
                break_ = True
                break
            if time > 1:
                log(f'time is off, actions on pause!', 5)
                yield
                log(f'continuing the action', 5)
        if break_:
            break
        _f(patient)
    [r.return_back() for r in allocated]
    log(f'End of action. Resources are returned.', 3)

def dynamic(self):
    for patient in self.patients:
        patient.death_rate = self.settings.death_rate_dynamic(patient)
        if np.random.uniform() < patient.death_rate:
            patient.saved = False
            log(f'patient has died, {patient}', 4)

def _mark_patient(self, patient: Patient) -> None:
    patient.label = self.settings.triage_label(patient)
    patient.stage = 1
    log(f'...patient is marked, {patient}', 4)

def _transport_patient(self, patient: Patient) -> None:
    patient.stage = 2
    log(f'...patient is transported, {patient}', 4)

def _treat_patient(self, patient: Patient) -> None:
    if patient.saved is None and patient.stage == 2:
        patient.ttl -= 1
        if patient.ttl > 0:
            log(f'...patient is treated, {patient}', 4)
        else:
```

```
            patient.saved = True
            log(f'...patient has recovered, {patient}', 4)

    def mark(self) -> None:
        try:
            next(self._mark)
        except (StopIteration, TypeError):
            try:
                self._mark = self._action(self._mark_patient, 'mark', self.patients.stage(0))
                next(self._mark)
            except StopIteration:
                pass

    def transport(self) -> None:
        try:
            next(self._transport)
        except (StopIteration, TypeError):
            try:
                self._transport = self._action(self._transport_patient, 'transport', self.settings.triage_queue(self.patients.stage(1)))
                next(self._transport)
            except StopIteration:
                pass

    def treat(self) -> None:
        try:
            next(self._treat)
        except (StopIteration, TypeError):
            try:
                self._treat = self._action(self._treat_patient, 'treat', self.patients.stage(2))
                next(self._treat)
            except StopIteration:
                pass

    def run(self) -> float:
        return self.settings.main_dynamic(self)

    def __repr__(self) -> str:
        ps = self.patients
        return f'HospitalPatients(active={len(ps.active)}, dead={len(ps.dead)}, saved={len(ps.saved)})'


class Tester:
    def __init__(self, settings: 'Settings', **kwargs: Any) -> None:
        self.values = [*kwargs.values()]
        self.keys = [*kwargs.keys()]
        self.var_len = len(self.values)
        self.combs = []
        self.results = []
        self.settings = settings  # get_settings()
        assert all(isinstance(v, list) and v for v in self.values), 'values should consist of non-empty lists'
```

```
    assert all(k in self.settings.__dict__.keys() for k in self.keys), 'keys should consist of settings class names'

@property
def res_len(self) -> int:
    return len(self.results)

@property
def means(self) -> list:
    return [r.mean() for r in self.results]

def _test_results(self) -> np.ndarray:
    res = np.array([])
    for i in range(self.settings.repeats):
        log(f'Repeating test {i}', 2)
        res = np.append(res, Hospital(self.settings).run())
    return res

def _run_tests(self, comb: tuple[...]) -> None:
    self.results.append(self._test_results())
    self.combs.append(comb)

def run(self) -> None:
    start_time = time.time()
    if not self.var_len:
        log(f'Starting the test...', 1)
        self._run_tests((len(self.results), ))
    else:
        for comb in itertools.product(*self.values if self.var_len > 1 else self.values):
            [setattr(self.settings, self.keys[i], comb[i]) for i in range(self.var_len)]
            log(f'Settings has been updated for combination {comb}, running tests...', 1)
            self._run_tests(comb)
    log(f'Total execution time: aprox. {time.time()-start_time} seconds', 1)

def _repr(self, i) -> str:
    return f'Combination {self.combs[i]} for keys ({", ".join(self.keys)}):'

def show(self, _max=10) -> None:
    print('RESULTS')
    for i in range(self.res_len):
        end = '...' if len(self.results[i]) > _max else ''
        print(f'{self._repr(i)}  {", ".join([str(round(i, ROUND)) for i in self.results[i][:_max]])}{end}')

def describe(self) -> None:
    print('RESULTS DESCRIPTIONS')
    for i in range(self.res_len):
        r = self.results[i]
        print(f'{self._repr(i)}  mean: {r.mean():{NUM_LEN}.{ROUND}},  sd: {r.std():{NUM_LEN}.{ROUND}}')

def normality(self) -> None:
    print('NORMALITY TESTS P-VALUES')
```

```
            for i in range(self.res_len):
                r = self.results[i]
                print(f'{self._repr(i)}  k-squared: {normaltest(r).pvalue:.{ROUND}}, shapiro: {shapiro(r).pvalue:.{ROUND}}')

        def means_prob(self, deviation: float = 0.1) -> None:
            print(f'MEANS PROBABILITY TO FALL WITHIN DEVIATION OF {deviation} ...')
            for i in range(self.res_len):
                r = self.results[i]
                std, df = r.std(), len(r)-1
                assert std > 0, 'Standart deviation is zero'
                z_left, z_right = - deviation / (std / np.sqrt(df)), + deviation / (std / np.sqrt(df))
                print(f'{self._repr(i)} ... is expected to be {norm.cdf(z_right) - norm.cdf(z_left)}')

