МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Чернігівська політехніка»

МОДЕЛЮВАННЯ СИСТЕМ

МЕТОДИЧНІ ВКАЗІВКИ

до виконання розрахунково-графічної роботи
для здобувачів вищої освіти
за освітньою програмою “Комп’ютерна інженерія”
(освітній ступінь бакалавр)

Обговорено і рекомендовано
на засіданні кафедри
інформаційних і комп'ютерних систем
Протокол № 1 від 25.01.24

Чернігів 2024

Моделювання систем. Методичні вказівки до виконання розрахунково-графічної роботи для для здобувачів вищої освіти за освітньою програмою „Комп'ютерна інженерія". /Укл.: Бивойно П.Г., Пріла О.А, Бивойно Т.П. – Чернігів, 2024. – 64 с.

Укладачі: Бивойно Павло Георгійович, канд. техн. наук, доцент
Пріла Ольга Анатоліївна, канд. техн. наук, доцент
Бивойно Тарас Павлович, старший викладач

Відповідальний за випуск: Базилевич В.М., зав. кафедрою інформаційних и комп'ютерних систем, канд.економ.наук.

Рецензент: Білоус І.В., канд. техн. наук, зав. кафедрою інформаційних технологій та програмної інженерії НУ «Чернігівська політехніка»

ЗМІСТ

ВСТУП

Поняття розрахунково-графічна робота (РГР) прийшло у вищу школу з дисциплін навчальних планів інженерно-механічних спеціальностей. У межах розрахунково-графічної роботи студент повинен був виконати деякі розрахунки і графічні побудови (креслення, ескіз, діаграму), що були необхідні для вирішення деякої інженерної проблеми. До того ж часто використовувалися так звані графоаналітичні методи розрахунків. Тому назва «розрахунково-графічна робота» була, безсумнівно, доречною.

Проте потім, завдання студентам, що були пов'язані з вирішенням проблем інших галузей знань, теж почали називати розрахунково-графічними роботами. З'явилися РГР з хімії, бухгалтерського обліку та інші. У цих завданнях, зазвичай, ніяких графічних елементів вже не було, залишалися тільки розрахунки, а інколи і їх не було, та назва збереглася.

У курсі «Моделювання систем» виконання розрахунково-графічної є частиною самостійної роботи студентів над дисципліною. Робота передбачає проведення об’єктно-орієнтованого аналізу деякої предметної області і подальшу побудову імітаційної моделі цієї області. Ніяких розрахунків робота не передбачає, але має буди програмна реалізація моделі та проведені дослідження системи шляхом моделювання. Розрахунки виконує комп’ютер.

Окрім аналізу предметної області студент має проаналізувати можливості фреймворку Simulation Java та визначити перелік класів цього фреймворку, що можуть бути використані або безпосередньо, або як батьківські класи для реалізації класів майбутньої моделі. У процесі програмної реалізації мають бути створені шар подання, шар моделі та шар компонентів.

Номер варіанту завдання для РГР визначається відповідно до номера студента у списку групи. Також студент може і сам обрати предметну область для моделювання, але завдання обов'язково має бути погоджено з викладачем.

Ргр студенти виконують поетапно, одночасно з виконанням лабораторних робіт. З кожного етапу створюється звіт, який є частиною звіту з лабораторної роботи.

Бали за етапи розрахунково-графічної роботу виставляються з урахуванням своєчасності та якості виконання, а також оформлення розділів звіту.

Результатом розрахунково-графічної роботи є jar файл, який можна виконати, а також .zip файл проекту.

Методичні вказівки створено на основі методичних вказівок видання 2023 року. За рекомендацією старшого викладача Бивойно Т.П. та за його участі змінено технологію створення моделі та активних об’єктів (етапи 3 та 4).

1 ЗАВДАННЯ ДО РОБОТИ

1. Моделювання продажу квитків в аеропорті, варіант KassaInAirport

У курортному аеропорті малої авіації працюють кілька касирів із продажу квитків, причому до кожної каси окрема черга, а пасажири стають у ту, де черга менше.

Якщо довжина черги перевищує деяке критичне значення, пасажир відразу йде на посадку,щоб придбати квиток у однієї зі стюардес.

На продаж квитка стюардеса витрачає такий же час, як і касир. Але основне завдання стюардеси - проводити групи пасажирів до літаків, які чекають завантаження, тому періодично стюардеси йдуть із групою пасажирів на літовище. А якщо літака нема, пасажири чекають.

В аеропорті обмежена кількість малих літаків, які летять і повертаються.

Якщо довжина всіх черг на покупку квитків до вільних стюардес теж перевищують критичне значення, пасажир іде на залізничний вокзал.

2. Моделювання роботи ділянки тестування комп'ютерів, варіант TestPC

На останній стадії складання системних блоків комп’ютера виконується їх тестування та пакування. Блоки з головного конвеєра надходять на ділянку тестування через випадкові проміжки часу. На ділянці тестування є площадка для розміщення блоків, розмір якої обмежений. Якщо на площадці немає місця для чергового блоку, він відправляється на пакування без тестування. Блоки, що пройшли тестування йдуть на пакування. Для пакування потрібні картонні коробки, кількість яких на площадці обмежено. Якщо коробок нема, пакувальник іде на склад за новою партією коробок.

3. Моделювання роботи цеху перемотування ниток, варіант Nitki

У цеху перемотування шовку робітниці обслуговують перемотувальні машини. Порядок обслуговування робочого місця машини наступний: установлюється бобіна, заправляється нитка і включається перемотування. Під час перемотування можливі обриви нитки. При обриві нитки необхідно усунути обрив і продовжити перемотування. Коли перемотування закінчується, процес повторюється.

4. Моделювання посівної компанії, варіант BeanFeast

Сівбу зернових на полі забезпечують декілька сівалок. Зерно для сівалок підвозять вантажівки. З однієї вантажівки можна завантажити декілька сівалок. Вантажівки завантажуються на складі.

5. Моделювання авіаперевезень, варіант AviaBridg

У аеропорт прибувають контейнери з гуманітарною допомогою для відправлення у зону стихійного лиха. Контейнери після митного контролю потрапляють на навантажувальну площадку обмежених розмірів, а з площадки бригада вантажників завантажує контейнери у літак, якщо він готовий для

навантаження. Літак відправляється відразу після заповнення. У зоні стихійного лиха літак розвантажують. Після повернення на свій аеродром літак проходить технічне обслуговування й знову готовий до завантаження.

6. Моделювання роботи річкового порту, варіант RiverPort

У річковий порт протягом доби прибувають баржі з кавунами в контейнерах Бригади вантажників перевантажують контейнери в автомобілі, що прибувають у порт. В автомобіль міститься кілька контейнерів. Автомобілі розвозять кавуни по різних районах міста. Якщо автомобілів нема, контейнери вивантажуються на розвантажувальний майданчик обмеженого розміру. Якщо баржі під розвантаженням нема, автомобілі завантажуються контейнерами з майданчика.

Відображати чергу барж, завантаженість майданчика, чергу авто.

За одиницю часу взяти середнє значення часу на обробку контейнера.

Підібрати налаштування, щоб система працювала нормально.

Статистику по чергам: авто, барж, вантажного майданчика. Часу чекання барж, чекання авто, простою бригад. Час обслуговування контейнера.

Дослідити вплив кількості бригад на всі ці параметри при різних значеннях кількості авто (не менше 3-х значень).

7. Моделювання роботи банку для автомобілістів, варіант AvtoBank

У банку для автомобілістів віконеце з касиром і має під'їзну смугу обмеженої довжини. Автомобілі прибувають через випадкові інтервали часу. Якщо всі місця на смузі зайняті, то черговий клієнт вважається загубленим. Після обслуговування, клієнт виїжджає на смугу для виїзду з банку, але проблема полягає в тому, що автомобіль може потрапити на дорогу тільки з появою певного інтервалу між їдучими по ній автомобілями. Якщо на смузі для виїзду немає вільних місць, автомобіль не може від'їхати від віконця касира й обслуговування припиняється.

8. Моделювання розвантаження автомобілів, варіант TransAvto

Вантажівки прибувають на розвантажувальну станцію у випадкові моменти часу. Станція знаходиться біля кордону, тому до розвантаження вантажівки мають пройти митний контроль. Час митного контролю та розвантаження вантажівки випадковий. Розвантаженням вантажівок займається декілька бригад вантажників, але кожну вантажівку обслуговує одна бригада..

9. Моделювання виборів, варіант Election

На виборчій дільниці працює кілька реєстраторів, до яких надходить потік виборців. Зареєструвавшись, виборець стає у чергу до кабінок, потім заходить до неї і вибирає свого кандидата. Після цього він опускає бюлетень до однієї з урн.

Відображати черги до реєстраторів, кабінок, урн.

За одиницю часу взяти середнє значення часу на опускання бюлетеня до

урни.

Підібрати налаштування, щоб система працювала нормально.

Статистику по чергам. Часу простою реєстраторів. Час що витратив виборець на голосування.

Дослідити вплив кількості реєстраторів на всі ці параметри при достатній кількості кабінок та урн та визначитися з раціональним значенням цього параметру.

Дослідити вплив кількості кабінок на параметри обслуговування при раціональній кількості реєстраторів і достатній кількості урн та визначитися з раціональною кількістю урн.

Дослідити вплив кількості кабінок на параметри обсуговування.

10. Моделювання роботи станції швидкої допомоги, варіант Ambulance

На станції швидкої допомоги бригади лікарів чекають виклику до хворих. Коли надходить заявка, бригада їде до хворого й надає йому допомогу. Існує ймовірність того, що хворого буде потрібно відвезти в лікарню. Виконавши необхідну роботу, бригада повертається на станцію. Під час повернення, бригада може одержати завдання їхати до іншого хворого з того ж району. У цьому випадку хворий буде обслужений скоріше.

11.Моделювання лісозаготівель, варіант SibirLes

У Карпатах бригада лісорубів заготовляє ліс. На спилювання та обробку дерева лісоруб витрачає деякий час. Стовбури, що підготовлені до вивозу, за допомогою навантажувача навантажують у лісовози. Лісовози відвозять стовбури до деревообробного цеху і повертаються до лісу. У цеху працює кілька деревообробних верстатів, що роблять із стовбурів кругляк.

12. Моделювання зернозбиральних робіт, варіант Zerno

Сільськогосподарський загін по збиранню зерна складається з декількох зернозбиральних комбайнів й автомобілів для вивезення зерна з поля. Комбайни працюють цілодобово. Коли комбайн намолотить повний бункер зерна, він зупиняється й чекає автомобіля, якщо його ще немає. Коли автомобіль під'їжджає, зерно перевантажується в кузов автомобіля, і комбайн продовжує роботу. Автомобіль відвозить зерно на елеватор, чекає розвантаження, потім повертається й стає в чергу на завантаження зерном від комбайнів.

13. Моделювання роботи кар'єру, варіант Ruda

У кар'єрі самоскиди доставляють руду від екскаватора до каменедробарки й там вивантажують руду в бункер, якщо дробарка чекає завантаження. Якщо дробарка завантажена, то руду висипають на площадку. Для завантаження дробарки із площадки використається навантажувач.

14. Моделювання роботи відділу технічного контролю, варіант TestTV

На заключній стадії виробництва телевізорів здійснюється їх перевірка. Якщо під час перевірки виявилося, що телевізор працює неправильно, то він направляється в пункт налаштування. Після налаштування телевізор знову направляється в пункт контролю для перевірки. Телевізори, які пройшли перевірку, направляються в цех пакування.

15. Моделювання роботи ресторану, варіант Restoran

У ресторан приходять клієнти й сідають за вільні столики. Якщо вільних столів нема, клієнти чекають на вулиці. Офіціант приймає замовлення клієнта з декількох блюд і робить замовлення на кухні. Приготовлені блюда офіціант приносить клієнтові. Після того як клієнт пообідав, він запрошує офіціанта, щоб розрахуватися. Коли офіціант підходить, клієнт розраховується з ним і залишає ресторан.

16. Моделювання роботи поліклініки, варіант Polyclinlk

У поліклініці працює кілька лікарів. Пацієнт, що прийшов в перший раз, одержує талон до одного з лікарів і стає в чергу. Лікар якийсь час оглядає чергового пацієнта й призначає курс лікування. Після курсу лікування хворий знову приходить до лікаря і йому з деякою ймовірністю може знову знадобитися курс лікування. Існує також імовірність того, що хворому призначаються амбулаторні дослідження. Одержавши, через деякий час, результати цих досліджень, хворий знову приходить до того ж лікаря.

17. Моделювання роботи багажного відділення, варіант Bag

В аеропорт прилітають пасажирські літаки різної місткості. Пасажири виходять із літака й проходять до місця одержання багажу. У зв'язку з крадіжками багажу його стали видавати персонально кожному пасажирові. Деякі з пасажирів відразу стають у чергу за багажем, інші заходять перекусити, а потім ідуть за багажем. Тим часом вантажники вивантажують багаж з літака на візки. Візки перевозять багаж до місця видачі, і там їх розвантажують. Кожен пасажир літака має квитанцію, по якій він повинен одержати багаж. Коли підходить черга, пасажир дає квитанцію службовцеві багажного відділення, а той іде подивитися, чи прибув відповідний багаж, і віддає його пасажирові. Якщо багажу нема, то пасажир через деякий час знову приходить за ним.

18. Моделювання роботи докерів, варіант Doker

У порт прибувають судна з контейнерами для розвантаження. Контейнери ставлять на розвантажувальну площадку обмеженого розміру. Якщо місць на площадці нема, розвантаження припиняється. Цех автоперевезень порту має кілька тягачів, які перевозять контейнери з порту на залізничну станцію, де кількість місць для контейнерів не обмежена. Для розвантаження суден і завантаження тягачів використається той самий кран.

Докери працюють цілодобово. Після розвантаження на станції тягач повертається за наступним контейнером.

19. Моделювання роботи супермаркету, варіант SuperMarket

Покупці приходять у магазин через випадкові проміжки часу. У магазині покупець може зробити кілька покупок. Покупець розраховується за покупки в касах на виході. Час розрахунку на касі й час перебування покупця в магазині залежить від кількості покупок. Якщо черга перевищує критичне значення (випадкова величина), то покупець не заходить до магазину. Кількість покупців, що втрачено за одиницю часу теж обчислювати.

20. Моделювання роботи потокової лінії, варіант Line

На потокову лінію, де виконуються послідовно дві операції (відповідно є два робочих місця), оброблювані деталі надходять через випадкові інтервали часу. Оброблювані деталі громіздкі, тому кількість деталей, що можна розмістити перед робочими місцями обмежена. Якщо на площадці до другого робочого місця немає вільних місць, то перше робоче місце блокується. Якщо немає вільних місць перед першим робочим місцем, чергова деталь передається на склад без обробки.

21. Моделювання роботи технічного дивізіону, варіант Raketa

У технічному дивізіоні ракетного полку складаються й перевіряються ракети перед відправленням у вогневий дивізіон. На складання ракети витрачається якийсь час, а потім ракета перевіряється послідовно на 2-х іспитових стендах. Перед кожним стендом є площадка для розміщення ракет, розмір якої обмежений. Якщо на площадці перед першим стендом немає місця для нової ракети, що надійшла з ділянки складання, вона відправляється без перевірки. Якщо немає місця на площадці перед другим стендом, то робота на першому стенді припиняється.

22.Моделювання лісозаготівель, варіант KarpatWood

У Карпатах бригада лісорубів заготовляє ліс. На спилювання та обробку дерева лісоруб витрачає деякий час. Стовбури, що підготовлені до вивозу, за допомогою навантажувача навантажують у лісовози. Лісовози відвозять стовбури на залізничну станцію, де вони навантажуються у вагони, що періодично надходять.

23. Моделювання роботи парку відпочинку, варіант Gardens

Відпочиваючі приходять в парк, щоб покататися на атракціонах. У вихідні дні кількість відпочиваючих значно більше, аніж в робочі. У парку є декілька кас для продажу квитків. Відпочиваючий спочатку купує квитки в касі (причому стає в ту касу, де черга менша), після чого відправляється на атракціон і чекає своєї черги, оскільки атракціон може обслуговувати обмежену

кількість людей. Існує вірогідність того, що відпочиваючий захоче покататися більш ніж на одному атракціоні та, можливо, декілька разів.

24. Моделювання дороги студента, варіант Student'sWay

Студенти витрачає якийсь час на дорогу до університету. Зайшовши в університет, він ст в чергу перед турнікетом, далі йде і стає в чергу до ліфта, щоб піднятися на верхній поверх. Ліфт може вміщати обмежену кількість людей.

25. Моделювання обслуговування ветеранів, варіант Veteran

9 травня 2011 року ветеранам, що зареєстровані у місті Києві, після урочистого мітингу на Майдані Незалежності видавали продуктові пакети з гречкою. Спочатку ветеран мав показати посвідчення і отримати талон, потім отримував пакет. Внаслідок поганої організації цього процесу у пунктах видачі талонів та пакетів створилися великі черги. Деяким ветеранам довелося чекати пакету майже годину. Президент, дізнавшись про це, наказав у наступному році підготуватися до видачі пакетів краще і створити імітаційну модель, яка дозволить промоделювати цей процес и визначити потрібну кількість обслуговуючого персоналу в залежності від кількості ветеранів.

Можливо у наступному році ветеранів у автобусах, після отримання пакетів будуть відвозити до палацу «Україна» на концерт. Це теж треба передбачити у моделі.

26.Моделювання ковзанки, варіант Skating

На ковзанку у міському парку культури та відпочинку приходить молодь, щоб покататися на ковзанах. У святкові та вихідні дні бажаючих багато і тому створюються черги. Це пов'язано з тим, що кількість ковзанів і кількість місць у роздягальні обмежена, а на перевзування потрібен час, до того ж обслуговування здійснює тільки один працівник. Люди, що закінчили катання, обслуговуються в першу чергу. Після ковзанки, відвідувачі можуть прийняти душ та висушити голову феном, щоб не застудитися. Кількість душових кабінок та крісел з феном обмежена.

27. Моделювання кафе на пляжі, варіант BeachCafee

На пляжі є кафе. Офіціанти чекають, чи не потребує хтось обслуговування. Якщо виклик є, офіціант біжить до відпочиваючого і приймає заказ, після цього іде до буфету і замовляє страву. Одержавши готову страву несе її відпочиваючому. Побачивши, що відпочиваючий закінчив із трапезою, прибирає посуд і розраховується.

