МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чернігівський національний технологічний університет

МОДЕЛЮВАННЯ СИСТЕМ МАСОВОГО ОБСЛУГОВУВАННЯ

МЕТОДИЧНІ ВКАЗІВКИ
до виконання розрахунково-графічних робіт
та самостійної роботи з дисципліни
«Моделювання систем»
для студентів спеціальності
123 –"Комп'ютерна інженерія"

ЗАТВЕРДЖЕНО
на засіданні кафедри
інформаційних та комп'ютерних систем
протокол № 1 від 29.08.18

Чернігів ЧНТУ 2018

Моделювання систем масового обслуговування. Методичні вказівки до виконання розрахунково-графічних робіт та самостійної роботи з дисципліни «Моделювання систем» для студентів спеціальності 123 – „Комп’ютерна інженерія”. /Укл.: Бивойно П.Г., Пріла О.А – Чернігів: ЧНТУ, 2018. – 76 с.

Укладачі: Бивойно Павло Георгійович, канд. техн. наук, доцент
Пріла Ольга Анатоліївна, канд. техн. наук, доцент

Відповідальний за випуск: С.В. Зайцев, доктор. техн. наук, зав. кафедри інформаційних та комп'ютерних систем Чернігівського національного технологічного університету

Рецензент: В.А. Бичко, канд. ф-м. наук, доцент кафедри інформаційних та комп’ютерних систем Чернігівського державного технологічного університету

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.

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.

Роботу виконує команда з трьох студентів.

Дизайнер перш за все розробляє перелік публічних методів (програмний інтерфейс) для шару подання і після цього створює візуальну частину проекту.

Team leader перш за все розробляє перелік публічних методів (програмний інтерфейс) та створює клас моделі, а також займається тестуванням проекту і узгодженням питань між членами команди.

Кодер створює класи для компонентів.

Кожен з членів команди готує звіт по своїй частині роботи. Об’єднує звіти керівник команди.

Результатом розрахунково-графічної роботи є звіт обсягом приблизно 30 сторінок друкованого тексту оформленого відповідно до стандартів кафедри. Бали за розрахунково-графічну роботу виставляються з урахуванням своєчасності та якості виконання, а також особистого внеску у розробку проекту.

За звичай номер варіанту завдання для РГР призначається викладачем, та студент може і сам обрати предметну область для моделювання, але завдання обов'язково має бути погоджено з викладачем

1 ЗАВДАННЯ ДО РОЗРАХУНКОВО - ГРАФІЧНОЇ 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 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.

3. 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.

4. 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 не пройшли тестування відправляються на ділянку налагодження, після чого знову проходять тестування. У тому випадку, якщо блок не проходить тестування повторно, він відправляється у брак. 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.

5. 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.

6. 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.

7. 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.

8. 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.

9. 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. Кожна бригада обслуговує одну баржу. В автомобіль міститься кілька контейнерів. Автомобілі розвозять кавуни по різних районах міста. Якщо автомобілів нема, контейнери вивантажуються на розвантажувальну площадку обмеженого розміру. Якщо баржі під розвантаженням нема, автомобілі завантажуються контейнерами із площадки.

10. 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.

11. 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..

12. Моделювання виборів, варіант Election

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

13. 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.

14. Моделювання кільцевої конвеєрної лінії, варіант Circle

Досліджувана система складається з декількох 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.

15. 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.

16. 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.

17. 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.

18. 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.

19. Моделювання роботи ресторану, варіант Restoran

У ресторан приходять клієнти й сідають за вільні столики. Інтенсивність появи клієнтів міняється протягом дня. 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.

20.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. При кожній зустрічі із хворим лікар із деякою ймовірністю може направити хворого до іншого лікаря, у цьому випадку хворий до першого лікаря вже не повертається.

21. 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.

22. 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.

23. 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.

24. 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.

25. Моделювання роботи технічного ракетного 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.

26. 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.

27. 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.

28. 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.

29. 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.

30. 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.

31. 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.

32. 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.

33. Моделювання дороги студента до університету, варіант Student’sWay

Деяка кількість студентів університету проживає в одному мікрорайоні міста. Студенти їдуть на заняття в першу зміну (на першу пару) або в другу (на четвертую пару), і приходять на зупинку за деякий час до початку пари. З цього мікрорайону до університету можна доїхати одним видом тролейбуса, який приходить на зупинку з певною періодичністю. Існує вірогідність того, що тролейбус, що прийшов, буде переповнений і студентові доведеться чекати наступного тролейбуса, причому наповненість тролейбуса з ранку значно вища, ніж в другій половині дня.

Трапляється, що тролейбус в дорозі зламався, тоді студент чекатиме, доки тролейбус полагодять, або якщо раніше прийде наступний тролейбус, то спробує сісти в нього.