        def anova(self) -> None:
            print(f'ONEWAY ANOWA TESTS P-VALUE FOR EACH PAIR OF SIMULATED RESULTS')
            for i in range(self.res_len):
                for j in range(i + 1, self.res_len):
                    print(f'Comparing combination {self.combs[i]} with {self.combs[j]}: {f_oneway(self.results[i],
self.results[j]).pvalue:.{ROUND}}')

        def bars(self) -> None:
            fig = px.bar(y=self.means, text=[str(c) for c in self.combs])
            fig.update_layout(xaxis_title=f'variables {", ".join(self.keys)}', yaxis_title='results', showlegend=False)
            fig.update_traces(textposition='outside')
            fig.update_xaxes(visible=False)
            fig.show()

        def hist(self) -> None:  # simple histogram
            cols = []
            for i in range(self.res_len):
                cols.extend([self.combs[i]]*len(self.results[i]))
            fig = px.histogram(x=np.concatenate(self.results), color=cols, nbins=20)
            fig.update_layout(xaxis_title='results', yaxis_title='amount')
            fig.show()

        def histogram(self, _max=5) -> None:
            fig = go.Figure()
            for i in range(self.res_len)[:_max]:
                fig.add_trace(go.Histogram(x=self.results[i]))

            fig.update_layout(barmode='overlay')
            fig.update_traces(opacity=0.75)
            fig.show()

        def heatmap(self) -> None:
            assert self.var_len == 2, 'The amount of test variables is different from 2'
            x_ind, y_ind = 0, 1
            x, y = [c[x_ind] for c in self.combs], [c[y_ind] for c in self.combs]
            fig = go.Figure(data = [go.Heatmap(x=x, y=y, z=self.means, texttemplate='%{z}')])
            fig.update_layout(xaxis_title=self.keys[x_ind], yaxis_title=self.keys[y_ind], coloraxis_colorbar=dict(title='result'))
```

```
        fig.show()


class Settings:
    a = 1  # additional variable
    n = 20  # number of patients
    categories_n = 3
    _min_label, _max_label = 1, categories_n
    death_means = [
        [0.1, 0.1, 0.1], # death probabitity by stage for category 1
        [0.1, 0.5, 0.00001], # death probabitity by stage for category 2
        [0.3, 0.00001, 0.00001], # death probabitity by stage for category 3
    ]
    death_stds = [
        [0.05, 0.05, 0.05], # standart dev. of prob. of death by stage for category 1
        [0.05, 0.05, 0.05], # standart dev. of prob. of death by stage for category 2
        [0.05, 0.05, 0.05], # standart dev. of prob. of death by stage for category 2
    ]
    categories_prob = (0., 0.5, 0.5)  # relative probability for categories
    treatment_time = 2  # expected treatment time
    def death_rate_dynamic(patient: Patient) -> float:  # how death rate is changing wtih time
        return patient.death_rate * 0.9

    resources = [
        Resource(10,  # n
                 True,  # reusable
                 True,  # parralel_use
                 {  # allocation_amount >
            'mark': lambda patients, available: int(available*0.7) if len(patients.stage(0)) > int(available*0.7) // 0.2 else available,
            'transport': lambda patients, available: 2,
            'treat': lambda patients, available: available*1
                 },
                 {  # amount_use >
            'mark': lambda patient: 1,
            'transport': lambda patient: 1,
            'treat': lambda patient: 1
                 },
                 {  # time_use >
            'mark': lambda patient: 0.2,
            'transport': lambda patient: 0.1,
            'treat': lambda patient: 0.5
                 },  # name >
            'perosnel'),

        # here is a place for another resource
    ]

    def _triage_inacuracy(label: int) -> int:
        x = np.random.uniform()
        if x < 0.15:  # mistriage probability
```

```
                if x < 0.3:
                    return label+1  # overtriage
                else:
                    return label-1  # undertriage
            return label

        def triage_label(patient: Patient) -> int:
            label = max(patient.death_rate - 0.2, 0) // 0.1
            label = Settings._triage_inacuracy(label)
            return int(max(Settings._min_label, min(Settings._max_label, label)))

        def triage_queue(patients: Patients) -> list[Patient]:
            d = {1:1, 2:3, 3:2} if Settings.a else {1:1, 2:2, 3:3}
            queue = sorted(patients.stage(1), key=lambda p: d[p.label], reverse=True)
            log(f'Labels in patients queue: {", ".join([str(p.label) for p in queue])}', 3)
            return queue

        repeats = 300
        _time = 0
        _max_time = MAX_ITER

        def _result(hospital: Hospital) -> float:
            return len(hospital.patients.saved) / Settings.n

        def main_dynamic(hospital: Hospital) -> float:
                    hospital.add(Gen.patients(Settings.n,  Settings.categories_n,  Settings.categories_prob,  Settings.death_means,
Settings.death_stds, Settings.treatment_time))
            for Settings._time in range(Settings._max_time):
                log(f'New hospital time cycle in main_dynamic {Settings._time}', 3)
                if hospital.patients.stage(0):
                    hospital.mark()
                if hospital.patients.stage(1):
                    hospital.transport()
                hospital.treat()
                hospital.dynamic()
                log(f'Hospital time cycle stats: {hospital}, time: {Settings._time}', 3)
                if not hospital.patients.active:
                    break
            return Settings._result(hospital)


    tester = Tester(Settings, a=[1, 0])
    tester.run()
    tester.describe()
```