28. Розчистка завалів, варіант Rebuilding

На місці зруйнованих цегляних будинків працює екскаватор, який вантажить каміння у самоскиди. Кіміння з руйнувань використовують для

створення твердого покриття на грунтовій дорозі. Кожну порцію каміння послідовно, спеціальнгим чином обробляєть на дорозі, тому самоскидам іноді доводиться чекати.

29. Моделювання походу за покупками, варіант Shopping

У мікрорайоні міста, де є аптеки, супермаркети і банкомати, проживає деяка кількість родин. Періодично хтось із родини відправляється за покупками. Існує вірогідність того, що перш ніж здійснювати покупку, доводиться зняти гроші в одному з банкоматів. Покупець стає до того банкомату, де черга менша. Якщо необхідно придбати медикаменти, то покупець заходить в аптеку, причому, якщо в аптеці немає всіх потрібних медикаментів, то він намагається придбати їх в інших аптеках.

Після цього покупець йде до супермаркету. Якщо в супермаркеті немає в наявності деяких потрібних продуктів, то покупець намагається придбати їх в інших супермаркетах.

Якщо в аптеці або супермаркеті є декілька кас, то покупець стає в касу, де черга менша.

30.Моделювання іспиту, варіант Examen

Студентська група здає іспит. Кількість студентів, що мають здавати іспит визначається деканатом. Перших кілька студентів приходять до початку іспиту разом з викладачем. Інші з'являються через деякі проміжки часу. Якщо всі присутні здали іспит, а хтось ще не прийшов, іспит закінчується, і відсутнім виставляються двійки. Студент витрачає якийсь час на підготовку до відповіді, і потім чекає, поки викладач зможе його вислухати. З деякою ймовірністю викладач може поставити студентові двійку. Коли черговий студент одержує оцінку й виходить із аудиторії, на його місце входить наступний студент. Студенти, що одержали двійки, потрапляють до деканату, який знову організує екзамен за тими же правилами, і, можливо ще втретє, якщо не всі здали з другого разу. Ті, що одержали 3 двійки відраховуються.

31.Моделювання роботи оптового магазина, варіант OptoParfum

В оптовому магазині парфумів використовується нова процедура обслуговування. Клієнти приходять у магазин і за каталогом замовляють у одного із клерків товари, які вони хочуть придбати. Час, необхідний для оформлення замовлення випадковий. Якщо до клерка черга клієнтів, він набирає замовлення відразу від декількох клієнтів, але не більше 3. Після цього клерк іде на склад, вибирає й привозить необхідні товари. Час, що витрачається на виконання замовлення випадковий й, крім того, залежить від кількості замовлень. Після повернення зі складу клерк розраховується з кожним клієнтом окремо, відповідно до черги, витрачаючи на це якийсь час. Інколи товар доводиться привозити з іншого складу, у цьому випадку клієнт має чекати, поки товар привезуть. Трапляється, що клерк помиляється у виборі товару, у цьому випадку він не виконує розрахунок з покупцем а повертається на склад і

приносить потрібний товар, після чого виконується розрахунок. Розрахувавшись із усіма своїми клієнтами, клерк починає обслуговувати нових клієнтів.

32.Моделювання роботи пожежної частини, варіант Diablo

У місті Дьябло часто трапляються пожежі. Сигнал про пожежу надходить у пожежну частину й на гасіння пожежі виїжджає кілька пожежних машин. Якщо всі машини виїдуть на пожежу, то може виявитися, що іншу пожежу не буде кому гасити. Машини виїжджають, якщо вони є й готові до гасіння. Інформація про те, що пожежа закінчилася, не надходить, тому машина їде на пожежу у будь якому разі. Прибувши на пожежу, пожежні починають гасити її, у результаті чого час горіння й, відповідно, збитки від пожежі скорочуються. Але якщо машин мало, те засобів пожежегасіння може не вистачити, і тоді виявиться, що поїздка на пожежу була марною. Якщо пожежа вже догоріла, то машина відразу ж їде назад. Після повернення в пожежну частину машина проходить техобслуговування, після чого знову готова до гасіння.

33. Моделювання кільцевої конвеєрної лінії, варіант Circle

Досліджувана система складається з 3-х обробних пристроїв й одного завантажувального, розташованих навколо кільцевого конвеєра. На конвеєрі розташовані піддони для деталей. Конвеєр робить миттєві переміщення на відстань між сусідніми пристроями через рівні проміжки часу, у результаті чого піддон від одного пристрою переміщається до наступного. Завантажувальний пристрій через випадкові проміжки часу завантажує деталі в піддон, що перебуває перед ним. Обробний пристрій бере з піддона, що перед ним, деталі для обробки. Закінчивши обробку деталі, пристрій бере з піддона наступну. Якщо піддон порожній - пристрій переходить у режим очікування.

34. Моделювання проведення тесту, варіант TestStudents

Студентська група здає тест на комп'ютерах. В аудиторії обмежена кількість комп'ютерів, тому до початку тесту приходить стільки студентів, скільки є комп'ютерів. Інші з'являються пізніше, через деякі проміжки часу. Сидячи за комп'ютером, студент відповідає на питання. Відповівши на всі питання, він отримує оцінку й іде. Вільне місце займає інший студент із черги. Якщо всі комп'ютери звільнилися, а хтось ще не прийшов, то прийом тесту закінчується, а ті хто спізнився одержують двійки. Для студентів, що одержали незадовільні оцінки, організується повторний прийом тесту, і так доти, поки всі студенти не здадуть його.

35.Моделювання командної розробки програмного забезпечення, варіант TeamSuccess

Команди розробників приймають замовлення на розробку програмного забезпечення, причому замовники віддають перевагу тій команді, у якої нараховується найменша кількість «провалених» проектів. Час розробки проекту замовник встановлює сам. Кожна команда складається з менеджера, декількох розробників і тестерів, і може одночасно вести роботу над деякою кількістю проектів (залежно від розміру команди).

Після прийняття замовлення менеджер займається проектуванням, декомпозицією задачі і визначає кількість ітерацій розробки. Перед кожною ітерацією менеджер розподіляє задачі між розробниками. В рамках ітерації розробник працює над своєю задачею, після чого віддає результат у відділ тестування. Якщо модуль розробника не проходить тестування декілька разів, то розробника звільняють, а його задача призначається на іншого розробника. Після того, як всі задачі ітерації пройшли тестування, менеджер проводить збірку проекту і відправляє її тестерам: якщо збірка не пройшла тестування, то менеджер повертає задачі попередньої ітерації розробникам на доопрацювання, інакше команда переходить до наступної ітерації або, якщо ітерація була останньою, надає створене програмне забезпечення замовнику.

Існує вірогідність того, що хтось із членів команди захворіє: якщо захворіє розробник або тестер, то на час хвороби його задачі виконує інший член команди з того ж відділу; якщо захворіє менеджер проекту, то на час його хвороби команда вибирає нового менеджера з відділу розробників.

Крім того, перед кожною черговою ітерацією менеджер проводить оцінку затраченого часу, і якщо команда не вкладається в установлений термін, то розробка проекту зупиняється, проект вважається «проваленим», а замовник намагається оформити договір з іншою командою.

36. Моделювання командної розробки програмного забезпечення, варіант TeamFinance

Команди розробників приймають замовлення на розробку програмного забезпечення, причому замовники віддають перевагу тій команді, прибуток якої є найбільшим. Кожна команда складається з менеджера, декількох розробників і тестерів і приймає нове замовлення тільки після завершення роботи над попереднім.

Після надходження замовлення менеджер робить оцінку вартості та часу, необхідного для розробки проекту залежно від трудомісткості замовлення та кількості розробників. Існує вірогідність того, що замовника не влаштує запропонована ціна або час, тоді він намагатиметься оформити договір з іншими командами розробників.

Якщо договір із замовником оформлений, то менеджер займається проектуванням і декомпозицією задачі, і створює деяку кількість (залежить від трудомісткості проекту) підзадач для розробників.

Після завершення роботи над черговою задачею, розробник відправляє результат менеджеру проекту і приймається за наступну задачу. Менеджер спочатку самостійно проводить тестування задач, що поступили від розробників, і якщо помилок не виявлено, то відправляє тестерам для подальшої перевірки. У разі виявлення помилки менеджером або тестерами, задача повертається розробнику на доопрацювання. Після того, як всі задачі пройшли повне тестування, менеджер проводить збірку проекту і знову відправляє її на тестування. Якщо збірка тестування не пройшла, то менеджер знову повертає задачі розробникам на доопрацювання. І так до тих пір, поки кінцева збірка не пройде тестування.

Як тільки процес розробки закінчується, замовник розплачується з командою, причому якщо час розробки перевищив запланований, то замовник знижує оплату пропорційно затримці за часом.

37.Моделювання паралельних обчислень, варіант ForkBloks

У спеціалізоване обчислювальне середовище, що складається з деякої кількості обчислювальних пристроїв, які працюють паралельно, з мережі надходять задачі особливого типу. Алгоритм вирішення кожної задачі полягає у послідовному виконанні обчислювальних блоків, кожен з яких складається з деякої кількості гілок, які можна обчислювати паралельно. Виконання кожного з блоків можна починати тільки після обчислення усіх паралельних гілок попереднього.

Приблизні значення кількості гілок кожного блоку та часу виконання кожної гілки відомі, але вони можуть змінюватися під впливом параметрів задачі і таким чином є випадковими. Закони розподілення цих випадкових величин для кожного з блоків відомі.

2 МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ РОБОТИ

2.1 Аналіз системи що підлягає моделюванню

Перший крок аналізу системи, що підлягає моделюванню, передбачає виділення так званих абстракцій, на основі яких будуть створені класи, необхідні для побудови моделі. Поняття «абстракція» передбачає, що при розгляді якоїсь частини реальної системи ми беремо до уваги тільки ті її властивості, які мають значення для вирішення поставленого завдання. Інколи може бути і так, що абстракція не відповідає жодній частині реальної системи.

Від правильної декомпозиції реальної систем і представлення її у вигляді сукупності абстракцій, що пов'язані між собою, залежить трудомісткість створюваного програмного продукту, його якість та життєздатність. Іноді ці показники входять у протиріччя, у цьому випадку слід приймати компромісні рішення.

Абстракції реальної системі та їх поведінка мають відповідати реаліям системи. Це робить модель більш зрозумілою для користувача. Якщо ви моделюєте, наприклад, ресторан, то замовлення офіціанту має робити клієнт а не стілець.

Результатом першого кроку аналізу має бути перелік ключових абстракцій, та перелік завдань, що має вирішувати кожна з абстракцій відповідно до цілей моделювання.

Наступний крок аналізу полягає у більш детальному аналізі активних абстакцій системи. Активними абстракціями є такі що мають власну поведінку розподілену у часі. Зазвичай ця поведінка характеризується наявність затримок у часі, що імітують виконання якихось обов'язків абстракції, та (або) наявністю зупинок діяльності до виконання деяких умов.

Для цих абстракції доведеться створювати класи, тому перш за все слід визначити перелік інформації, необхідної для функціонування абстракції, а також представити правила дії у вигляді схеми алгоритму, або діаграми діяльності.

Наступним кроком аналізу має бути аналіз можливостей фреймворку Simulation для реалізації класів, що будуть відтворювати абстракції системи у проекті. Тут слід мати на увазі, що до фреймворку майже кожен рік вносяться зміни та доповнення, які враховують досвід його застосування. Версії різних років несумісні. Тому у роботі слід використовувати версію Simulation, яка використовувалася при виконанні лабораторних робіт.

Результатом цього етапу має бути перелік класів та інтерфейсів, що мають бути представлені у вигляді таблиці.

Слід відзначити, що результат роботи над проектом суттєво залежіть від якості виконання етапу аналізу. Все, що буде робитися далі, це суто технічна робота.

2.2 Огляд можливостей фреймворку Simulation

Фреймворк Simulation Java було створено зусиллями викладачів та студентів кафедри ІКС для спрощення реалізації нестандартних імітаційних моделей. Цей фреймворк є альтернативою до розповсюджених стандартних пакетів та мов імітаційного моделювання, на кшталт GPSS. Особливість згаданих засобів полягає у тому, що там використовується структурний підхід до побудови моделі. При цьому трансакція розглядається як пасивний елемент, що переміщується через накопичувачі різного роду від одного активного елементу до іншого. Популярність такої концепції пояснюється тим, що вона досить легко трансформується у графічне зображення. Модель являє собою схему, що складається з накопичувачів та обслуговуючих пристроїв. Типовим прикладом такої схеми є мережа Петрі, де активними елементами є переходи, накопичувачами – позиції, а трансакціями – фішки. У цій схемі фішка не є активним елементом, вона сама не вирішує, яким маршрутом переміщуватися і як це робити у часі, усе вирішує активний елемент – перехід. Але у реальному житті трапляються і інші ситуації, наприклад, лікар, що приймає хворих, є обслуговуючим пристроєм, але вийшовши на перерву до буфету він перетворюється у трансакцію, що чекає на обслуговування. У структурному підході такі ситуації не передбачені, але якщо описувати поведінку лікаря за допомогою алгоритму, то ніяких проблем не виникає. Тому у фреймворці Simulation для побудови моделі використовується алгоритмічний підхід орієнтований на об’єкти. Фреймворк Simulation Java допомагає створювати моделі на мові Java. До складу фреймворку входять повністю реалізовані класи і візуальні компоненти, а також абстрактні класи та інтерфейси, що значно спрощують побудову моделі.

Використання фреймворку у навчальному процесі на кафедрі спрямоване ще і на закріплення навичок об’єктно-орієнтованого програмування на мові Java.

Слід зазначити, що фреймворк постійно змінюється і розширюється. При цьому мета сумісності з попередніми версіями не ставиться, навіть навпаки. Це робиться з метою уникнення плагіату під час розробки проектів студентами. Тому для виконання навчальних завдань слід використовувати версію фреймворку, що рекомендована викладачем. Нижче наводиться опис можливостей фреймворку версії Simulation2022.

2.2.1 Засоби для створення активних об’єктів моделі

2.2.1.1 Клас Actor

Базою для створення активних компонент моделі є клас Actor. Це абстрактний клас, що потребує реалізації методу **protected void rule().** У цьому методі має бути визначена поведінка (правила дії) активного елементу. При цьому можна використовувати такі спеціальні методи:

protected void holdForTime(double time) – забезпечує затримку виконання правил дії на заданий проміжок часу;

replaceActivateTimeBy(double newActivateTime) дозволяє змінити час активізації після затримки у часі можна за допомогою методу;

protected void waitForCondition(BooleanSupplier condition, String textForProtocol) – забезпечує затримку виконання правил до виконання заданої умови;

protected void waitForConditionOrHoldForTime(BooleanSupplier condition, String textForProtocol, double time) забезпечує затримку виконання правил або до виконання заданої умови, або на заданий проміжок часу.

protected void holdForTimeOrWaitForCondition (double time, BooleanSupplier condition, String textForProtocol) забезпечує затримку виконання правил на заданий проміжок часу або до виконання заданої умови.

Умови, виконання яких чекає об’єкт у останніх методах, визначаються методом public boolean getAsBoolean(), реалізації якого вимагає функціональний інтерфейс BooleanSupplier. Умови можна представити і у вигляді лямбда функцій. Параметр методів textForProtocol використовується для ідентифікайії події у протоколі.

Примусово переривати чекання можна за допомогою методу terminateWaiting().

Об'єкти цього класу здатні накопичувати інформацію про час чекання виконання умови у об’єктах класу Histo, якщо такий об’єкт передано за допомогою методу setWatingTimeHisto(Histo).

Для того, щоб активний об’єкт почав виконувати свої правила дії, його слід передати об’єкту класу Dispatcher за допомогою методу addStartingActor(Actor) для екземплярів класу Dispatcher.

Об’єкт класу Dispatcher забезпечує псевдопаралельне виконання правил дії активних об’єктів моделі. Цей об’єкт не слід створювати у моделі, його треба створити зовні. Це пов’язано з тим, що інколи необхідно, щоб декілька моделей працювали паралельно, під керівництвом одного диспетчера.

2.2.1.2 Клас MultiActor

Цей клас використовується для створення груп однакових об'єктів (бригад), що моделюють багатоканальну обробку в СМО. При такому обслуговуванні група паралельно працюючих приладів працює з однією чергою. Для налаштування об’єкту цього класу потрібно визначити значення двох атрибутів:

nClone – визначає кількість копій.

original – містить посилання на зразок, з якого будуть зроблені копії.

Для цього використовується конструктор з відповідними параметрами, або методи setNumberOfClones(int) та setOriginal(Actor).

Клас успадковує клас Actor і у ньому визначено метод rule(). Під час виконання правил дії цього об’єкта створюються копії оригіналу шляхом клонування і ці копії одразу записуються у стартовий список диспетчера.

Для клонування використовується метод clone() класу Actor. Тут слід зазначити, що операція клонування в Java є операцією поверхневого копіювання (shallowCopy). При такому копіюванні створюється копія області

пам'яті, що містить інформацію про об'єкт. Внаслідок цього ми отримуємо копії значень для примітивних типів і типу String, а для решти класів отримуємо не копії об’єктів, а копії посилань на об'єкти. Таким чином, поля всіх клонованих об'єктів посилаються на ті самі об'єкти, що визначаються зразком.

При клонуванні акторів, що забезпечують багатоканальну обробку, такий спосіб клонування нас майже влаштовує. Всі клони будуть посилатися на ту саму чергу, на той самий генератор випадкових чисел, у всіх буде той самий диспетчер. Імена клонів можна робити різними. Єдиний недолік поверхневого копіюванні в тім, що всі клони будуть мати той самий семафор (об'єкт класу BooleanSemaphore). Для того, щоб усунути цей недолік, у класі Actor перевизначений базовий метод clone(), у якому для кожного клону створюється власний об'єкт класу BooleanSemaphore.

Цей метод слід перевизначити і у спадкоємців класу Actor, якщо вони мають посилання на об’єкти, що мають бути унікальними для кожної копії.