Під'їхавши до зупинки університету, студент витрачає якийсь час на дорогу до університету. Зайшовши в університет, він займає чергу в роздягальню (йде в те вікно, де черга менша). Залишивши одяг в роздягальні, студент йде і стає в чергу до ліфта, щоб піднятися на верхній поверх. Ліфт може вміщати обмежену кількість людей. Існує вірогідність того, що до ліфта підійде хтось з начальства, тоді студентові доведеться йти пішки.

34. 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.

35. 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.

36. 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.

2 ВИМОГИ ДО РОБОТИ

2.1 Створення команди та розподіл обов'язків

Особливість роботи полягає у тому, що вона має бути розроблена командою. Студенти мають об'єднатися у команди по 3 особи. Кожна команда обирає із свого складу керівника (team leader). Розподіл ролей і зон відповідальності між членами команди орієнтовно може бути таким:

– керівник команди організує роботу команди, розробляє технічне завдання, визначає об'єкти, що мають входити до моделі, розподіляє роботу між членами команди. За звичай керівник реалізує модель системи і таким чином контактує і з розробником візуальної частини і з розробником класів для акторів. Він же виконує тестування проекту.

– член команди – дизайнер, розробляє графічний інтерфейс користувача та пов'язує його з моделлю.

– член команди – кодер, розробляє правила дії об'єктів-акторів і створює класи для них.

Кожен з членів команди готує свою частину пояснювальної записки. Керівник команди об'єднує частини до єдиного звіту.

Робота має виконуватися на протязі семестру.

Виконання роботи починається з детального аналізу завдання, визначаються абстракції предметної області, формується перелік класів, що мають бути створені та складається графік виконання робіт з переліком контрольних точок (milestones).

Команда має періодично проводити збори. На зборах кожний член команди доповідає про виконану роботу і про проблеми, що виникли. Команда обговорює стан справ і приймає необхідні рішення. Стан справ по роботі в цілому і по кожному виконавцю, а також проблеми, що виникли і прийняті рішення обговорюються з викладачем під час консультацій.

2.2 Обов'язкові розділи пояснювальної записки

Пояснювальна записка до роботи створюється відповідно до вимог кафедри викладених у методичних вказівках [1] (СОККР-2002). Наведені нижче пункти можна вважати доповненням до вимог СОККР-2002.

2.2.1 Технічне завдання

У технічному завданні має бути наведено опис системи, для якої створюється модель.

Обумовлені спрощення та обмеження, що будуть прийняті при створенні моделі.

Докладно викладені цілі моделювання, які команда має визначити сама, виходячи з опису системи та обраного рівня складності проекту.

Визначено, які режими роботи мають бути передбачені у проекті та вимоги до інтерфейсу користувача.

Як мінімум проект має забезпечити реалізацію таких функцій:

– тестові запуски моделі при різних налаштуваннях системи з індикацією стану основних компонент системи;

– проведення експериментів задля отримання статистичних характеристик для черг системи та часу чекання активних об’єктів системи.

– проведення багаторівневих експериментів для визначення впливу найбільш вагомих факторів на показники роботи системи;

– дослідження перехідних процесів для черг, що сторюються у системі.

2.2.2 Аналіз системи що підлягає моделюванню

У цьому розділі слід перш за все навести схематичне зображення предметної області. Ц можна зробити у вигляді діаграми бізнес процесів, як це зроблено у прикладі, який наведено у додатку А, але можна використовувати будь який спосіб зображення.

Розділ має складатися з трьох підрозділів.

Перший підрозділ має бути зорієнтований на виділення абстракцій системи.

У другому підрозділі слід проаналізувати поведінку активних абстракцій

Третій підрозділ передбачає аналіз фрейморку Simulation на предмет використання його класів для реалізації абстракцій системи.

2.2.2.1 Виділення основних абстракцій системи

У цьому розділі слід провести аналіз системи, що підлягає моделюванню, і визначити так звані абстракції, 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.

Результатом цього етапу має бути перелік абстракцій. Доцільно це зробити у вигляді таблиці, де навести назви абстракцій та перелік завдань, що вони вирішують.

2.2.2.2 Аналіз поведінки активних об’єктів системи

У цьому підрозділі слід проаналізувати поведінку активних абстракцій та представити її у вигляді схеми алгоритму, або діаграми діяльності.

Окрім того слід визначити, яка інформація потрібна абстракції для нормального функціонування і представити цю інформацію у вигляді таблиці.

2.2.2.3 Аналіз можливостей фреймворку для побудови моделі

У цьому підрозділі необхідно визначити, які абстракції системи можуть бути реалізовані засобами фреймворку Simulation, а також обґрунтувати необхідність створення нових класів або успадкування існуючих.