2.2.2 Засоби для створення черг та накопичувачів

2.2.2.1 Клас QueueForTransactions

Клас використовується для моделювання черг у системах масового обслуговування. Для збереження об’єктів, що потрапляють до черги, клас має поле dequeue типу ArrayDeque, але робота з цим об’єктом забезпечується через методи класу QueueForTransactions.

Об'єкти цього класу можуть відображати свій стан у вигляді діаграми, використовуючи об'єкти класу paint.Painter, накопичувати інформацію про поточні розміри черги у об’єкті класу DiscretHisto, а також виводити інформацію про зміну розмірів до протоколу роботи моделі.

Для виконання цих завдань черга має мати посилання на диспетчера, під керівництвом якого працює імітаційна модель, щоб отримувати значення поточного часу.

Окрім того, для відображення стану якоїсь черги на діаграмі, цій черзі слід передати посилання на об'єкт класу paint.Painter, за допомогою методу setPainter(Painter). А цей об'єкт класу paint.Painter, у свою чергу, має мати посилання на якусь діаграму.

А для накопичення статистичних даних про розмір черги їй слід передати посилання на об'єкт класу DiscretHisto за допомогою методу setDiscretHisto(DiscretHisto).

Для об’єктів даного класу використовується також поняття максимального розміру, вище якого черга заповнена бути не може. У випадку спроби додавання об'єкта до заповненої черги формується подія QueueOverflowEvent, але об'єкт не додається. Доступ до неприйнятої заявки можна отримати через об’єкт класу QueueOverflowEvent, створивши слухача даної події.

При повторному використанні, перед початком роботи моделі черга повинна бути проініціалізована. Для цього використовується метод init(). Під час ініціалізації черга очищається, а об'єкт «painter», якщо він існує, і існує

діаграма, на якій буде зображуватися черга, переводиться у положення з координатами 0,0.

Зміна розмірів черги відбувається за допомогою методів addLast(Object), remove(Object), removeFirst().

Особливість методу addLast(Object) полягає в тому, що в ньому аналізується ступінь заповнення черги й, у випадку неможливості її подальшого збільшення, формується подія QueueOverflowEvent.

Особливість методів додавання й видалення об'єктів полягає в тому, що перед зміною черги та після її зміни викликається один з наступних методів – beforeAdd(), beforeRemove(), afterAdd(), afterRemove(), залежно від виконаної операції.

Ці методи виводять до протоколу диспетчера інформацію про новий розмір черги, а також забезпечують відображення стану черги на діаграмі, та накопичення інформації у гістограмі, якщо необхідні для цього об’єкти визначені. Методи мають специфікатор доступу protected і тому можуть бути довизначені у спадкоємців класу.

Обєкти цього класу здатні також накопичувати інформацію про середнє значення черги на деякому інтервалі. Для цього використовуються методи resetAccum() та getAccumAverage(). Ці можливості, зокрема, використовуються під час дослідження перехіднх процесів у чергах.

2.2.2.2 Клас Store

Цей клас схожий на попередній. Різниця полягає у тому, що об’єкти цього класу накопичують значення типу double у полі size. Так само як і у попередньому класі можливо відображення поточного значення size на діаграмі і накопичення цих значень у гістограмі, тільки гістограма має бути типа Histo..

2.2.3 Засоби для збирання та обробки статистичної інформації

Класи Histo і DiscretHisto дозволяють накопичувати інформацію про значення випадкових безперервних і дискретних величин, і подавати інформацію про ці величини у вигляді гістограм. Гістограма може бути представлена у вигляді стовпчастої діаграми або у вигляді таблиці.

Для зручності деякі методи гістограм винесено у інтерфейс IHisto:

– init(). Цей метод без параметрів можна використовувати для ініціалізації об'єктів обох класів. У цьому випадку межі гістограми формуються автоматично.

– add(double). Цей метод додає у гістограму число, що передається до методу в якості параметру.

– addFrequencyForValue(double, double). В якості першого параметра задається вага переданого випадкового числа, а другим передається саме випадкове число. У простих випадках вага може дорівнювати 1. Саме так реалізовано метод add(double).

– showRelFrec(Diagram). Використовується для виведення результатів у вигляді стовпчастої діаграми. До методу як параметр передається посилання на

об'єкт класу Diagram, де буде відображатися діаграма. Можлива й інша модифікація методу, з більшим числом параметрів і, відповідно, більшими можливостями.

– getAverage(). Повертає середнє значення для накопичених даних.

– toString(). Використовується для виведення результатів обробки накопичених даних у вигляді тексту з таблицею відносних частот.

Нижче наведені ще деякі методи, що не увійшли до єдиного інтерфейсу.

Для ініціалізації, додатково можна використовувати такі методи:

– initFromTo(int, int). Цей метод використовують для об'єктів класу DiscretHisto. Два цілих числа, що передаються в якості параметрів, визначають межі гістограми.

– initFromTo(double, double, int). Цей метод використовують для об'єктів класу Histo. Два дійсних числа, що першими передаються в якості параметрів, визначають межі гістограми, а ціле число визначає кількість інтервалів.

2.2.4 Засоби для генерації випадкових величин

2.2.4.1 Клас ChooseRandom

Цей клас визначає візуальний компонент, що забезпечує вибір і налаштування потрібного генератора випадкових чисел. Елементом, які постійно знаходяться на формі є панель з кнопкою і полем для виведення інформації про вибраний законі розподілу. Вигляд цієї панелі представлений на рисунку 2.1.

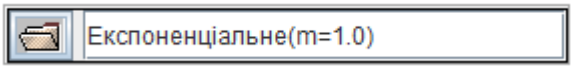

Рисунок 2.1 – Вигляд компонента ChooseRandom

При натисканні на кнопку відкривається комбобокс з переліком можливих розподілів, рисунку 2.2.

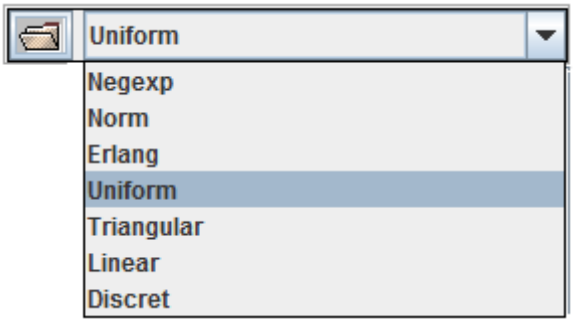

Рисунок 2.2 – Перелік можливих розподілів компонента ChooseRandom

Після вибору необхідного розподілу з’являється діалогове вікно для налаштувань параметрів вибраного розподілу, рисунок 2.3.

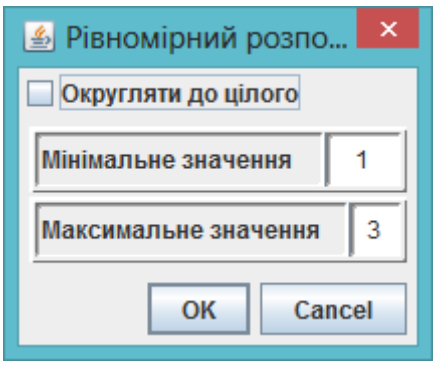

Рисунок 2.3 – Діалогове вікно для налаштування рівномірного розподілу компоненту ChooseRandom

Даний компонент реалізує інтерфейс Randomable. Основні методи цього інтерфейсу наступні:

– next() - повертає наступне випадкове число;

– probability(double) - повертає значення інтегральної функції розподілу.

2.2.5 Компоненти для створення інтерфейсу користувача та відображення результатів моделювання

2.2.5.1 Клас ChooseData

Об’єкти цього класу доцільно використовувати для введення числових параметрів компонент моделі. Цей клас успадковує клас JTextField. Поряд із усіма можливостями стандартного текстового компоненту користувач класу ChooseData отримує додаткові послуги. Вигляд компоненту представлено на рисунку 2.4.

Рисунок 2.4 – Вигляд компоненту класу ChooseData

Метод setTitle(String) дозволяє позначити призначення компоненту.

Методи getInt() та getDouble() дозволяють отримати числа відповідних типів.

Методи setInt(int) та setDouble(double) дозволяють відображати числа відповідних типів.

Методи getIntArray() та getDoubleArray() дозволяють отримати масиви відповідних типів.

2.2.5.2 Класи для графічного відображення результатів моделювання

Для графічного відображення результатів моделювання використовуються класи Diagram і Painter.

Клас Diagram представляє візуальний компонент для відображення графіків і діаграм. Вид компоненту класу Diagram показано на рисунку 2.5.

Компонент дозволяє настроювати мінімальні і максимальні значення по осях координат, заголовок діаграми. Можна настроювати координатну сітку. Розміри компоненту можна змінювати.

Нижче наведено деякі методи з класу Diagram.

Рисунок 2.5 – Вигляд компоненту класу Diagram

Методи setGridByX(int), setGridByY(int) використовуються для налаштування масштабної сітки на діаграмі.

Методи setHorizontalMinText(String), setHorizontalMaxText(String), setVerticalMinText(String), setVerticalMaxText(String) використовуються для налаштування діапазону діаграми по горизонтальній та вертикальній висях.

Метод setTitle(String) використовуються для налаштування заголовку діаграми.

Метод clear() очищає діаграму.

Зображення геометричних фігур на діаграмі виконуються за допомогою об'єкту класу Painter, що входить до складу об'єктів класу Diagram. З його допомогою можна зображувати лінії, прямокутники, овали. Колір ліній налаштовується. Особливість цих об'єктів полягає в тому, що після зображення лінії, вони пам'ятають своє становище на діаграмі, що спрощує зображення графіків

Об'єкти класу Painter можуть існувати і незалежно від діаграми, а мати тільки посилання на неї. Завдяки цьому, кілька об'єктів класу Painter можуть використовувати одну й ту ж діаграму, що дозволяє зображати на одній діаграмі кілька графіків.

Метод placeToXY(float, float) використовуються для переміщення пера у точку із заданими відносно діаграми реальними координатами.

Метод drawToXY(float, float) використовуються для проведення лінії у задану точку на діаграмі.

Метод drawDependency(…) можна використовувати для виведення графіків. Як параметри в останній метод передається два масиви, які задають координати точок графіка, колір графіка і параметр логічного типу, який вказує, чи слід налаштовувати діаграму під дані графіка.

Метод drawBarsDiagram(…) цього класу можна використовувати для виведення стовпчастих діаграм. Як параметри в метод передається масив границь інтервалів по горизонтальній вісі, масив значень по вертикалі, ширина стовпчика в долях ширини інтервалу, зсув стовпчика від лівої межі проміжку і параметр логічного типу, який вказує, чи потрібна зміна налаштування

діаграми під дані.

Метод drawNeedleDiagram(…) цього класу можна використовувати для відображення голчастої діаграми. Перелік параметрів цього методу майже такий самий як і у попереднього.

Перелічені вище методи можна викликати і через об’єкти класу Diagram.

На діаграмах також можна відображати гістограми. Але ці методи викликаються для об’єктів-гістограм, а посилання на діаграму у ці методи передається як параметр.

2.2.6 Компоненти, що спрощують проведення експериментів та відображення результатів моделювання

Наведені нижче компоненти можна використовувати як елементи інтерфейсу користувача у застосуваннях для імітаційного моделювання.

2.2.6.1 Компонент StatisticsManager

Цей компонент забезпечує запуск моделі у режимі отримання статистичних даних та відображає отримані статистичні дані. Зовнішній вигляд компоненту наведено на рисунку 2.6.

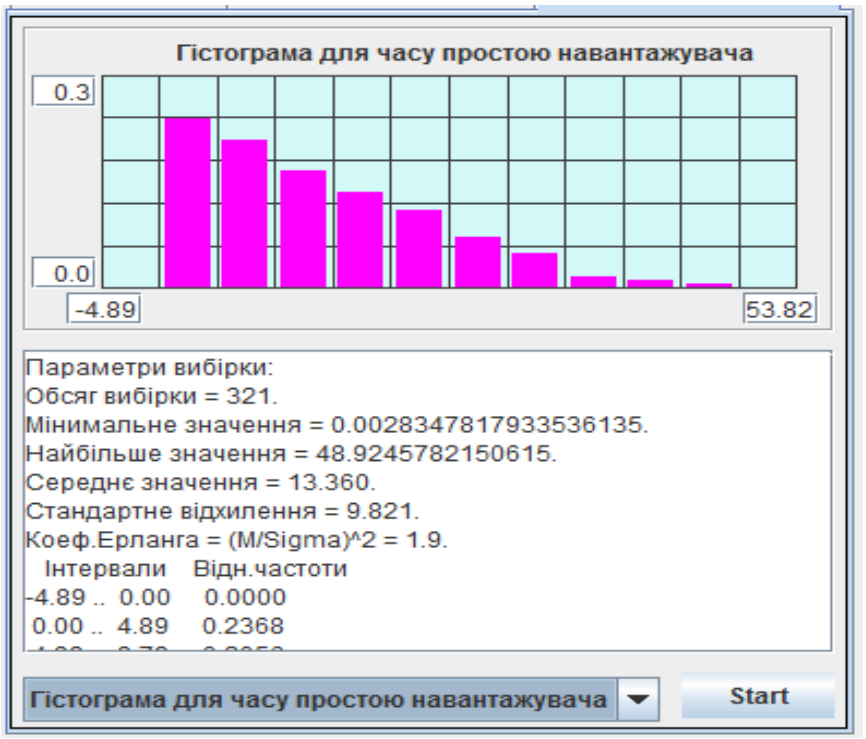

Рисунок 2.6 – Вигляд компоненту класу StatisticsManager

Необхідною умовою для роботи компонента є передача йому посилання на фабрику моделей, що реалізує інтерфейс IModelFactory. Єдиний метод цього інтерфейсу createModel(Dispatcher) забезпечує створення моделі і передачу їй посилання на диспетчера. Зв’язок компонента ExperimentManager з фабрикою моделей налаштовується через метод setFactory(IModelFactory).

Виходячи з того, що інтерфейс IModelFactory містить тільки один метод, то створити фабрику можна за допомогою лямбда функції, у якій реалізується

звернення до конструктора моделі:

statisticsManager.setFactory((d)-> new Model(d, this));

У цьому прикладі this – це посилання на візуальну частину, де розташований менеджер.

Отримавши модель, компонент ExperimentManager приводить її до типу widgets.stat.IStatisticsable. Це інтерфейс, через який компонент StatisticsManager буде працювати моделлю. Тобто модель має реалізовувати такий інтерфейс.

Цей інтерфейс передбачає реалізацію двох методів.

Метод void initForStatistics() дозволяє виконати підготовчі дії перед початком роботи моделі, якщо це потрібно.

Метод Map<String, IHisto> getStatistics() спрацьовує після завершення роботи моделі і повертає колекцію типу Map, в якої кожен елемент у якості ключа містить текст, що ідентифікує статистичні дані, а значенням має гістограму, що містить ці дані.

2.2.6.2 Компонент ExperimentManager

Цей компонент забезпечує запуск моделі у режимі проведення однофакторних однорівневих та багаторівневих експериментів, відображає результати експериментів та дозволяє проводити дисперсійний та регресійний аналіз отриманих даних. Зовнішній вигляд компоненту наведено на рисунку 2.7.

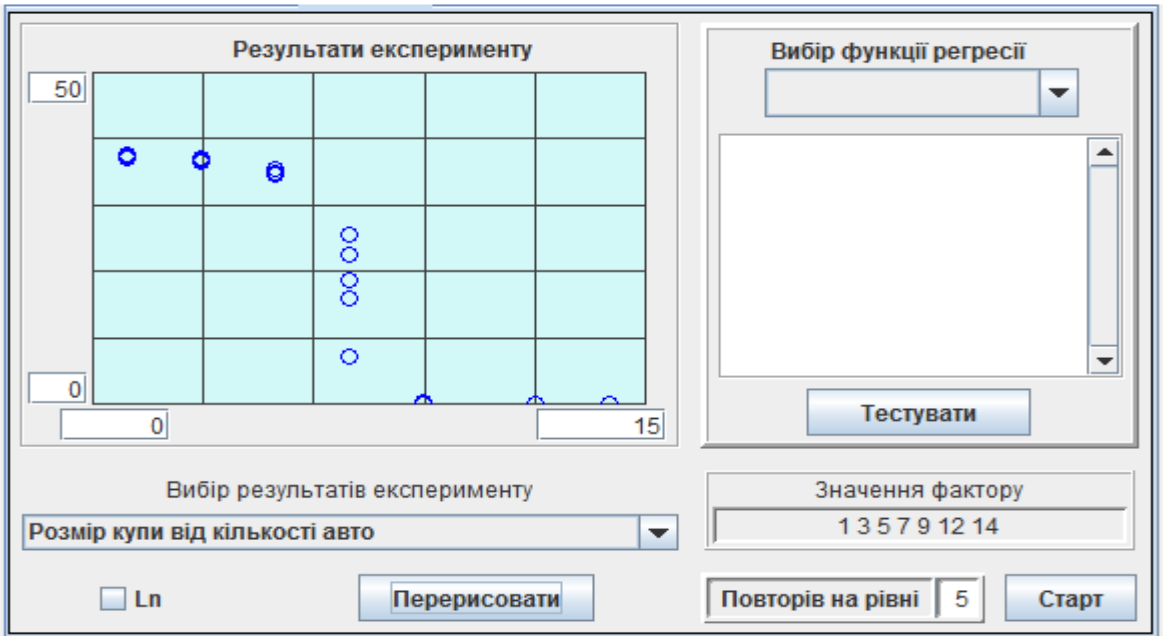

Рисунок 2.7 – Вигляд компоненту класу ExperimentManager

Компонент передбачає технологію, відповідно до якої модель створюється, ініціалізується та запускається самим компонентом.

Необхідною умовою для роботи компонента є передача йому посилання на фабрику моделей, що реалізує інтерфейс IModelFactory. Єдиний метод цього інтерфейсу createModel(Dispatcher) забезпечує створення моделі і передачу їй

посилання на диспетчера. Зв’язок компонента ExperimentManager з фабрикою моделей налаштовується через метод setFactory(IModelFactory).

Отримавши модель, компонент ExperimentManager приводить її до типу widgets.experiments.IExperimentable. Це інтерфейс, через який компонент ExperimentManager буде працювати моделлю. Тобто модель має реалізовувати такий інтерфейс.

Метод void initForExperiment(double) цього інтерфейсу викликається перед кожним запуском моделі і має забезпечити підготовку моделі до однократного запуску. В якості параметра у метод передається значення фактору, вплив якого вивчається.

Метод Map<String, Double> getResultOfExperiment() використовується для отримання результатів експерименту після його закінчення. Кожен елемент колекції, що повертає цей метод, в якості ключа містить текст, що ідентифікує результат експерименту, а значенням є сам результат. Таким чином метод може повертати будь яку кількість відгуків на задане значення фактору.

На підставі переліку ключових значень елементів отриманої колекції створюється модель для компоненту комбобокс, який дозволяє вибрати потрібну залежність.

Значення фактору, для яких потрібно провести експерименти, задаються у вигляді рядка символів в полі «Значення фактору:». Числові значення факторів повинні бути розділені стандартними розділовими знаками.

Кількість експериментів, що мають бути проведені для кожного значення фактору, задається у полі «Повторів на рівні:».

Кнопка «Старт» викликає метод buttonStartClick(), який забезпечує проведення заданої кількості експериментів на кожному рівні. Під час проведення експериментів на діаграмі відображаються точки, які відповідають отриманим результатам, що забезпечує динамічну індикацію перебігу експериментів.

Результати експериментів можуть бути логарифмовані. Для цього використовується перемикач «Ln».

До складу компоненту ExperimentManager включено також компонент класу RegresAnaliser за допомогою якого можна підібрати функцію регресії для отриманих результатів і протестувати її на адекватність

Компонент може працювати або у режимі однорівневого експерименту, або у режимі багаторівневого експерименту.

У першому випадку компонент видає інформацію про довірчий інтервал для результатів експерименту і графічно відображає його розміри на фоні експериментальних даних.

У режимі багаторівневого експерименту, компонент дозволяє вибрати функцію регресії і знайти її параметри і отримати графічне відображення. Окрім того надається інформація про результати перевірки на однорідність дисперсій, вплив фактору та адекватність функції регресії.

Перелік функцій регресії можна розширити. Для цього необхідно створити клас, що успадковує клас RegresTesters, і реалізувати у ньому абстрактні методи суперкласу. Завдання спрощується, якщо нова функція

регресії має вигляд, що передбачений класами Regres1 або Regres2. У цьому випадку достатньо успадкувати один з цих класів та реалізувати у ньому метод fi1(double), або методи fi1(double) та fi2(double).

Створивши клас, що відповідає за нову функцію регресії, слід створити об’єкт цього класу і передати його компоненту RegresAnaliser за допомогою методу addFunction(RegresTesters).

2.2.6.3 Компонент TransProcessManager

Цей компонент забезпечує дослідження перехідних процесів у чергах. Зовнішній вигляд компоненту наведено на рисунку 2.8.

Інформація про перехідний процес формується шляхом усереднення розміру черг за часом та по реалізаціям для великої кількості паралельно працюючих моделей.

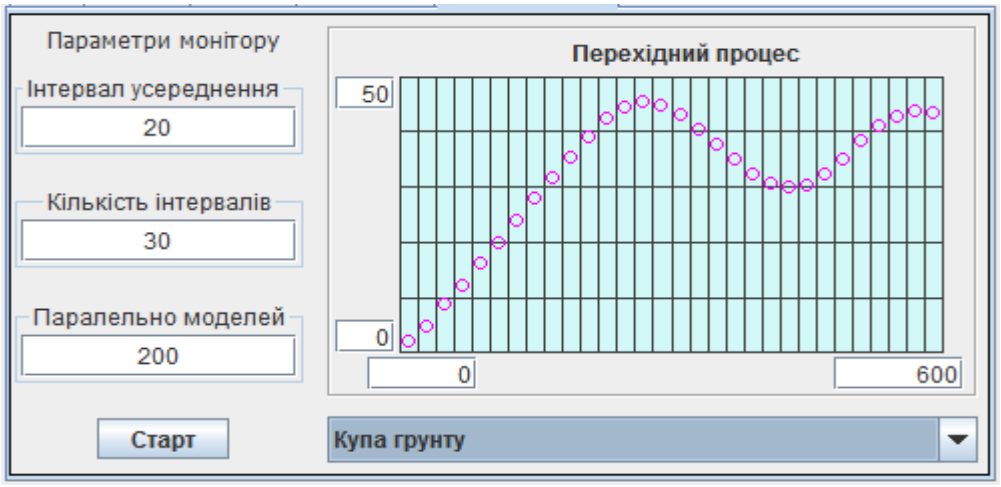

Рисунок 2.8 – Вигляд компоненту класу TransProcessManager

Так само як у випадку попереднього компоненту тут використовується технологія, відповідно до якої моделі створюються, ініціалізуються та запускаються самим компонентом.

Необхідною умовою для роботи компонента є передача йому посилання на фабрику моделей, що реалізує інтерфейс IModelFactory. Зв’язок компонента ExperimentManager з фабрикою моделей налаштовується через метод setFactory(IModelFactory).

Створивши модель, компонент TransProcessManager приводить її до типу widgets.trans.ITransProcesable. Це інтерфейс, через який компонент TransProcessManager буде працювати моделлю і модель має реалізовувати такий інтерфейс. Інтерфейс ITransProcesable передбачає реалізацію двох методів.

Метод void initForTrans(double finishTime) використовується для ініціалізації компонентів моделі. Як параметр до методу передається тривалість моделювання.

Метод Map<String, ITransMonitoring> getMonitoringObjects() надає компонентові перелік черги, що досліджуються.

Кожен елемент колекції, що повертає цей метод, в якості ключа містить текст, що ідентифікує чергу, а значенням є сама черга. Таким чином компонент може працювати з будь якою кількістю черг.

На підставі переліку ключових значень елементів отриманої колекції результатів створюється модель для компоненту JComboBox, який дозволяє викликати для перегляду потрібний перехідний процес.

Візуальна частина компоненту, рисунок 2.8, забезпечує його налаштування, а також запуск процесу моделювання.

По натисканню кнопки “Старт” починається процес моделювання.

Поле “Інтервал усереднення” визначає довжину інтервалу накопичення інформації.

Поле “Кількість інтервалів” визначає кількість інтервалів накопичення.

У полі “Паралельно моделей” задається кількість паралельно працюючих систем.

2.3 Методика побудови моделі СМО

Сучасні технології проектування програмних систем рекомендують створювати програмний продукт, як поєднання декількох шарів. Для побудови застосування, що забезпечує моделювання СМО, можна рекомендувати використовувати такі шари:

- шар подання (presentation layer);
- шар моделі;
- шар компонентів.

Схематичне зображення проекту для моделювання у вигляді взаємопов’язаних шарів представлено на рисунку 2.9.

2.3.1 Шар подання

За звичай першим шаром програмного продукту є шар подання. У нашому випадку це не щось інше, як графічний інтерфейс користувача, що є посередником між користувачем і самою моделлю. Основні завдання цього шару такі:

- отримання від користувача налаштувань компонент моделі;
- надання доступу до компонент, з налаштуваннями користувача;
- забезпечення запуску моделі;
- надання засобів для динамічної індикації процесу моделювання;
- надання засобів для відображення результатів моделювання.

У випадку використання фреймворку Simulation для побудови моделі рекомендовано у шарі подання реалізувати такі завдання:

- створення диспетчера;
- створення моделі;
- створення моделі;
- передачу моделі посилання на диспетчера;
- передачу моделі посилання на інтерфейс користувача;
- відображення результатів моделювання, отриманих від моделі.

Рисунок 2.9 – Структура застосування для моделювання СМО

Шар подання реалізується візуальним класом з необхідним набором візуальних елементів.

Перед початком моделювання у класі має бути створений диспетчер, фабрика моделей та модель. Створення моделі перед кожним її запуском дещо уповільнює роботу програми, тому що заново створюються усі об’єкти, але це спрощує програмування.

Для створення моделі слід використовувати фабрику моделей, що має реалізувати інтерфейс IModelFactory. Інколи без використання фабрики можна обійтись, але це є обов’язковим, якщо у програмному додатку будуть використовуватися компоненти, що були описані у 2.2.6.1 – 2.2.6.3.

Відповідно до вимог інтерфейсу IModelFactory у фабриці має бути реалізований усього один метод – createModel(Dispatcher). Але, окрім того фабрика має ще передавати моделі посилання на візуальну частину, або інший об’єкт, з якого модель буде отримувати налаштування.

Для спрощення завдань, пов’язаних із створенням моделі, можна рекомендувати для створення фабрики використовувати лямбда функцію.

Для запуску моделі використовується метод диспетчера start().

Відображення результатів моделювання, отриманих від моделі після завершення її роботи, можна пов’язати із подією DispatcherFinishEvent, створивши відповідного слухача цієї події. Інший варіант розв’язання цієї задачі полягає у тому, щоб після запуску моделі створити потік виведення результатів моделювання і приєднати його до потоку диспетчера за допомогою методу join().

Взагалі, для побудови інтерфейсу користувача достатньо стандартних засобів, що надає Java, але використання компонентів фреймворку Simulation спрощує цю задачу.

На етапі проектування шару подання першочергову увагу слід приділити визначенню публічних методів класу (або інтерфейсу), що будуть надавати доступ до компонентів моделі, що містять налаштування користувача. Це дозволить доручити розробку класу одному з членів команди.

2.3.2 Шар моделі

Модель створює і містить у собі посилання на усі об’єкти, що моделюють складові частини досліджуваної системи і засоби для накопичення статистичної інформації, а також має налаштувати ці об’єкти відповідно до вимог користувача, і надавати доступ до результатів моделювання у шар подання.

Нижче наведено перелік основних завдань цього шару:

– створити усі необхідні на момент старту об’єкти моделі;
– створити засоби для накопичення статистичної інформації;
– передати об’єктам моделі посилання на модель та шар подання;
– завантажити «акторів» моделі до стартового списку диспетчера;
– надати публічний (або пакетний) доступ до компонент моделі;
– надавати шару подання доступ до результатів моделювання.

Шар моделі має декілька особливостей.

Перша особливість полягає у тому, що незважаючи на те, що модель нібито весь час працює, насправді це найбільш пасивний елемент. Модель - це перш за все сховище посилань на моделі складових частин системи. У моделі має бути також метод ініціалізації, що викликається перед запуском диспетчера.

Тобто методи моделі працюють тільки на початковому етапі моделювання та після його завершення, а в процесі моделювання виконуються методи складових її частин, що утворюють третій шар програмної системи.

Можна рекомендувати таку послідовність програмної реалізації моделі:

– визначити поля класу, що відповідають усім складовим моделі (акторам, чергам, гістограмам та іншим складовим) без створення відповідних об’єктів. Зважаючи на те, що тільки модель відповідає за створення цих об’єктів, доступ до всіх складових не слід робити публічним. Слід також передбачити поля для посилань на диспетчера та візуальну частину, хоча ці компоненти не створюються у моделі;

– визначити конструктор для моделі, через параметри якого передавати посилання на диспетчера та візуальну частину. Пустий конструктор краще не

визначати. Це унеможливить створення моделі без посилань на диспетчера та візуальну частину. У конструкторі окрім визначення посилань на диспетчера та візуальну частину можна завантажити усіх акторів до стартового списку диспетчера. Для цього доцільно реалізувати метод componentsToStart, у якому передати диспетчеру усіх акторів, що мають почати роботу після старту моделі, але для звернення до акторів слід використовувати методи get…();

– створити об’єкти для складових частин моделі. Цю частину класу доцільно реалізувати за методикою відкладеного створення об’єктів. Відповідно до цієї методики об’єкт створюється під час першого звернення до нього, у методі get…(), після аналізу, чи дорівнює посилання на цей об’єкт null. Більшість цих методів має бути публічними, для того, щоб об’єкти мали можливість спілкуватися між собою через модель.

2.3.3 Шар компонент

Цей шар складається з класів, що реалізують моделі складових частин досліджуваної системи.

Основні завдання цього шару такі:

– моделювати складові частини системи;
– налагодити через модель зв’язки з усіма необхідними компонентами;
– реалізувати правила дії активних компонент моделі;
– передавати дані, що характеризують роботу, моделі до накопичувачів статистики;
– надавати дані для динамічної індикації процесу моделювання.

Особливість цього шару полягає у тому, що до його складу входять як класи, що потребують створення, так і вже існуючі класи, які надає користувачеві фреймворк Simulation та Java.

Найчастіше, під час реалізації цього шару доводиться створювати класи для активних компоненти системи, які мають успадковувати клас Actor і у цих класах має бути реалізований метод rule(). Для реалізації черги можна використовувати колекції, але краще використовувати класи черг фреймворку Simulation. Теж саме стосується і класів для накопичення статистичної інформації.

3 ЕТАПИ ВИКОНАННЯ РОБОТИ

Виконання РГР відбувається параллельно з виконанням лабораторних робіт і складається з 7 етапів.

3.1 Етап1

На цьому етапі студент знайомиться із завданням, проводить первинний аналіз предметної області і створює перший варіант інтерфейсу користувача.

3.1.1 Первинний аналіз завдання

Результатом цього аналізу має бути перелік основних об’єктів моделі і їх характеристики, які підлягають налаштуванню через інтерфейс користувача.

Перш за все слід визначити активні об’єкти моделі, які потребують налаштування часу виконання робочих операцій (зазвичай випадкового).

Далі слід визначитися з параметрами, які потрібно налаштовувати, але вони не потребують генерації випадкових чисел (кількість об’єктів, їх параметри, імовірності деяких подій)

Майже у всії проектах має налаштовуватися час моделювання.

Результати первинного аналізу можуть виглядати приблизго так, як показано у таблиці 3.1.

Таблиця 3.1 – Перелік компонентів для налаштування моделі

Назва	Тип	Ідентифікатор
Продуктивність бульдозера	ChooseRandom	chooseRandomBuldo
Кількість авто	ChooseData	chooseDataAvto
Час моделювання	ChooseData	chooseDataFinishTime

Особливу увагу слід приділити назвам компонентів. Доцільно зберігати префікс, що визначає тип. Назва має однозначно ідентифікувати компонент.

3.1.2 Вікно інтерфейсу користувача.

Інтерфейс користувача може вглядати так, як представлено на рисунках 3.1–3.2. Він являє собою сукупність декількох основних панелей. Основою інтерфейсу є компонент JSplitPanel. Можна також обійтися звичайною панеллю з відповідними налаштуваннями GridBagLayout.

Ліву частину головного вікна займає панель для розміщення елементів налаштування моделі і присутня на екрані у всіх режимах роботи. В якості менеджера компоновки цієї панедлі вибрано GridLayout. Дані, що були визначені вище слід використати для формування цієї панелі. Всі компоненти панелі треба зробити доступним, скориставшись функцією контекстного меню компонента Expose Component. Одночасно будуть сформовані відповідні геттери.

Праворуч розташований компонент TabbedPane, закладки якого (панелі JScrollPane, JPanel або інші) забезпечують реалізацію різних варіантів роботи з моделлю. Назви закладок можна налаштувати через властивість панелі Tab.title

3.1.3 Технічне завдання

Технічне завдання створюється після первинного аналізу завдання і внесення до нього корективів. Студент має право спростити завдання, або внести до нього певні корективи. Всі ці зміни фіксуються у .docx документі, а також зберігається у файлі формату .htm з фільтром. Останній файл буде входити до проекту і відображатися у закладці ТЗ інтерфейсу користувача. Ця закладка, рисунок 3.1, містить текст технічного завдання у вигляді відображення HTML файлу. Сам файл tz.htm розташуємо в пакеті разом з цим класом.

Сама закладка являє собою панель JScrollPane на якій розташовано компонент JTextPane, який використовується для відображення тексту.

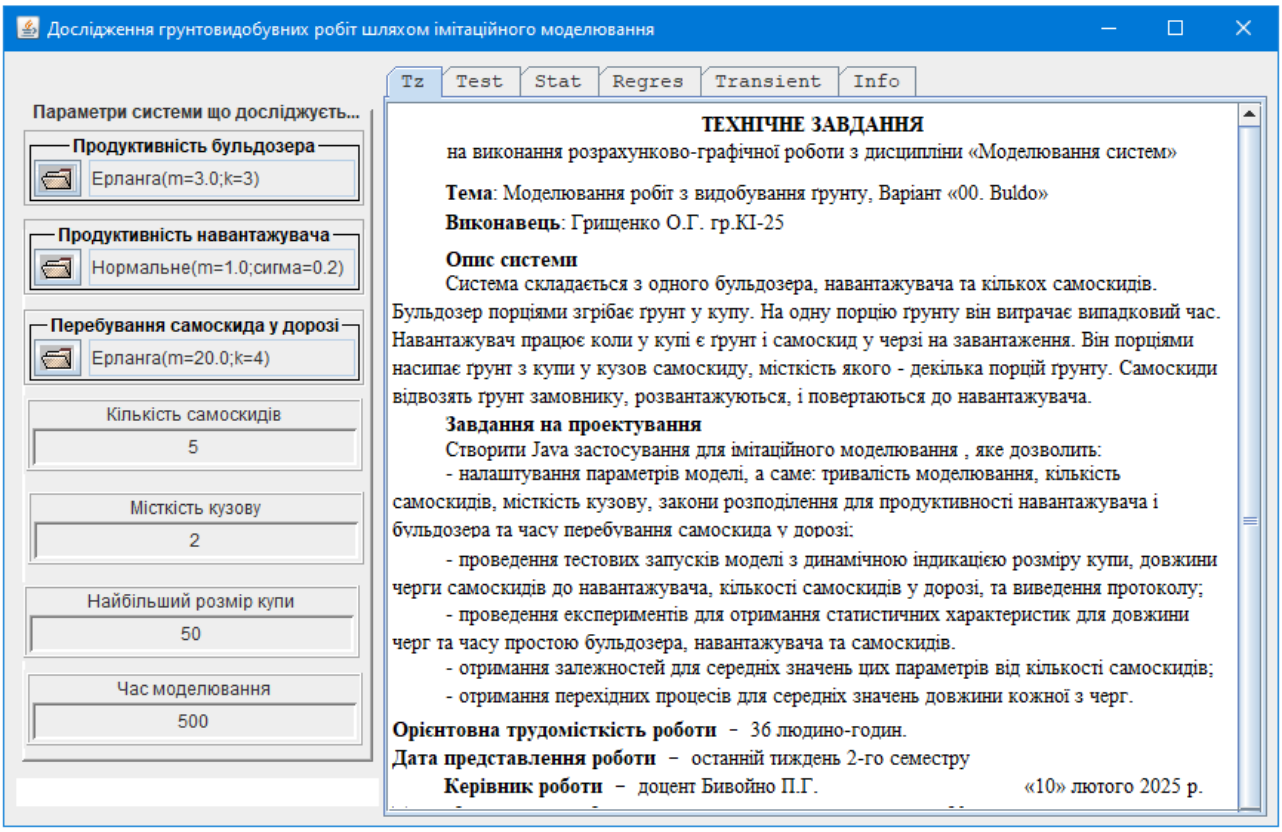

Рисунок 3.1 – Інтерфейс користувача моделі у режимі перегляду технічного завдання

Зробимо компонент JTextPane доступним, скориставшись функцією контекстного меню компонента Expose Component. Як результат отримаємо метод getJTextPane(), який треба модіфікувати відповідно з лістингом 3.1.