Результатом цього етапу має бути перелік класів та інтерфейсів, що будуть використані. Доцільно це зробити у вигляді таблиці, де навести назви

абстракцій та назви класів та інтерфейсів.

2.2.3 Реалізація системи

У цьому розділі слід докладно представити результати реалізації шару подання, шару моделі та шару складових частин моделі. Кожний з шарів має бути описаний у окремому підрозділі.

2.2.3.1 Реалізація шару подання

У цьому розділі пояснювальної записки слід дати зображення інтерфейсу користувача для можливих режимів роботи застосування і пояснити призначення його складових частин.

Далі слід перелічити вимоги до окремих компонентів та зв'язків між ними. При необхідності, дати перелік подій компоненту, що будуть оброблятися. Слід також обґрунтувати вибір менеджерів компоновки, що мають бути налаштовані у візуальній частині.

Обов'язково потрібно надати перелік публічних методів шару подання, що будуть надавати доступ до компонентів візуальної частини та метод створення моделі.

Слід також навести методи запуску процесу моделювання у запроектованих режимах роботи, а текст усього класу наводити не слід.

2.2.3.2 Реалізація шару моделі

У цьому розділі слід охарактеризувати клас моделі. Якщо текст класу добре прокоментований, то його можна навести повністю. У іншому випадку слід окремо навести конструктор та методи і дати пояснення до них.

2.2.3.3 Реалізація шару компонентів

Тут, у окремих підрозділах, слід охарактеризувати класи для компонент моделі, що створювалися. Якщо тексти класів добре прокоментовані, то їх можна наводити цілком. У іншому випадку слід так само, як і для моделі, окремо навести перелік полів, конструктор та методи і дати пояснення до них.

2.2.4 Результати тестування програми

У цьому розділі слід докладно представити результати тестування програми. Тестування проводиться з метою з'ясування працездатності моделі, адекватної реакції на зміну налаштувань та підтвердження можливості її використання для дослідження реальної системи.

Слід вибрати налаштування моделі, які б могли відповідати реаліям системи. Слід також визначити час моделювання, що забезпечує прийнятні результати визначення статистичних характеристик черг.

У цьому розділі має бути достатня кількість копі екранів із результатами роботи застосування у різних режимах.

3 МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ РОБОТИ

3.1 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.

Аналіз системи краще проводити усією командою. Для цього можна використати метод мозкового штурму. Слід відзначити, що продуктивність подальшої роботи команди над проектом суттєво залежіть від якості виконання цього етапу роботи. Все, що буде робитися далі, це суто технічна робота.

3.2 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 Simulation_2014.

3.2.1 Засоби для створення активних об’єктів моделі

3.2.1.1 КласActor

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.

3.2.1.2 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 Semaphore). Для того, щоб усунути цей недолік, у класі Actor перевизначений базовий метод clone(), у якому для кожного клону створюється власний об'єкт класу Semaphore.

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

3.2.2 Засоби для створення черг та накопичувачів

3.2.2.1 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.

3.2.2.2 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..

3.2.3 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.

3.2.4 Засоби для генерації випадкових величин

3.2.4.1 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 3.1.

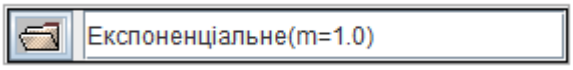

Рисунок 3.1 – Вигляд компонента ChooseRandom

При натисканні на кнопку відкривається комбобокс з переліком можливих розподілів, рисунку 3.2.

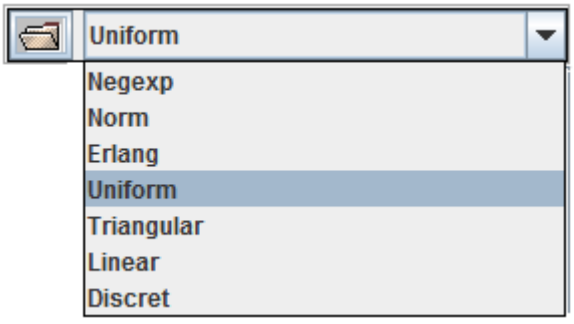

Рисунок 3.2 – Перелік можливих розподілів компонента ChooseRandom

Після вибору необхідного розподілу з’являється діалогове вікно для налаштувань параметрів вибраного розподілу, рисунок 3.3.

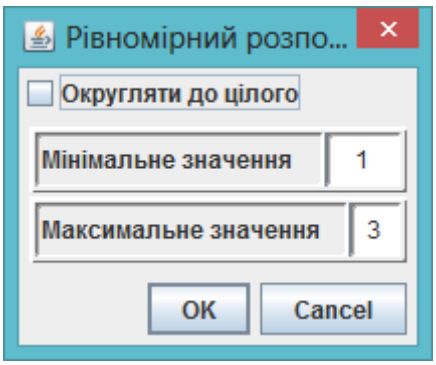

Рисунок 3.3 – Діалогове вікно для налаштування рівномірного розподілу компоненту ChooseRandom

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

– next() - повертає наступне випадкове число;

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

3.2.5 Компоненти для створення інтерфейсу користувача та відображення результатів моделювання

3.2.5.1 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 3.4.

Рисунок 3.4 – 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.

3.2.5.2 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 3.5.

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 3.5 – 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.

3.2.6 Компоненти, що спрощують проведення експериментів та відображення результатів моделювання

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

3.2.6.1 Компонент StatisticsManager

Цей компонент забезпечує запуск моделі у режимі отримання статистичних даних та відображає отримані статистичні дані. Зовнішній вигляд компоненту наведено на рисунку 3.6.

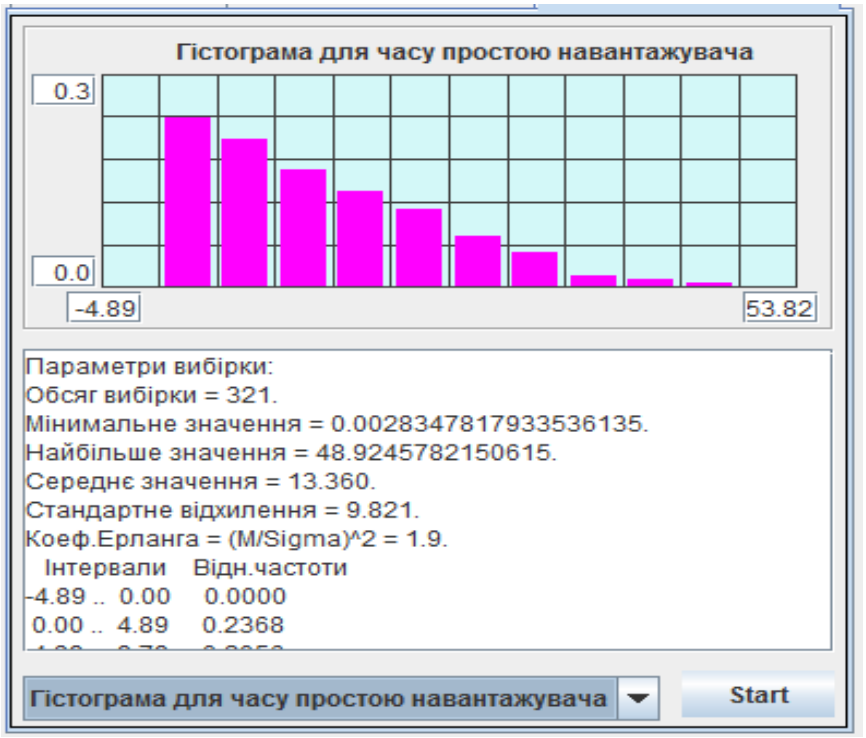

Рисунок 3.6 – Вигляд компоненту класу StatisticsManager

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.

3.2.6.2 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 3.7.

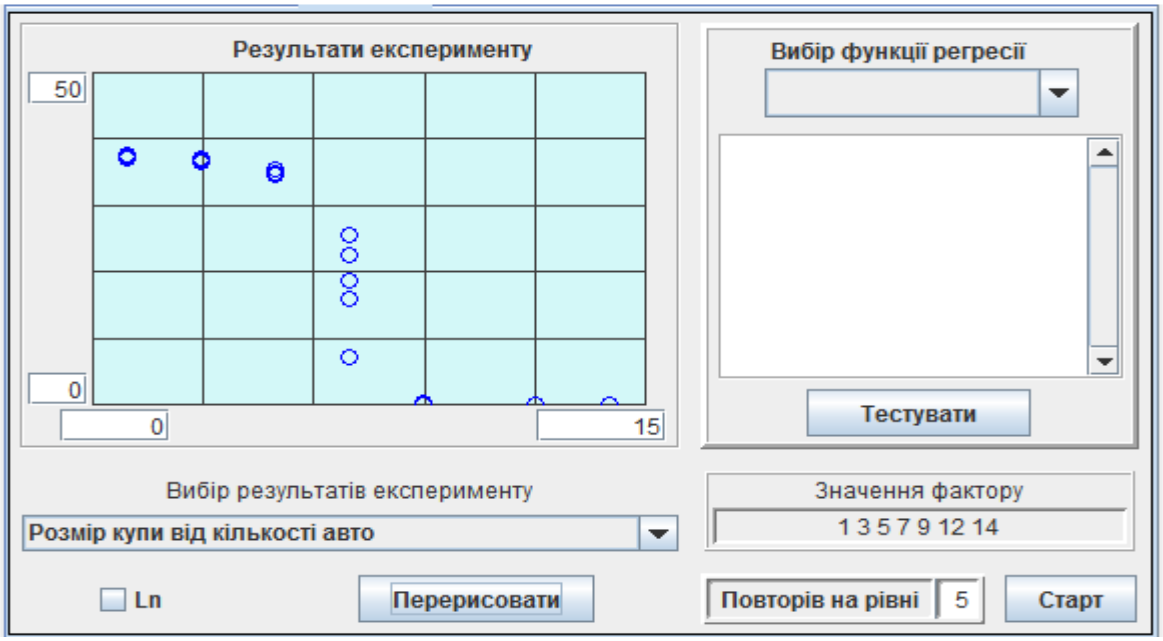

Рисунок 3.7 – Вигляд компоненту класу ExperimentManager

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).