Лістинг 3. 1 – Код для відображення тексту файлу Tz на панелі

```
private JTextPane getJTextPane() {
  if (jTextPane == null) {
    jTextPane = new JTextPane();
```

```
        String str="tz.htm";
        URL url = getClass().getResource(str);
        try {
          jTextPane.setPage(url);
        } catch (IOException e33) {
          System.err
               .println("Problems with file "+str);
        }

      }
      return jTextPane;
    }
```

3.1.4 Реалізація закладки Info

Ця закладка має надавати інформацію про розробника проекту. Після вибору закладки на панелі має з'являтися віконце, де має бути повне ім'я студента, група, номер варінта, адреса електронної пошти і номер телефону.

Окрім того, зважаючи на дистанційне навчання, треба мати фото студента, щоб у викладача була більш повна уява про автора проекту.

Вигляд закладки показано на рисунку 3.2.

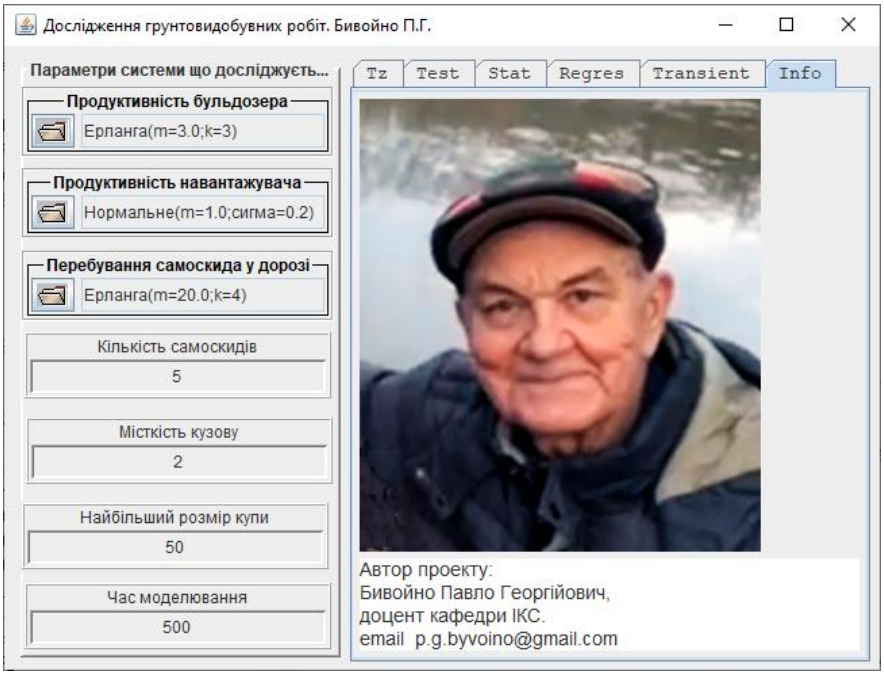

Рисунок 3.2 – Варіант закладки з інформацією про розробника

3.1.5 Створення панелі з інформацією

Додаймо до JTabbedPane панель типу JPanel і налаштуємо назву Info через властивість панелі Tab.title.

Налаштовуємо GridBagLayout для панелі.

У другому рядку контейнера розміщуємо JtextArea.

У першому рядку розміщуємо звичайну панель JPanel і назвемо її panelPhoto.

У компоненті JtextArea записуємо текстову інформацію про студента відкривши редактор тексту для властивості text компонента textArea.

Можна також налаштувати шрифт повідомлення, поле editable…

На завершення налаштуйте властивість менеджера компоновки Grow = none для нижнього рядка, а для верхнього і колонки налаштуйте Grow = 1.

3.1.6 Розміщення фото студента

Для початку треба мати фото. Імпортуйте його у папку resource проекту. Завантаження можна зробити через контекстне меню проекту, викликавши функції Import... → FileSystem → Next, вибравши папку з фото. Щоб побачити файл у пакеті після завантаження, треба зробити refresh проекту.

Для розміщення фото будемо використовувати панель panelPhoto, яку ми розташували над компонентом textArea.

Зробимо цю панель доступною, скориставшись функцією Expose Component. Як результат отримаємо метод getPanelPhoto().

Метод getPanelPhoto() слід перевизначити таким чином, як показано у лістингу 3.2. Слід також проконтролювати, щоб звернення до панелі у класі реалізовувалися тільки через цей метод.

Лістинг 3.2 – Метод створення панелі для відображення фото

```
private JPanel getPanelPhoto() {
 if (panelPhoto == null) {
   panelPhoto = new JPanel() {
     public void paintComponent(Graphics g){
       super.paintComponent(g);
       Graphics2D g2d = (Graphics2D) g;
       BufferedImage img;
       URL url =  getClass().getResource("/resource/photo.jpg");
       try {
         img = ImageIO.read(url);
       } catch (IOException e) {
         e.printStackTrace();
         return;
       }
       double k = (double)img.getHeight() / img.getWidth();
       int width = getWidth();
       int height = getHeight();
       if((double)height / width > k)
         height = (int) (width *k);
```

```
            else
              width = (int) (height /k);
            Image scaledImg = img.getScaledInstance(
                                  width, height, Image.SCALE_SMOOTH);
            g2d.drawImage(scaledImg,0,0,null);
          };
        };
       }
       return panelPhoto;
      }
```

Виправте помилки викликавши функцію Source/OrganizeImport використовуючи класи пакету java.awt.

Після цього можна тестувати застосунок.

3.2 Етап 2

3.2.1 Налаштування компонентів інтерфейсу користувача

Виберіть закони розподілу для генераторів випадкових чисел використовуючи знання набуті в результаті виконання лабораторної роботи 2. За одиницю виміру часу приймайте середнє значення найкоритшого інтервалу часу між подіями, що формують активні елементи. Наприклад, у моделі супермаркету найменшим інтервалом може бути час обробки касиром однієї покупки.

Визначтенсь з початковими значеннями числових параметрів моделі. Результати прийнятих рішень доцільно представити у вигляді таблиці 3.2.

Таблиця 3.2 – Налаштування компонентів управління моделлю

Назва	Ідентифікатор	Конструктор/Значення
Продуктивність бульдозера	chooseRandomBuldo	Erlang(2, 3)
Продуктивність навантажувача	chooseRandomLoader	Uniform(0.5, 1.5)
Кількість авто	chooseDataAvto	5
Час моделювання	chooseDataFinishTime	500

Після визначення із значеннями слід налаштувати компоненти.

Числові значення компонентів ChooseData задаються через властивість компонента text.

Для налаштування компонентів Choose Random треба виділити його на панелі дизайну і перейти у режим редагування коду. Далі знайти рядок коду, де цей компонент створюється. У наступному рядку призначити полю random посилання на потрібний генератор випадкових чисел, наприклад, так:

```
chooseRandomBuldo.setRandom(new Erlang(2, 3);
```

3.2.2 Створення панелі динамічної індикації «Test»

Закладка «Test» використовується для тестування роботи моделі із динамічною індикацією зміни розмірів черг та виведенням протоколу роботи моделі.

Перед створенням закладки слід визначитися з кількістю діаграм, які будуть відображати стан черг у системі. Нижче розглядається приклад з трьома діаграмами.

Додаймо до JTabbedPane панель типу JPanel і налаштуємо назву «Test» через властивість панелі Tab.title.

Налаштовуємо GridBagLayout для панелі. На рисунку 3.3 показано цю панель з рядками і колонками менеджера компоновки.

У четвертому рядку контейнера розміщуємо ще одну панель типу JPanel.

У трьох попередніх рядках розміщуємо компоненти widgets.Diagram фреймворку Simulation.

Додайте до менеджера компоновки ще одну колонку і розтягніть всі компоненти на дві колонки.

На нижній панелі розташуйте CheckBox і кнопку, як це показано на рисунку. Значенн Grow для нижнього рядка встановіть 0, а для рядків з діаграмами і обох колонок для Grow встановіть 1.

Тепер діаграми будуть змінювати свої розміри відповдно до розміру вікна.

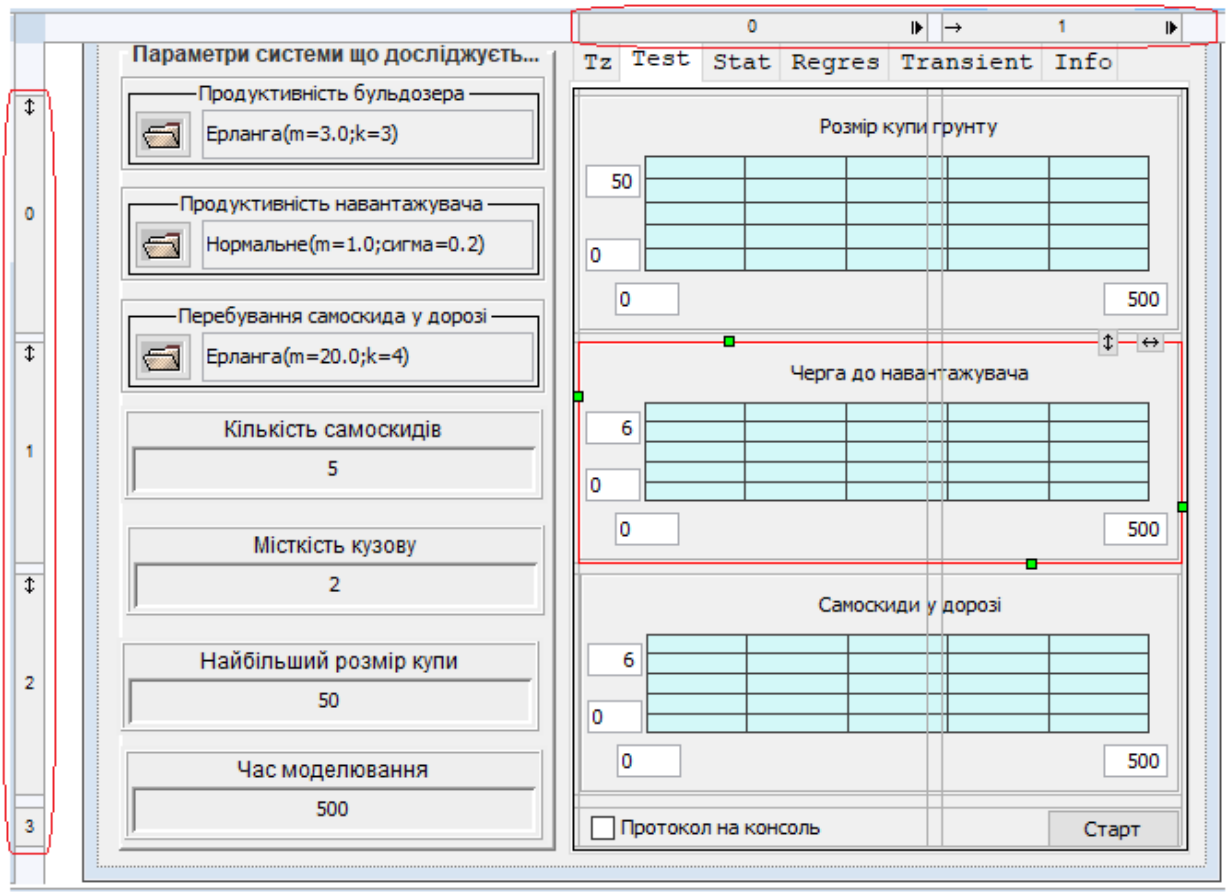

Рисунок 3.3 – Схема менеджера компоновки для панелі «Тест»

За допомогою контекстного меню компонентів rename налаштуйте зрозумілі назви діаграм (префікс diagram залишайте) і надайте публічний доступ (expose component) до них та компонента checkbox.

Налаштуйте заголовки диаграм (властивість titleText) та колір ліній майбутніх графіків (властивість painterColor).

3.2.3 Зв'язування налаштувань моделі з налаштуваннями діаграми

Діаграми мають автоматично змінювати свої налаштування у разі зміни налаштувань моделі. Для того, щоб налаштування діаграм відповідали налаштуванням моделі, події caretUpdate компонентів, що використовуються для налаштувань параметрів моделі, слід пов'язати з налаштуваннями діаграм.

З цією ж метою використовується і подія панелі componentShown.

Відповідні зв'язки схематично показані на рисунку 3.4.

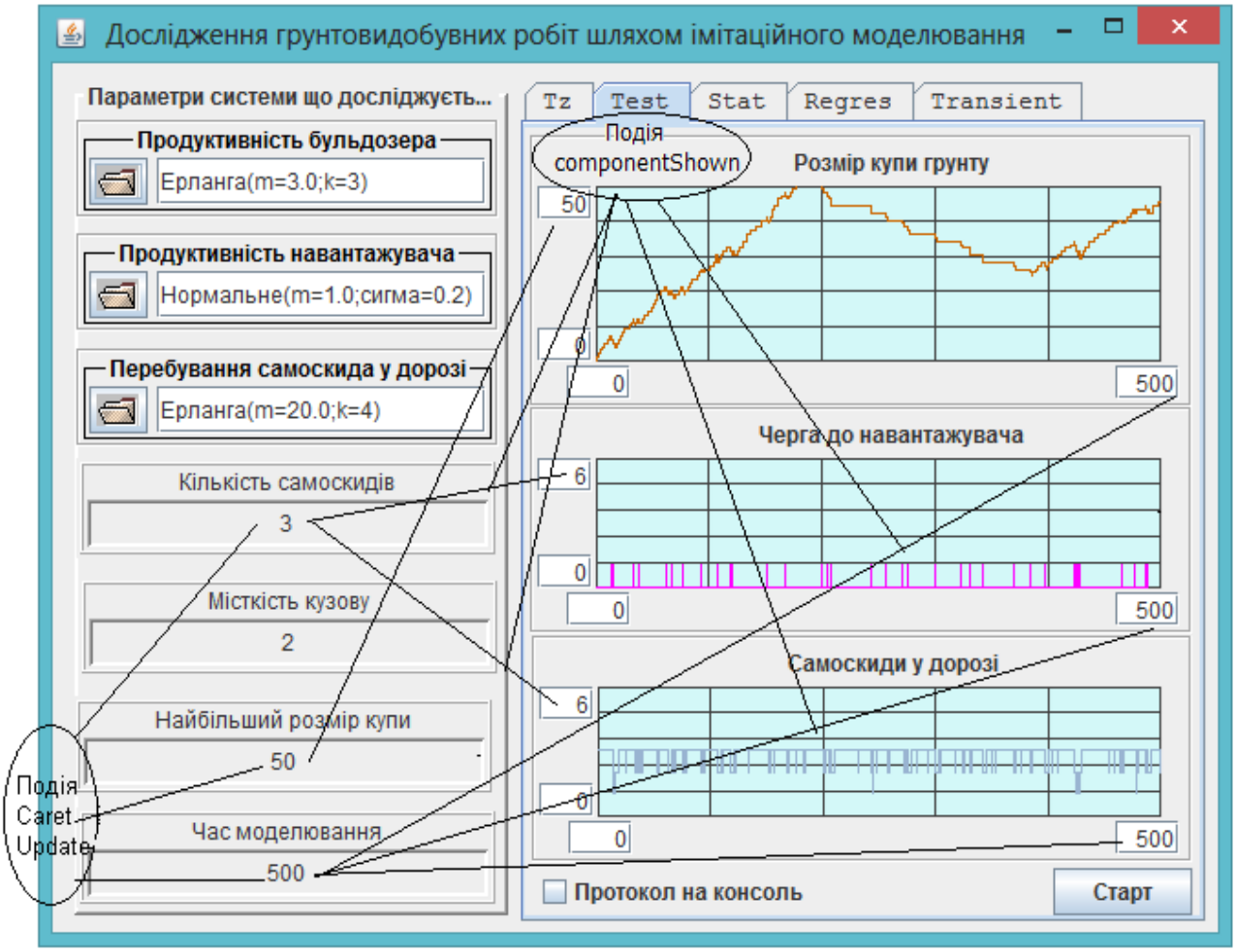

Рисунок 3.4 – Інтерфейс користувача моделі у режимі тестування

Подія caretUpdate компоненту, що використовується для налаштування часу моделювання, пов'язана з параметром horisontalMaxText діаграм. Але цей зв'язок має підтримуватися тільки у тому випадку, якщо відкрита панель «Test». Приклад методу, що пов'язаний з подією caretUpdate, і змінює

налаштування діаграм наведено у лістингу 3.3.

Лістинг 3.3 – Метод оновлення налаштувань діаграми, що пов’язаний з подією caretUpdate

```
  private void onChangeModelingTime() {
    if (getJPanelTest().isShowing()) {
      String str = chooseDataFinishTime.getText();
      getDiagramHeepSize().setHorizontalMaxText(str);
      getDiagramQueueToLoader().setHorizontalMaxText(str);
      getDiagramLorryOnRoad().setHorizontalMaxText(str);
    }
  }
```

Аналогічно можна пов’язати події caretUpdate інших компонентів ChooseData з властівостю діаграм verticalMaxText, якщо це потрібно.

Для того, щоб відновити зв’язки компонентів ChooseData з діаграмами після відкриття панелі «Test», використовується подія цієї панелі componentShown. Цю подію слід зв’язати з методом, код якого наведено в лістингу 3.4.

Лістинг 3.4 – Метод оновлення налаштувань діаграми, що пов’язаний з подією componentShown панелі «Test»

```
  private void onPanelTestComponentShown() {
    // Штучно формуємо подію CaretUpdate,
    // щоб обновити налаштування діаграми
    getChooseDataFinishTime().select(0,0);
  }
```

Далі слід протестувати застосунок.

3.2.4 Публічний програмний інтерфейс візуальної частини

Виконання даного етапу роботи над РГР дозволило сформувати програмний інтерфейс візуальної частини, який складається з методів доступу до елементів управління і діаграм., рисунок 3.5.

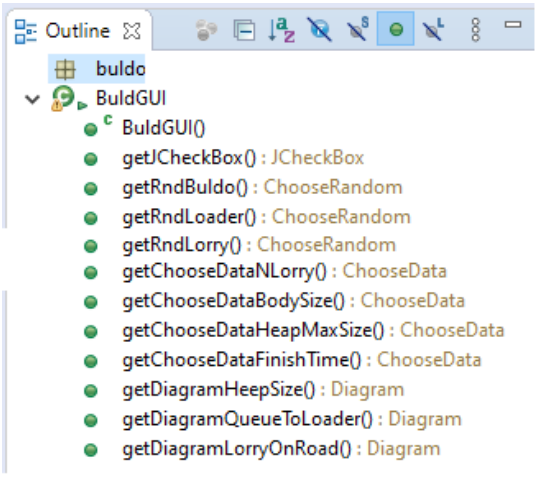

Рисунок 3.5 – Публічний програмний інтерфейс візуальної частини

3.3 Етап 3

На цьому етапі ми почнемо створювати модель нашої системи. Але першим кроком буде створення методу startTest() візуальної частини, який має бути пов’язаний з кнопкою «Старт» панелі «Test». Ця кнопка використовується для запуску моделі у режимі тестування

У лістингу 3.5. наведено приклад реалізації цього методу. Його можна взяти за основу для своєї реалізації, але доведеться виправляти помилки, які виникнуть після копіювання.

Лістинг 3. 5 - Метод запуску процесу моделювання у режимі тестування

```
  private void startTest() {
    //Готуємо діаграми для виведення графіку
    getDiagramHeepSize().clear();
    getDiagramQueueToLoaer().clear();
    getDiagramQueueLorryOnRoad().clear();
    //Створюємо диспетчера
    Dispatcher dispatcher = new Dispatcher();
    //Створюємо модель за допомогою фабрики
    IModelFactory factory = (d)-> new Model(d, this);
    Model model =(Model) factory.createModel(dispatcher);
    // Робимо кнопку «Старт» недосяжною на період роботи моделі
    getJButtonStart().setEnabled(false);
    dispatcher.addDispatcherFinishListener(
        ()->getJButtonStart().setEnabled(true));
    //Готуємо модель до роботи у режимі тестування
    model.initForTest();
    //Запускаємо модель
    dispatcher.start();
  }
```

3.3.1 Пристосування методу startTest() до іншого проєкту

Перш за все слід привести у відповідність назви методів звернення до діаграм та їх кількість.

Наступним кроком буде створення класу Model у проекті. Це можна зробити шляхом автогенерації коду для виправлення помилки.

Таким же способом можна згенерувати конструктор у класі Model.

Якщо ім’я ідентифікатора кнопки «Старт» не є jButtonStart,то слід привести у відповідність назву кнопки.

Остання помилка виникає у зв’язку з тим, що в моделі нема ще методу initForTest(). Заготовку цього методу можна також створити шляхом автогенерації коду для виправлення помилки.

3.3.2 Клас Model

У попередньому пункті ми вже створили клас моделі. Тепер будемо наповнювати його вмістом.

Клас будемо будувати виходячи з того, що модель буде створюватися перед кожним її запуском. Це значно спрощує програмування і підвищує надійність моделі, тому що при цьому усі компоненти моделі теж створюються заново і не потребують додаткової ініціалізації.

Модель являє собою сукупність об’єктів, які взаємодіють між собою. Тож розробку моделі треба починати з формування переліку цих об’єктів. Такий перелік є результатом об’єктно орієнтованого аналізу системи і його можна представити у вигляді таблиці 3.3, в якій, для прикладу, наведені об’єкти моделі грунтовидобувних робіт.

Таблиця 3.3 – Абстракції системи, що входять до складу моделі

Абстракція	Завдання	Клас
Бульдозер (buldo)	Передавати через випадкові інтервали часу порції ґрунту до купи.	Buldo (Actor)
Навантажувач (loader)	Завантажувати потрібну кількість порцій ґрунту у самоскид, за випадковий час.	Loader (Actor)
Самоскид (lorry)	Їхати до навантажувача, ставати у чергу, чекати заповнення кузову, їхати на розвантаження і повертатися знов.	Lorry (Actor)
multiLorry	Бригада самоскидів	MultiLorry (MultiActor)
heap	Купа ґрунту.	Store
queueToLoader	Черга самоскидів до навантажувача	QueueFor Transation
queueLorryOn Road	Кількість самоскидів у дорозі	QueueFor Transation
histoHeap	Накопичувач інформації про розміри купи	
histoLoader	Накопичувач інформації про час простою навантажувача	Histo
histoLorry	Накопичувач інформації про час простою самоскидів	Histo
histoQueueTo Loader	Накопичувах інформаці] про довжину черги до навантажувача.	DiscretHisto
histoLorryOn Road	Накопичувач інформації про кількість самоскидів у дорозі	DiscretHisto

3.3.2.1 Поля моделі

Перелік атрибутів, які мають бути у будь-якій моделі, наведено у лістингу 3.6. У цьому переліку Dispatcher – це клас Simulation, а Gui– це клас

візуальної частини проекту. Якщо у проекті назва класу візуальної частини інша, то замість Gui треба використовувати іншу назву класу.

Лістинг 3.6 - Перелік обов'язкових атрибутів моделі

```
//Посилання на диспетчера
private Dispatcher dispatcher;
//Посилання на візуальну частину
private Gui gui;
```

У лістингу 3.7, як приклад, наведено перелік атрибутів для моделі грунтовидобувних робіт, яка розглядається в якості зразка. Перелік сформовано на підставі таблиці 3.3.

Формуючи перелік атрибутів ми пишемо назви їх класів. Якщо для черг та гістограм ми викоистовуємо класи фреймворку Simulation, то для активних компонентів доводиться створювати класи, які успадковують клас Actor.

Заготовки таких класів можна створювати шляхом автогенерації коду у режимі виправлення помилки. Але треба не забувати дописувати, що клас успадковує клас Actor. Одночасно з цим можна згенерувати і заготовку методу rule для такого класу.

Лістинг 3.7 - Перелік атрибутів моделі грунтовидобувних робіт

```
//Бульдозер
private Buldo buldo;
//Навантажувач
private Loader loader;
//Самоскид (зразок для multiLorry)
private Lorry lorry;
//Бригада самоскидів
private MultiActor multiLorry;
//Купа грунту
private Store heep;
//Черга самоскидів до навантажувача
private QueueForTransactions<Lorry> queueToLoader;
//Черга для самоскидів у дорозі
private QueueForTransactions<Lorry> queueLorryOnRoad;
// Гістограма для довжини черги до навантажувача
DiscretHisto histoForQueueToLoader = new DiscretHsto();
// Гістограма для часу простою бульдозера
Histo histoBuld = new Histo();
// Гістограма для часу простою навантажувача
```

```
  Histo histoLoader = new Histo();
  // Гістограма для часу простою самоскида
  Histo histoLorry = new Histo();
  // Гістограма для розмірів купи
  Histo histoHeap = new Histo();
```

Після створення полів класу моделі слід згенерувати геттери до цих полів. Ці методи в подальшому ми будемо доопрацьовувати для відкладеного створення об’єктів.

3.3.2.2 Конструктор моделі

Для створення моделі використовується конструктор з двома параметрами, який забезпечує гарантовану передачу моделі посилань на візуальну частину і диспетчера. Заготовку цього конструкторв було створено під час редагування методу startTest() візуальної частини. Тепер його слід наповнити кодом, лістинг 3.8.

Такий конструктор можна вважати стандартним для виконання РГР, незалежно від теми.

Лістинг 3.8 - Конструктор моделі

```
public Model(Dispatcher d, Gui g) {
    if (d == null || g == null) {
      System.out.println("Не визначено диспетчера або GUI для Model");
      System.out.println("Подальша робота неможлива");
      System.exit(0);
    }
    dispatcher = d;
    gui = g;
    //Передаємо акторів до стартового списку диспетчера
    componentsToStartList();
  }
```

3.3.2.3 Метод передачі акторів моделі до стартового списку диспетчера

Останній рядок в конструкторі забезпечує передачу акторів моделі до стартового списку диспетчера за допомогою методу моделі componentsToStartList(), який ми ще не створили.

Вміст цього методу залежіть від конкретного завдання. У нашому випадку цей метод має вигляд, представлений у лістингу 3.9.

До стартового списку передається бульдозер, навантажувач і бригада самоскидів. Окремі самоскиди не передаються. Вони будуть створені і завантажені об’єктом типу MultiActor (бригадою самоскидів).

Лістинг 3.9 - Метод componentsToStartList()

```
public void componentsToStartList() {
  // Передаємо акторів диспетчеру
  dispatcher.addStartingActor(getBuldo());
  dispatcher.addStartingActor(getLoader());
  dispatcher.addStartingActor(getMultiLorry());
}
```

Слід звернути увагу на те, що для звертання до акторів використовуються методи get…(), у яких реалізовано відкладене створення об'єктів.

3.3.2.4 Доопрацювання методів доступу до об'єктів типу Actor

Акторів проекту ми будемо створювати у методах get…() шляхом використання усталених конструкторів. Всю інформацію, що потрібна цим об'єктам для функціонування будемо також передавати у цих методах, але зробимо це на наступному етапі, коли будуть повністю реалізовані класи цих об'єктів..

Як приклад методу для створення актора наведемо метод getBuldo(), лістинг 3.10.

Лістинг 3.10 - Метод відкладеного створення об'єкта бульдозер

```
public Buldo getBuldo() {
  if (buldo == null) {
    buldo = new Buldo();
  }
  return buldo;
}
```

3.3.2.5 Доопрацювання методів доступу до об'єктів типу MultiActor

Об'єкти класу MultiActor створюються дещо інакше. Такому об'єкту потрібно передати посилання на зразок, що буде клонуватися, та задати потрібну кількість клонів.

Як приклад, наводимо тут метод створення бригади авто, лістинг 3.11.

Лістинг 3.11 – Метод відкладеного створення бригади самоскидів

```
public MultiActor getMultiLorry() {
  if (multiLorry == null) {
    multiLorry = new MultiActor();
    multiLorry.setNameForProtocol("MultiActor для самоскидів");
    multiLorry.setOriginal(getLorry());
    multiLorry.setNumberOfClones(gui.getChooseDataNLorry().getInt());
```

```
    }
    return multiLorry;
  }
```

3.3.2.6 Доопрацювання методів доступу до черг

Об’єкти для черг створюються також інакше. Посилання, що необхідні чергам, слід передавати або через конструктори з параметрами, або через методи set…(). Для прикладу наведемо метод getQueueToLoader(), лістинг 3.12.

Лістинг 3.12 – Метод відкладеного створення черги до навантажувача

```
  public QueueForTransactions getQueueToLoader() {
    if (queueToLoader == null) {
      queueToLoader = new QueueForTransactions();
      queueToLoader.setNameForProtocol("Черга до навантажувача");
      queueToLoader.setDispatcher(dispatcher);
      queueToLoader.setDiscretHisto(getHistoForQueueToLoader());
    }
    return queueToLoader;
  }
```

3.3.2.7 Методи доступу до гістограм

Об’єкти для гістограм не потребують додаткових налаштувань, тому вони були створені при оголошенні полів. Також були згенеровані методи доступу до них, і ці методи не потребують доопрацювання.

У лістингу 3.13 наведено метод доступу до гістограми. Але без цих методів можна і обійтись, тому що поля гістограм мають пакетний доступ.

Лістинг 3.13 – Метод доступу до гістограми

```
  public Histo getHistoHeap() {
    return histoHeap;
  }
```

3.3.2.8 Метод ініціалізації моделі для режиму «Test»

Завершується робота над класом моделі створенням методів ініціалізації моделі для можливих режимів роботи. Ці методи налаштовують модель до вимог кожного з режимів.

Заготовку методу ініціалізації моделі для режиму «Test» вже було створено раніше. Тепер треба її доповнити кодом.

Лістинг 3.14 містить приклад реалізації цього методу для моделі грунтовидобувних робіт.

У методі чергам передаються посилання на об’єкти класу Painter відповідних діаграм. Це забезпечить відображення цих черг на діаграмах.

Окрім того, диспетчер може бути налаштований на виведення протоколу роботи моделі на консоль.

Лістинг 3.14 – Метод ініціалізації моделі для режиму «Test»

```
public void initForTest() {
  // Передаємо чергам painter-ів для динамічної індикації
  getHeap().setPainter(gui.getDiagramHeepSize().getPainter());
  getQueueToLoader()
      .setPainter(gui.getDiagramQueueToLoader().getPainter());
  getQueueLorryOnRoad()
      .setPainter(gui.getDiagramLorryOnRoad().getPainter());
  if (gui.getJCheckBox().isSelected())
    dispatcher.setProtocolFileName("Console");
}
```

3.3.3 Тестування моделі

На цьому етапі ми створили модель і всі засоби для створення об’єктів, що входять до її складу. Однак для акторів створено тільки заготовки класів. Тому актори моделі ще не знають, що їм робити. Але протестувати роботу моделі вже можна.

Для тестування запустимо застосунок на виконання. Відкриємо панель «Test». Увімкнемо перемикач «Протокол на консоль» і натиснемо кнопку «Старт».

На консолі має з’явитися протокол роботи моделі, який має виглядати приблизно так, як показано на рисунку 3.6.

Актори стартують і тут же закінчують роботу, тому що для них не прописані правила дії.

```
Модельний час 0.0
  buldo.Buldo стартує
  buldo.Buldo роботу завершив
  buldo.Loader стартує
  buldo.Loader роботу завершив
  MultiActor для самоскидів стартує
  MultiActor для самоскидів роботу завершив
  buldo.Lorry1 стартує
  buldo.Lorry1 роботу завершив
  buldo.Lorry2 стартує
  buldo.Lorry2 роботу завершив
  buldo.Lorry3 стартує
  buldo.Lorry3 роботу завершив
  Диспетчер роботу завершив
```

Рисунок 3.6 – Протокол роботи моделі

3.4 Етап 4

На цьому етапі ми маємо реалізувати спадкоємців класу Actor. У проекті, який розглядається в якості зразка, активними абстракціями системи є бульдозер, навантажувач та самоскид. Розглянемо докладно їх поведінку, представимо ці правила дії у вигляді діаграм діяльності та реалізуємо відповідні класи.

3.4.1 Бульдозер

Головне завдання цієї абстракції – додавати порції ґрунту до купи через випадкові інтервали часу. Якщо розмір купи збільшується до критичного розміру, бульдозер зупиняється і відновлює роботу, коли розмір купи зменшиться удвічі. Інформацію по зупинки у роботі бульдозер накопичує у гістограмі. Свої дії бульдозер має виконувати впродовж усього часу моделювання для будь якого режиму роботи моделі.

3.4.1.1 Атрибути абстракції Бульдозер

Для роботи бульдозеру необхідні дані, що наведено у таблиці 3.4.

Таблиця 3.4 – Атрибути абстракції Бульдозер

Назва поля	Клас	Призначення поля	Джерело
heap	Store	Посилання на купу ґрунту	Model
heapMaxSize	double	Критичний розмір купи	GUI
finishTime	double	Час моделювання	GUI
rnd	rnd.Randomable	Генератор випадкових чисел	GUI
histoBuldo	stat.Histo	Гістограма	Model

3.4.1.2 Діаграма діяльності абстракції «Бульдозер»

Правила дії бульдозера схематично можна представити у вигляді діаграми діяльності, рисунок 3.7.

Зверніть увагу на специфічні блоки діаграми, які притаманні тільки акторам:

- затриматися на випадковий час підготовки порції грунту;
- чекати поки розмір купи зменшиться удвічи.

3.4.1.3 Клас Buldo для абстракції «Бульдозер»

Об’єкти класу Buldo представляють абстракцію бульдозер. У лістингу 3.15 наведено текст класу з коментарями.