3.2.6.3 Компонент TransProcessManager

Цей компонент забезпечує дослідження перехідних процесів у чергах. Зовнішній вигляд компоненту наведено на рисунку 3.8.

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

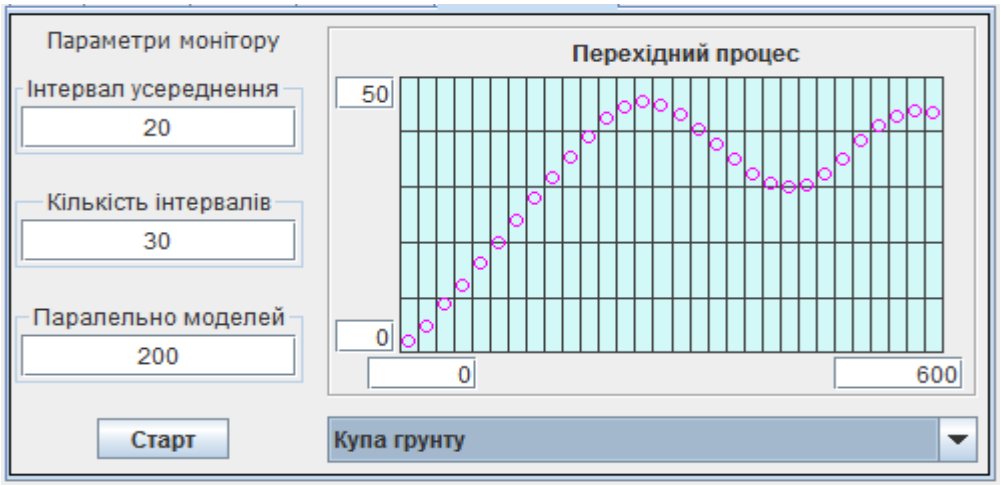

Рисунок 3.8 – Вигляд компоненту класу TransProcessManager

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 3.8, 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.

3.3 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 3.9.

3.3.1 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 3.9 – 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 3.2.6.1 – 3.2.6.3.

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.

3.3.2 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.

Реалізуючи останній пункт доцільно використовувати конструктори з параметрами.

3.3.3 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.

Рекомендована література

1. Структура і оформлення кваліфікаційних та курсових робіт. Методичні вказівки для студентів професійного спрямування «Комп'ютерна інженерія» / /Укл.: А.І.Вервейко, С.О., Нестеренко, Є.В.Нікітенко – Чернігів: ЧДТУ, 2001. – 28с.
2. Томашевський В.М. Моделювання систем. – К.: Видавнича група BHV, 2005. -352 с.:іл.
3. Лоу А.М., Кельтон В.Д. Имитационное моделирование. - 3-е изд. - СПб.: Питер, 2004. – 847с.

4. РГР з моделювання. Методичні вказівки до виконання розрахунково-графічних робіт з дисципліни «Моделювання» для студентів напряму підготовки 6.050102 – „Комп'ютерна інженерія". /Укл.: Бивойно П.Г., Пріла О.А, Бивойно Т.П. – Чернігів: ЧДТУ, 2012. – 64 с.
5. Імітаційне моделювання паралельних процесів. Методичні вказівки до лабораторного практикуму та самостійної роботи з дисципліни "Моделювання" для студентів напряму підготовки 6.050102 "Комп'ютерна інженерія". Укладачі Бивойно П.Г., Павловський В.І. - Чернігів, ЧДТУ, 2008. - 54 с.

Додаток А

ПРИКЛАД ОФОРМЛЕННЯ РОБОТИ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАИНЫ
ЧЕРНІГІВСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ
КАФЕДРА ІНФОРМАЦІЙНИХ ТА КОМП'ЮТЕРНЫХ СИСТЕМ

МОДЕЛЮВАННЯ ГРУНТОВИДОБУВНИХ РОБІТ

Розрахунково-графічна робота з дисципліни "Моделювання "

Виконавці О.Ю. Зайко
студенти гр. КІ-111 А.І. Самко
В.І. Грищенко

Керівник
к.т.н., доцент П.Г. Бивойно

2018

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання розрахунково-графічної роботи з дисципліни «Моделювання»

Тема: Моделювання робіт з видобування ґрунту, Варіант «Buldo»

Виконавці: Зайко О.Ю., Самко А.І., Грищенко В.І. гр.КС-111.