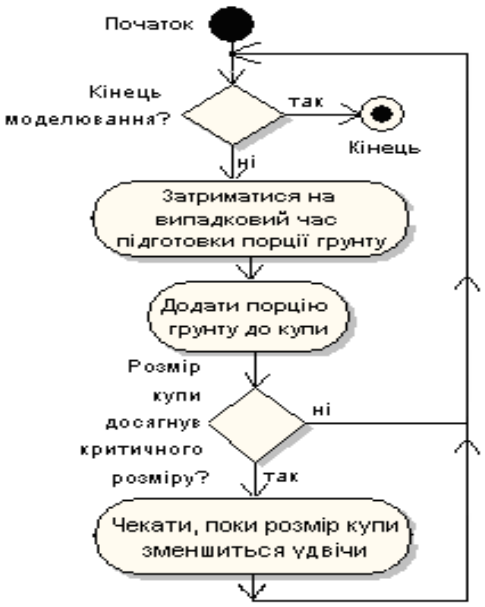

Рисунок 3.7 – Діаграма діяльності абстракції бульдозер

Лістинг 3.15 – Клас Buldo

```
public class Buldo extends Actor {
  // Посилання на купу грунту, до якої бульдозер додаає грунт
  private Store heap;

  // Критичний розмір купи земли, при якому бульдозер припиняє роботу
  private double heapMaxSize;

  //Тривлістьсть роботи бульдозера
  private double finishTime;

  // Генератор часу, що витрачає бульдозер на одну порцію грунту
  private Randomable rnd;

  // Правила дії бульдозера.
  // Бульдозер видобує порції ґрунту, витрачаючи на це деякий час
  protected void rule() throws DispatcherFinishException {
    // Умова після виконання якої бульдозер відновлює роботу
    BooleanSupplier heapHalfSize = ()->heap.getSize() <= heapMaxSize/2;
    // Цикл правил дії бульдозера
    while (getDispatcher().getCurrentTime() <= finishTime) {
      // Затримка на час формування порції грунту
      holdForTime(rnd.next());
      // Збільшення розміру купи на одну порцію
      getDispatcher().printToProtocol(
          "  " + getNameForProtocol() + " додає порцію грунту.");
      heap.add(1);
      // Зупинка,якщо купа досягла критичного розміру
      if (heap.getSize() >= heapMaxSize) {
          waitForCondition(heapHalfSize,
              "поки купа зменшіться удвічи");
      }
```

```
    }
  }

  // Методи доступу до полів
  public void setFinishTime(double finishTime) {
      this.finishTime = finishTime;
  }
  public void setHeap(Store heap) {
      this.heap = heap;
  }
  public void setHeapMaxSize(double heapMaxSize) {
      this.heapMaxSize = heapMaxSize;
  }
  public void setRnd(Randomable rnd) {
      this.rnd = rnd;
  }
}
```

3.4.2 Навантажувач

Навантажувач працює із купою ґрунту і чергою самоскидів. Якщо є самоскид, готовий до завантаження, навантажувач забирає його з черги на обслуговування. Далі починається цикл завантаження самоскиду. Якщо у купі є ґрунт , навантажувач перевантажує порцію ґрунту із купи до кузова самоскида. Цикл завантаження повторюється поки самоскид не буде завантажений. На перевантаження порції ґрунту навантажувач витрачає випадковий час. Якщо у купі закінчився ґрунт або нема самоскида, що потребує завантаження, навантажувач переходить у стан чекання. Після повного завантаження правила дії повторюються для наступного самоскиду.

Навантажувач працює впродовж усього часу моделювання. Інформацію по зупинки у роботі навантажувач накопичує у гістограмі.

3.4.2.1 Атрибути абстракції «Навантажувач»

Для роботи навантажувача необхідні дані, які наведені у таблиці 3.5.

Таблиця 3.5 – Атрибути абстракції Навантажувач

Назва поля	Клас	Призначення поля	Джерело
heap	Store	Посилання на купу ґрунту	Model
queueToLoader	qusystem.Queue ForTransaction	Посилання на чергу самоскидів, що чекають завантаження	Model
finishTime	double	Час моделювання	Gui
rnd	rnd.Randomable	Посилання на генератор випадкових чисел	Gui
histoLoader	stat.Histo	Гістограма	Model

3.4.2.2 Діаграма діяльності абстракції «Навантажувач»

Правила дії навантажувача схематично можна представити у вигляді

діаграми діяльності, рисунок 3.8.

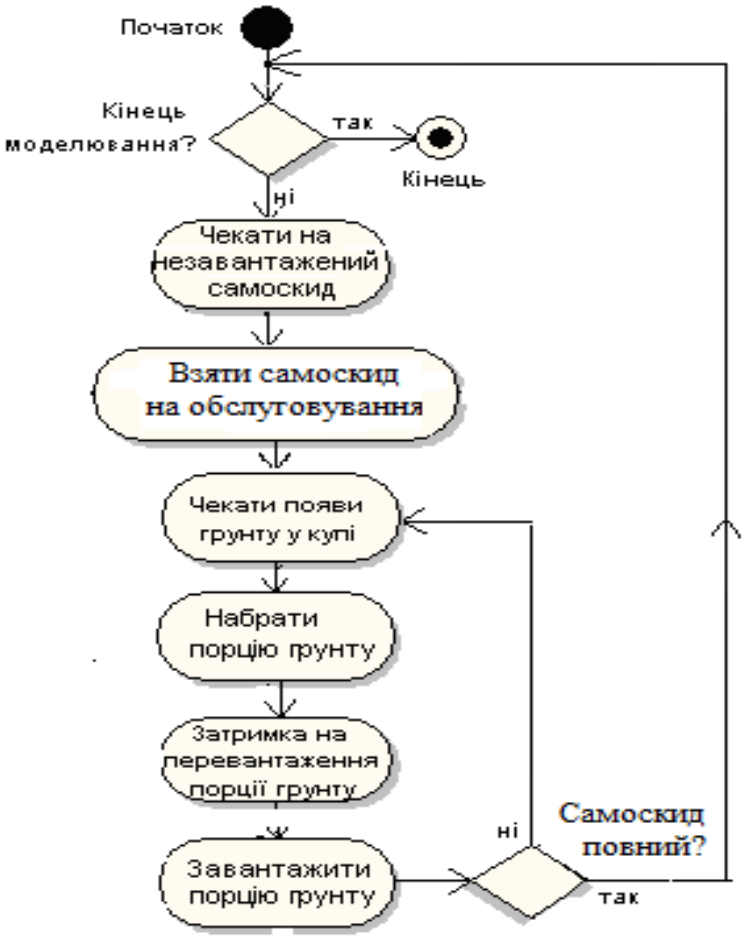

Рисунок 3.8 – Діаграма діяльності абстракції навантажувач

3.4.2.3 Клас Loader для абстракції «Навантажувач»

Об’єкти цього класу представляють абстракцію навантажувач. У лістингу 2.17 наведено текст класу з коментарями.

Лістинг 3.16 - Клас Loader

```
public class Loader extends Actor {

  // Купа грунту
  private Store heap;

  // Черга самосвалів, що чекають завантаження
  private QueueForTransactions<Lorry> queueToLoader;

  // Тривалість роботи навантажувача
  private double finishTime;

  // Генератор часу, що витрачається на одну порцію грунту
  private Randomable rnd;

  // Правила дії навантажувача.
  protected void rule() throws DispatcherFinishException {
```

```
    // Умова наявності самоскида у черзі
    BooleanSupplier  isLorry = () -> queueToLoader.size() > 0;
    // Умова наявності грунту у купі
    BooleanSupplier  heapSize = () -> heap.getSize() > 0;
    // Цикл виконанння правил дії навантажувача
    while (getDispatcher().getCurrentTime() <= finishTime) {
      // Перевірка, чи є в черзі самоскид,
      // і якщо його нема - чекання
      waitForCondition(isLorry, "має бути самоскид");
      // Забираємо самоскид на обслуговування
      Lorry lorry = queueToLoader.removeFirst();
      // Цикл завантаження самоскиду
      while (!lorry.isFull()) {
        waitForCondition(heapSize, "має бути грунт у купі");
        getDispatcher().printToProtocol(getNameForProtocol()
                                         + " бере порцію грунту");
        heap.remove(1);
        // Затримка на час перевантаження
        holdForTime(rnd.next());
        // Додаємо грунт у самосвал
        lorry.addPortion();
        getDispatcher().printToProtocol(
            getNameForProtocol() + " додає порцію грунту у самоскид "
                                 + lorry.getNameForProtocol());
      }
    }
  }
  // Методи доступу до полів
  public void setFinishTime(double finishTime) {
    this.finishTime = finishTime;
  }
  public void setHeap(Store heap) {
    this.heap = heap;
  }
  public void setQueueToLoader(
                             QueueForTransactions<Lorry>queueToLoader) {
    this.queueToLoader = queueToLoader;
  }
  public void setRnd(Randomable rnd) {
    this.rnd = rnd;
  }
}
```

3.4.3 Самоскид

Завданням абстракції самоскид є перевезення ґрунту від навантажувача до замовника. Шлях до замовника та від замовника до навантажувача потребує випадкових проміжків часу. Під'їхавши до навантажувача самоскид стає у чергу до навантажувача і чекає, поки його завантажать. Після цього знову їде до замовника. У замовника самоскид розвантажується і знову їде до навантажувача. Працює самоскид впродовж усього часу моделювання.

Інформацію про час чекання у черзі самоскид накопичує у гістограмі.

Особливість поведінки самоскида полягає у тому, що окрім основної діяльності він має реєструватися у списку, який реєструє самоскиди, що їдуть до навантажувача та від нього. Це пов’язано з вимогою завдання забезпечити динамічну індикацію самоскидів, які знаходяться у дорозі.

3.4.3.1 Атрибути абстракції «Самоскид»

Для роботи самоскиду необхідні дані, перелік яких наведено у таблиці 3.6.

Таблиця 3.6 – Атрибути абстракції Самоскид

Назва поля	Клас	Призначення поля	Джерело
queueToLoader	qusystem. QueueForTransaction	Посилання на чергу самоскидів, що чекають завантаження	Model
queueLorryOnRoad	qusystem. QueueForTransaction	Список, де реєструються самоскиди, що знаходяться у дорозі	Model
finishTime	double	Час моделювання	Gui
bodySize	double	Розмір кузова самоскида у порціях ґрунту	Gui
rnd	rnd.Randomable	Посилання на генератор випадкових чисел	Gui
body	double	Кількість порцій ґрунту у кузові самоскида	this
histoLorry	stat.Histo	Гістограма	Model

3.4.3.2 Діаграма діяльності для абстракції «Самоскид»

Правила дії самоскида схематично можна представити у вигляді діаграми діяльності, рисунок 3.9.

Особливість поведінки самоскида полягає у тому, що окрім основної діяльності він має реєструватися у списку, який реєструє самоскиди, що їдуть до навантажувача та від нього.

Це пов’язано з вимогою завдання забезпечити динамічну індикацію самоскидів, які знаходяться у дорозі.

Рисунок 3.9 – Діаграма діяльності абстракції самоскид

3.4.3.3 Клас Lorry для абстракції «Самоскид»

Об’єкти цього класу представляють абстракцію самоскид. У лістингу 3.17 наведено текст класу з коментарями.

Лістинг 3.17 - Клас Lorry

```
public class Lorry extends Actor {

  // Тривалість роботи самоскида
  private double finishTime;

  // Черга самоскидів у дорозі
  private QueueForTransactions<Lorry> queueLorryOnRoad;

  // Черга самоскидів до нантажувача
  private QueueForTransactions<Lorry> queueToLoader;

  // Генератор часу, що витрачається на дорогу в один кінець
  private Randomable rnd;

  // Місткість кузову
  private int bodySize;

  // Рівень завантаження кузову
  private int bodyLoad;

  // Правила дії абстракції "Самоскид".

  public void rule() throws DispatcherFinishException {
    //Умова, після виконання якої самоскид може відїжджати
```

```
    BooleanSupplier  isBodyFull = () -> isFull();
    // Цикл правил дії самоскида
    while (getDispatcher().getCurrentTime() <= finishTime) {
      // Самосвал їде до навантажувача
      // і реєструється у списку самоскидів, що їдуть
      queueLorryOnRoad.addLast(this);
      holdForTime(rnd.next());
      // вилучає себе із відповідного списку
      queueLorryOnRoad.remove(this);
      // Самосвал стає у чергу до навантажувача,
      queueToLoader.addLast(this);
      // Чекає поки завантажать
        waitForCondition(isBodyFull, "кузов має бути повним");
      // Самосвал їде на розвантаження
      getDispatcher().printToProtocol(
          getNameForProtocol() + " поїхав розвантажуватися");
      // реєструється у списку самоскидів, що їдуть
      queueLorryOnRoad.add(this);
      holdForTime(rnd.next());
      // вилучає себе із списку самоскидів, що їдуть
      queueLorryOnRoad.remove(this);
      getDispatcher().printToProtocol(
          getNameForProtocol() + " розвантажується");
      // Самоскид розвантажується
      bodyLoad = 0;
    }
  }

  // Методи, які використовує навантажувач
  // ////////////////////////////////////

  // Метод, що додає порцію грунту у кузов самоскида
  public void addPortion() {
    bodyLoad++;
    getDispatcher().printToProtocol(
        getNameForProtocol() + "- у кузові стало " + bodyLoad);
  }

  // Метод перевірки завантаженості самоскида
  public boolean isFull() {
    return bodyLoad >= bodySize;
  }
  ////////////////////////////////////////

  // Методи ініціалізації самоскида
  public void setQueueLorryOnRoad(
              QueueForTransactions<Lorry> queueLorryOnRoad) {
    this.queueLorryOnRoad = queueLorryOnRoad;
  }

  public void setQueueToLoader(
```

```
                QueueForTransactions<Lorry> queueToLoader) {
    this.queueToLoader = queueToLoader;
  }
  public void setRnd(Randomable rnd) {
    this.rnd = rnd;
  }
  public void setBodySize(int bodySize) {
    this.bodySize = bodySize;
  }
  public void setFinishTime(double finishTime) {
      this.finishTime = finishTime;
  }
}
```

3.4.4 Доопрацювання моделі

Після того, як створено класи акторів, можна завершити методи доступу до об'єктів цих класів у класі моделі. Переходимо до класу моделі і доповнюємо методи get… для акторів. Приклади реалізації цих методів наведено в лістигах 3.18 – 3.20.

Лістинг 3.18 – Метод доступу та ініціалізації об'єкта «Бульдозер»

```
  // Метод створення бульдозера
  public Buldo getBuldo() {
    if (buldo == null) {
      buldo = new Buldo();
      buldo.setNameForProtocol("Бульдозер");
      buldo.setHeap(getHeap());
      buldo.setHistoForActorWaitingTime(histoBuldo);
      buldo.setFinishTime(gui.getChooseDataFinishTime().getDouble());
      buldo.setHeapMaxSize(gui.getChooseDataHeapMaxSize().getDouble());
      buldo.setRnd(gui.getRndBuldo().getRandom());
    }
    return buldo;
  }
```

Лістинг 3.19 – Метод доступу та ініціалізації об'єкта «Навантажувач»

```
  // Метод створення навантажувача
  public Loader getLoader() {
    if (loader == null) {
      loader = new Loader();
      loader.setNameForProtocol("Навантажувач");
      loader.setHeap(getHeap());
      loader.setHistoForActorWaitingTime(histoLoader);
      loader.setQueueToLoader(getQueueToLoader());
      loader.setRnd(gui.getRndLoader());
      loader.setFinishTime(gui.getChooseDataFinishTime().getDouble());
    }
    return loader;
```

Лістинг 3.20 – Метод доступу та ініціалізації об’єкта «Самоскид»

```
// Метод створення зразка самоскида
public Lorry getLorry() {
  if (lorry == null) {
    lorry = new Lorry();
    lorry.setNameForProtocol("Самоскид");
    lorry.setQueueToLoader(getQueueToLoader());
    lorry.setQueueLorryOnRoad(getQueueLorryOnRoad());
    lorry.setHistoForActorWaitingTime(histoLorry);
    lorry.setBodySize(gui.getChooseDataBodySize().getInt());
    lorry.setRnd(gui.getRndLorry());
    lorry.setFinishTime(gui.getChooseDataFinishTime().getDouble());
  }
  return lorry;
}
```

3.4.5 Тестування моделі

Тестування моделі проводимо так само, як і на попередньому етапі. Але зараз протокол вже має бути довгим, а на діаграмах мають з’явитися графіки зміни розмірів черг у часі.

3.5 Етап 5

На цьому етапі ми реалізуємо роботу моделі в режиі накопичення та відображення статистичних даних. Частково роботу по реалізації цього етапу було виконано раніше. До черг та деяких акторів були підключені гістограми. В правилах дії деяки акторів було реалізовано збирання та накопичення статистичної інформаці.

Тепер настав час реалізувати доступ до статистичної інформації та відображення її на екрані. Для цього потрібно внести певні доповнення до моделі та візуальної частини проєкту.

3.5.1.1 Реалізація закладки «Stat» у візуальній частині проєкту

Додаймо компонент StatisticsManager із фреймворку Simulation на JTabbedPane в якості закладки і змінимо назву закладки на «Stat». Після цього інтефейс користувача з відкритою закладкою «Stat» може виглядати так, як показано на рисунку 3.10.

Далі слід переключитися в режим редагування коду і знайти рядок коду, який створює об’єкт statisticsManager.

У наступному рядку слід написати код, що передає цьому компоненту посилання на фабрику моделей, який може виглядати приблизно так (з точністю до назви класу моделі):

```
statisticsManager.setFactory((d)->new BuldModel(d, this));
```

Тут використання лямбда функції дозволяє не створювати клас для фабрики моделей з методом createModel(Dispatcher d). Фабрика потрібна компоненту statisticsManager для створення моделі.

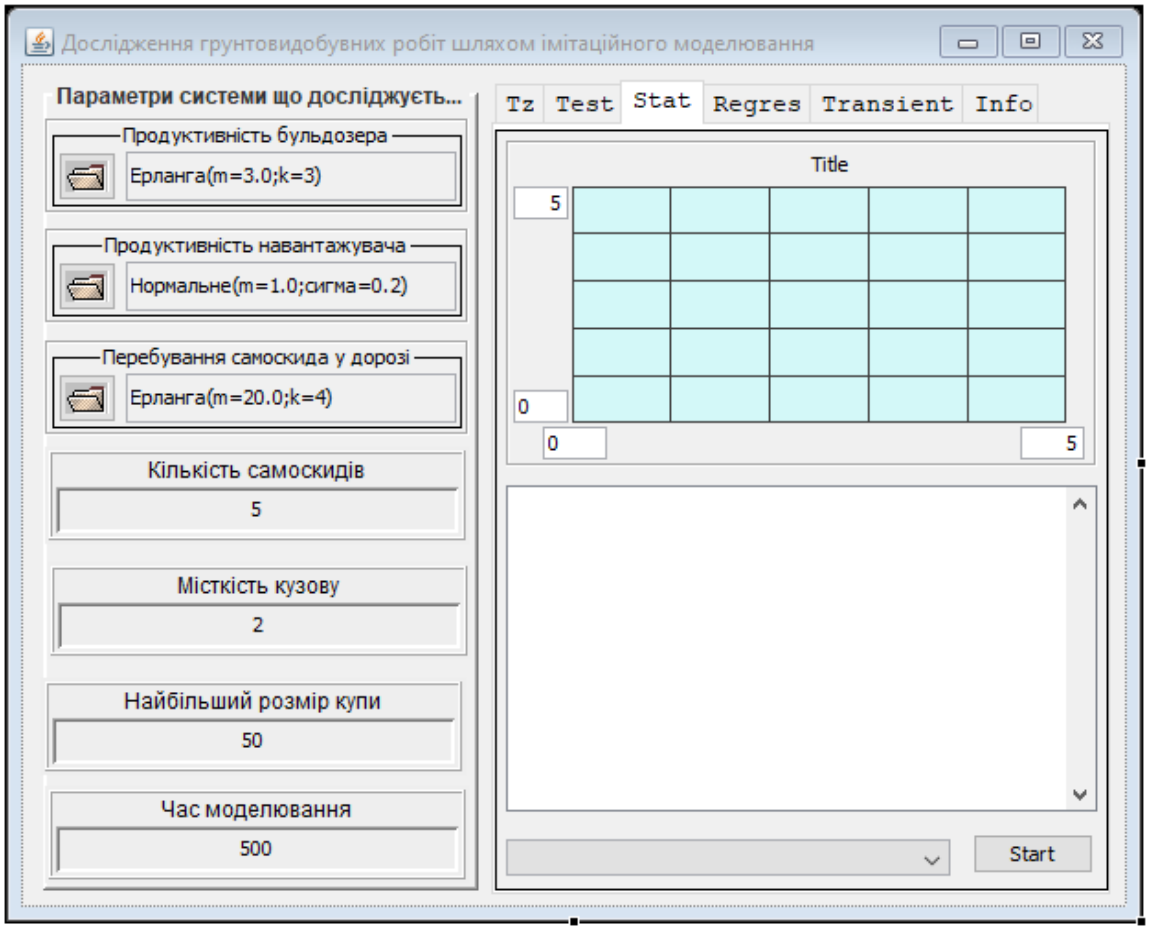

Рисунок 3.10 – Інтефейс користувача із закладкою «Stat»

3.5.1.2 Реалізація інтерфейсу IStatisticsable в моделі

Для того, щоб компонент StatisticsManager мав можливість працювати з моделю, вона має реалізувати інтерфейс IStatisticsable.