Опис системи

Система складається з одного бульдозера, навантажувача та кількох самоскидів. Бульдозер порціями згрібає ґрунт у купу. На одну порцію ґрунту він витрачає випадковий час. Навантажувач починає працювати коли у купі є ґрунт і є самоскид, що чекає завантаження. Він порціями насипає ґрунт з купи у кузов самоскиду. Місткість кузову самоскида декілька порцій ґрунту. Самоскиди виїжджають із автопарку и стають у чергу до навантажувача. Після завантаження самоскиди відвозять ґрунт замовнику, розвантажуються, і повертаються до навантажувача.

Завдання на проектування

Створити Java застосування для імітаційного моделювання робіт з видобування ґрунту, яке дозволить:

- налаштування параметрів моделі, а саме: тривалість моделювання, кількість самоскидів, місткість кузову, закони розподілення для випадкових величин продуктивності навантажувача і бульдозера та часу перебування самоскида у дорозі;
- проведення тестових запусків моделі при різних налаштуваннях з динамічною індикацією розміру купи, довжини черги самоскидів до навантажувача, кількості самоскидів у дорозі, та можливості виведення протоколу роботи моделі під час тестових запусків;
- проведення експериментів для отримання статистичних характеристик для довжини черг та часу простою бульдозера, навантажувача та самоскидів.
- отримання залежностей для середніх значень вказаних вище параметрів від кількості самоскидів;
- отримання перехідних процесів для середніх значень довжини кожної з черг.

Обсяг текстової документації

Робота обсягом 25-30 с. формату А4

Орієнтовна трудомісткість роботи – 36 людино-годин.

Дата представлення роботи – останній тиждень 2-го семестру

Керівник роботи
доцент Бивойно П.Г.
«4» березня 2018 г.

Зміст

1 АНАЛІЗ СИСТЕМИ, ЩО ПІДЛЯГАЄ МОДЕЛЮВАННЮ

1.1 Опис системи

Система, рисунок 1.1, складається з одного бульдозера, навантажувача та кількох самоскидів. Бульдозер порціями згрібає ґрунт у купу. На одну порцію ґрунту він витрачає випадковий час. Навантажувач починає працювати коли у купі є ґрунт і є самоскид, що чекає завантаження. Він порціями насипає ґрунт з купи у кузов самоскиду. Місткість кузову самоскида декілька порцій ґрунту. Самоскиди виїжджають із автопарку и стають у чергу до навантажувача. Після завантаження самоскиди відвозять ґрунт замовнику, розвантажуються, і повертаються до навантажувача.

Рисунок 1.1 – Грунтовидобувні роботи

1.2 Виділення основних абстракцій системи

Систему, що підлягає дослідженню шляхом моделювання, можна схематично представити не тільки увигляді рисунку 1.1, але й у вигляді діаграми бізнес процесів, що наведена на рисунку 1.2.

Аналізуючи опис системи, наведений у технічному завданні і наведену діаграму, перш за все можна виділити таку абстракцію, як «бульдозер».

Абстракція «бульдозер» моделює робочу машину, що додає до купи порції ґрунту, збільшуючи таким чином її розмір. Одноразова порція ґрунту в реальній системі, звичайно, є випадковою величиною. Але купа ґрунту накопичує ці порції, тобто інтегрує їх, і таким чином зменшує вплив коливань розміру порції. Тому, для спрощення моделі будемо вважати, що порція ґрунту має стале значення і дорівнює одиниці. Тобто одиницею виміру кількості ґрунту у системі буде середнє значення порції ґрунту, що бульдозер додає до

купи за один раз. На видобування порції ґрунту бульдозер витрачає деякий час, що є випадковою величиною. Найбільш вірогідно, що ця величина підпорядковується закону Ерланга. Бульдозер припиняє свою роботу, якщо розмір купи збільшується до деякого критичного розміру і відновлює роботу тільки після того, як розмір купи стає удвічі меншим за критичний розмір.

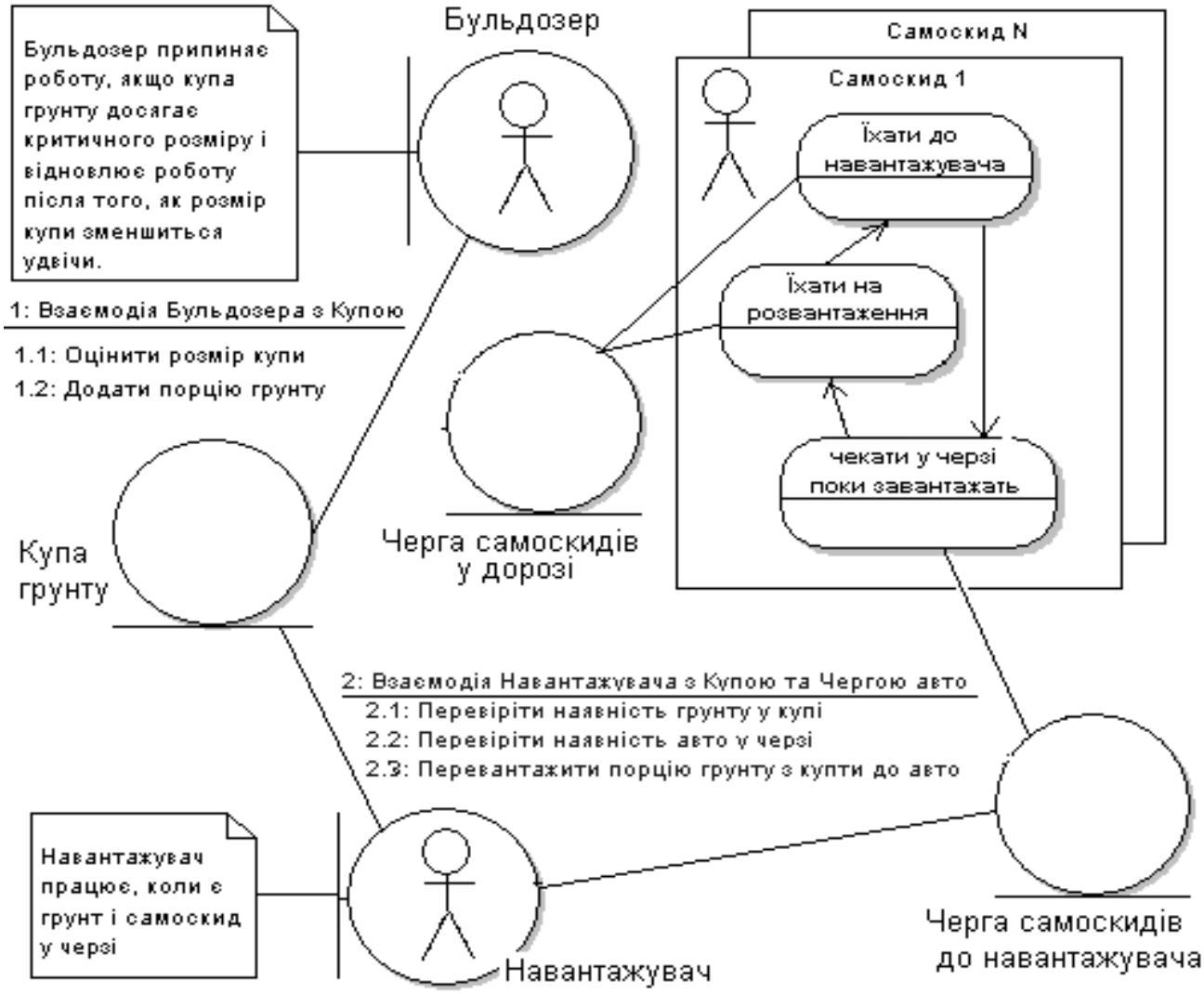

Рисунок 1.2 – Діаграма бізнес процесів у системі

Якщо для аналізу нашої системи використовувати терміни СМО, то бульдозер можна розглядати як гегнератор транзакцій, а порції грунту – як транзакції.

Відповідно до завдання для абстракції «бульдозер» необхідно накопичувати статистичну інформацію про час простою. Це завдання покладемо на абстракцію «Накопичувач інформації про час простою бульдозера».

Абстракція «бульдозер» у системі безпосередньо пов’язана абстракцією «купа ґрунту». Характеристикою цієї абстракції є її розмір. Цей параметр є індикатором процесів, що відбуваються у системі, бо його значення є результатом взаємодії усіх елементів системи. Відповідно до технічного

завдання необхідно забезпечити динамічну індикацію зміни цього параметру у часі а також накопичувати статистичну інформацію про зміну розміру купи. Це завдання покладемо на абстракцію «Накопичувач інформації про розмір купи».

Абстракція «навантажувач» моделює робочу машину, що навантажує ґрунт у самоскиди, беручи його з купи. Будемо вважати, що розмір порції, що навантажувач бере з купи за один раз дорівнює одинці. На завантаження порції ґрунту навантажувач витрачає деякий час, що є випадковою величиною. Найбільш вірогідно, що ця величина підпорядковується нормальному закону. Зрозуміло, що навантажувач може працювати коли є ґрунт у купі і самоскид, що чекає завантаження. Коли ж ці умови не виконуються, навантажувач чекає на їх виконання. Абстракція «навантажувач» у системі безпосередньо пов'язана з абстракцією «купа ґрунту» та абстракцією «самоскид».