Тому слід відкрити клас моделі і прописати в заголовку класу заявку на реалізацію цього інтерфейсу.

Після цього в режимі виправлення помилки створити заготовки методів, реалізації яких вимагає цей інтерфейс.

Інтерфейс IStatisticsable потребує реалізації двох методів.

Метод initForStatistics() у застосуванні не потрібен, тому залишається пустим.

Метод getStatistics() має повертаати асоціативний масив (мапу), який містить посилання на гістограми з накопиченою статистикою. В якості ключів мапи використовуються назви гістограм.

Саме ці дані компонент StatistcsManager виводить на свої панелі у графічному та текстовому вигляді.

У лістингу 3.21 наведено приклад реалізації такого методу.

Лістинг 3.21 – Метод, що повертає статистичні дані про роботу моделі

```
public Map<String, IHisto> getStatistics() {
  Map<String, IHisto> map = new HashMap<>();
  map.put("Гістограма для довжини черги до навантажувача",
      getHistoForQueueToLoader());
  map.put("Гістограма для розмірів купи", getHistoHeap());
  map.put("Гістограма для часу простою бульдозера", getHistoBuldo());
  map.put("Гістограма для часу простою самоскида", getHistoLorry());
  map.put("Гістограма часу простою навантажувача", getHistoLoader());
  return map;
}
```

3.5.1.3 Тестування моделі

В режимі «Stat» треба встановлювати достатньо великий час роботи моделі для того щоб перехідні процеси в системі не впливали суттєво на статистичні дані. Окрім того, налаштування моделі мають бути такими, щоб система працювала в сталому режимі, тобто розміри черг не повинні нескінченно зростати.

Для вибору потрібної гістограми використовується випадаючий список.

3.6 Етап 6

На цьому етапі ми реалізумо режим роботи “Regres”, в якому з моделлю можна проводити одноріневі та багаторівневі однофакторні експерименти по вивченню впливу заданого фактору на показники роботи моделі.

На цьому етапі слід визначится з фактором, вплив якого буде досліджуватися. У моделі, яку ми розглядаємо в якості зразка, таким фактором є кількість самоскидів.

3.6.1.1 Реалізація закладки «Regres» у візуальній частині проєкту

Додаймо компонент ExperimentManager із фреймворку Simulation на JTabbedPane в якості закладки і змінимо назву закладки на «Regres».

Після цього інтефейс користувача з відкритою закладкою «Regres» може виглядати так, як показано на рисунку 3.11.

Далі слід налаштувати компонент ChooseData менеджера, через який задаються значення фактора. Цей компонент знаходиться у правому нижньому куті менеджера. Треба його виділити і налаштувати властивості title і text. Тобто написати назву фактора і задати перелік адекватних значень цього фактора.

Далі слід переключитися в режим редагування коду і знайти рядок, в якому створює об’єкт experimentManager.

У наступному рядку слід написати код, що передає цьому об’єкту

посилання на фабрику моделей, який може виглядати приблизно так (з точністю до назви класу моделі):

```
experimentManager.setFactory((d)->new BuldModel(d, this));
```

Тут лямбда функція фактично передає компонентові конструктор для створення моделі.

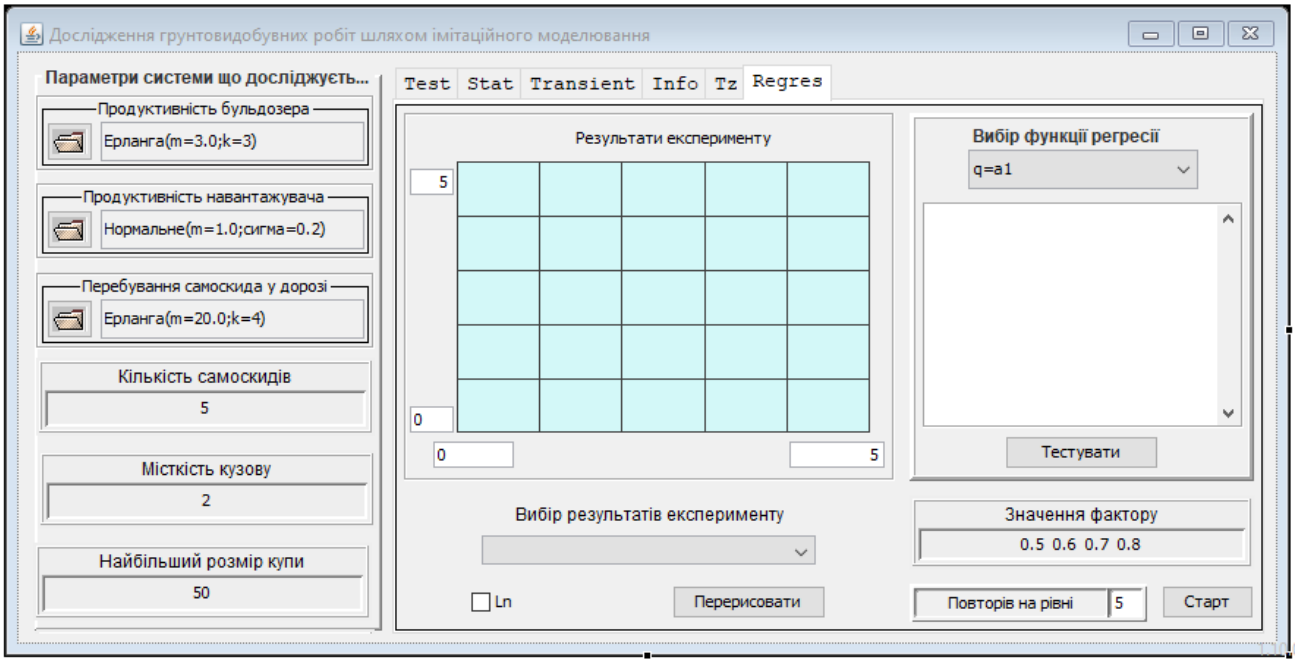

Рисунок 3.11 – Інтефейс користувача із закладкою Regres»

3.6.1.2 Реалізація інтерфейсу IExperimentable в моделі

Для того, щоб компонент ExperimentManager мав можливість працювати з моделю, вона має реалізувати інтерфейс IExperimentable.

Тому слід відкрити клас моделі і прописати в класі заявку на реалізацію цього інтерфейсу.

Після цього в режимі виправлення помилки створити заготовки методів, реалізації яких вимагає цей інтерфейс.

Інтерфейс IExperimentable потребує реалізації двох методів.

Метод initForExperiment(double) забезпечує передачу моделі значення фактору, вплив якого вивчається.

Вміст методу залежить від того, що є фактором в моделі. У прикладі, який ми розглядаємо в якості зразка, метод встановлює потрібну кількість авто у бригаді самоскидів, лістинг 3.22.

Лістинг 3.22 – Метод для передачі моделі значення фактору

```
public void initForExperiment(double factor) {
  multiLorry.setNumberOfClones((int) factor);
}
```

Другий метод цього інтерфейсу – getResultOfExperiment() повертає мапу з результатами експерименту. Таких результатів може бути декілька. Найчастіше це середні значення параметрів, які накопичувалися у гістограмах.

Ключами мапи є назви параметрів.

У лістингу 3.23 наведено такий метод для прикладу, який ми розглядаємо в якості зразка.

Лістинг 3.23 – Метод для повернення результатів експерименту

```
public Map<String, Double> getResultOfExperiment() {
  Map<String, Double> resultMap = new HashMap<>();
  resultMap.put("Час простою авто від їх кількості", getHistoLorry()
      .getAverage());
  resultMap.put("Розмір купи від кількості авто", getHistoHeap()
      .average());
  resultMap.put("Час простою бульдозера від кількості авто",
      getHistoBuldo().getAverage());
  resultMap.put("Час простою навантажувача від кількості авто",
      getHistoLoader().getAverage());
  return resultMap;
}
```

3.6.1.3 Тестування закладки

Перед початком експерименту слід визначитися із переліком значень фактора і кількістю експериментів на кожному рівні.

Далі натискаємо кнопку «Старт».

Після закінчення експерименту можна вибрати із списку потрібний результат і побачити на екрані сукупність експериментальних точок.

Далі можна вибрати функцію регресії для експериментальних даних і перевірити її на адекватність.

3.7 Етап 7

На цьому етапі ми реалізуємо режим “Transient” для дослідження перехідних процесів у чегах моделі. Реалізація цього завдання покладена на компонет типу TransProcessManager, який використовується для дослідження перехідних процесів у чергах. Компонент передбачає технологію, відповідно до якої моделі створюються, ініціалізуються та запускаються самим компонентом.

3.7.1.1 Реалізація закладки «Transient» у візуальній частині проєкту

Додаймо компонент TransProcessManager із фреймворку Simulation на JTabbedPane в якості закладки і змінимо назву закладки на «Transient».

Після цього інтефейс користувача з відкритою закладкою «Transient» може виглядати так, як показано на рисунку 3.12.

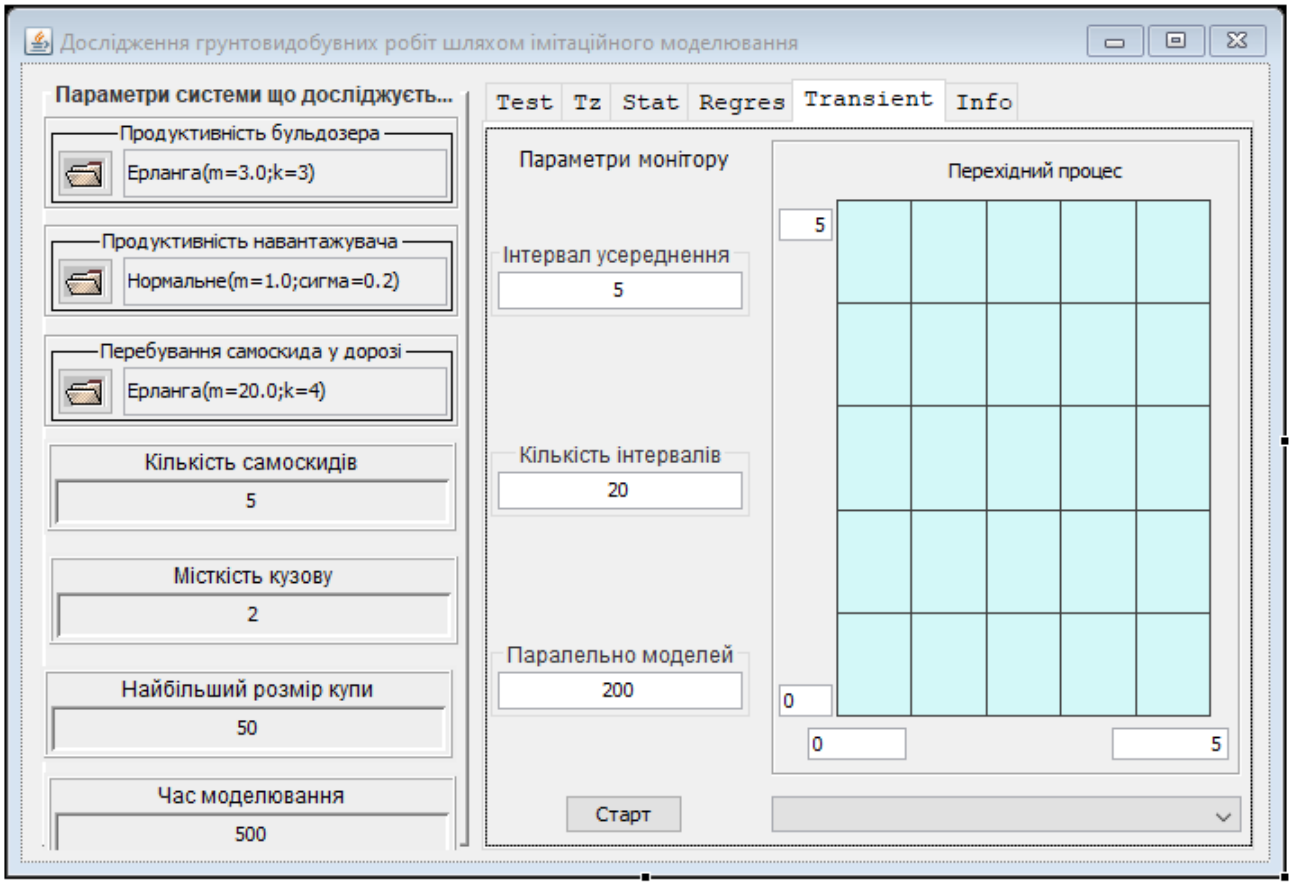

Рисунок 3.12 – Інтефейс користувача із закладкою «Transient»

Компоненту TransProcessManager необхідно передати посилання на фабрику моделей. Для цього слід переключитися в режим редагування коду і знайти рядок, в якому створює об’єкт transProcessManager.

У наступному рядку слід написати код, що передає цьому об’єкту посилання на фабрику моделей, який може виглядати приблизно так (з точністю до назви класу моделі):

transProcessManager.setFactory((d)-> new BuldModel(d, this)).

3.7.1.2 Реалізація інтерфейсу ITransPrcesable в моделі

Для співпраці з компонентом transProcessManager модель має реалізувати інтерфейс ITransPrcesable.

Тому слід відкрити клас моделі і прописати в заголовку класу заявку на реалізацію цього інтерфейсу.

Після цього в режимі виправлення помилки створити заготовки методів, реалізації яких вимагає цей інтерфейс.

Інтерфейс ITransPrcesable потребує реалізації двох методів.

Метод initForTrans(double) цього інтерфейсу, лістинг 3.24, забезпечує передачу моделі значення часу моделювання, який залежить від налаштувань

компоненту. У свою чергу модель має передати це значення візуальній частині, звідки час моделювання отримують актори моделі. Метод створено з припущенням, що у акторів реалізовано метод setFinishTime.

Лістинг 3.24 – Реалізація методу підготовки до дослідження перехідних процесів

```
public void initForTrans(double finishTime) {
  getBuldo().setFinishTime(finishTime);
  getLoader().setFinishTime(finishTime);
  getLorry().setFinishTime(finishTime);
  gui.getChooseDataFinishTime().setInt((int)finishTime);
}
```

Другий метод інтерфейсу ITransPrcesable – getTransResult() повертає мапу з посиланнями на черги, які підлягають дослідженню.

Ключами мапи є назви черг.

У лістингу 3.25 наведено такий метод для прикладу, який ми розглядаємо в якості зразка.

Лістинг 3.25 – Реалізація методу доступу до черг

```
public Map<String, Double> getTransResult() {
     Map<String, Double> transMap = new HashMap<>();
     transMap.put("Купа грунту", getHeap());
     transMap.put("Черга до навантажувача", getQueueToLoader());
     transMap.put("Самоскиди на шляхах", getQueueLorryOnRoad());
     return transMap;
}
```

3.7.1.3 Тестування закладки

Перед початком експерименту слід визначитися із довжиною інтервалу накопичення і кількістю цих інтервалів.

Далі натискаємо кнопку «Старт».

Після закінчення експерименту можна вибрати із списку потрібну чергу і побачити на екрані сукупність точок, які описують процес зміни середньої довжини черги.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Томашевський В.М. Моделювання систем. – К.: Видавнича група BHV, 2005. -352 с.:іл.
2. Стеценко, І.В. Моделювання систем: навч. посіб. [Електронний ресурс, текст] / І.В. Стеценко ; М-во освіти і науки України, Черкас. держ. технол. ун-т. – Черкаси : ЧДТУ, 2010. – 399 с

3. Теорія ймовірностей та математична статистика: навчальний посібник / О. І. Огірко,. Н. В. Галайко. – Львів: ЛьвДУВС, 2017. – 292 с.
4. Моделювання систем. Конспект лекцій з дисципліни «Моделювання систем» для студентів спеціальності 123 – „Комп'ютерна інженерія”. /Укл.: Бивойно П.Г. - Чернігів: НУ «Чернігівська політехніка», 2023. - 139 с.
5. Моделювання систем на Java. Методичні вказівки до лабораторного практикуму та самостійної роботи з дисципліни «Моделювання систем» для студентів спеціальності 123 – „Комп'ютерна інженерія”. /Укл.: Бивойно П.Г., Пріла О.А., Бивойно Т.П. - Чернігів: ЧНТУ, 2024. - 146 с.
6. ДСТУ 3008:2015. ЗВІТИ У СФЕРІ НАУКИ І ТЕХНІКИ. Структура та правила оформлювання. Київ. ДП «УкрНДНЦ». 2016