Відповідно до завдання для абстракції «навантажувач» необхідно накопичувати статистичну інформацію про час простою цього об'єкту. Це завдання покладемо на абстракцію «Накопичувач інформації про час простою навантажувача».

Абстракція «самоскид» моделює авто, що перевозить ґрунт від купи до місця призначення. У системі може працювати одразу декілька самоскидів. Самоскид може перевозити декілька порцій ґрунту. На доставку ґрунту та повернення до навантажувача самоскид витрачає деякий час, що є випадковою величиною. Найбільш вірогідно, що ця величина підпорядковується закону Ерланга. Абстракція «самоскид» у системі може перебувати у одному з трьох станів – дорога до навантажувача, чекання на завантаження, дорога до місця вивантаження ґрунту. Відповідно до вимог технічного завдання для цих станів має бути передбачена динамічна індикація. Окрім того необхідно накопичувати статистичну інформацію про час простою самоскидів. Це завдання покладемо на абстракцію «Накопичувач інформації про час простою самоскидів».

Абстракція «черга самоскидів до навантажувача» моделює чергу самоскидів, що чекають завантаження. Самоскиди стають у цю чергу, коли під'їжджають до навантажувача. Цю абстракцію використовує навантажувач, щоб визначити, чи є самоскиди, що потребують завантаження. Відповідно до вимог технічного завдання динаміка зміни цієї черги у часі має бути відображена на візуальній частині застосування.

Відповідно до завдання для абстракції «черга самоскидів до навантажувача» необхідно накопичувати статистичну інформацію про її довжину. Це завдання покладемо на абстракцію «Накопичувач інформації про довжину черги до навантажувача»

Абстракція «самоскиди у дорозі» моделює групу самоскидів, що знаходяться у дорозі. Ця абстракція необхідна для реалізації вимоги технічного завдання щодо динамічної індикації кількості самоскидів що перебувають у дорозі. Для накопичення статистичну інформацію про кількість самоскидів, що перебувають у дорозі, будемо використовувати абстракцію «Накопичувач інформації про самоскиди, що перебувають у дорозі»

Результати першого кроку аналізу системи по визначенню ключових абстракцій, наведені у таблиці 1.1.

Таблиця 1.1 – Абстракції системи, що входять до складу моделі

Абстракція	Перелік завдань
Бульдозер	Передавати через випадкові інтервали часу порції ґрунту до купи. Стежити за розміром купи. Якщо розмір купи більше прийнятного то припиняти роботу. Відновлювати роботу коли розмір купи зменшиться у два рази.
Накопичувач інформації про час простою бульдозера	Накопичувати інформацію про час простою бульдозера. Відображувати статистичні характеристики часу простою у графічному та текстовому вигляді.
Купа ґрунту	Накопичувати порції ґрунту від бульдозера. Відображати на діаграмі зміни розміру купи у часі.
Накопичувач інформації про розміри купи	Накопичувати інформацію про розміри купи. Відображувати статистичні характеристики довжини черги у графічному та текстовому вигляді.
Навантажувач	Чекати на появу ґрунту у купі і самоскида у черзі до навантажувача. Перевантажувати потрібну кількість порцій ґрунту із купи у самоскид, затрачуючи на це випадковий час.
Накопичувач інформації про час простою навантажувача	Накопичувати інформацію про час простою навантажувача. Відображувати статистичні характеристики часу простою у графічному та текстовому вигляді.
Самоскид	Перебувати у дорозі до навантажувача, ставати у чергу до навантажувача, чекати заповнення кузову, їхати на розвантаження і повертатися знов до навантажувача. Реєструватися у віртуальній чергах самоскидів, що їдуть.
Накопичувач інформації про час простою самоскидів	Накопичувати інформацію про час простою самоскидів. Відображувати статистичні характеристики часу простою у графічному та текстовому вигляді.
Черга самоскидів до навантажувача	Місце, де знаходяться самоскиди, що чекають на завантаження. Забезпечити динамічну індикацію змін свого розміру у часі.
Накопичувач інформації про довжину черги до навантажувача	Накопичувати інформацію про довжину черги до навантажувача. Відображувати статистичні характеристики довжини черги у графічному та текстовому вигляді.

Продовження таблиці 1.1

Абстракція	Перелік завдань
Самоскиди у дорозі	Динамічна індикація кількості самоскидів у дорозі.
Накопичувач інформації про кількість самоскидів у дорозі	Накопичувати інформацію про кількість самоскидів у дорозі. Відображувати статистичні характеристики довжини черги у графічному та текстовому вигляді.

До складу абстракцій додамо також таку абстракція, як модель системи у цілому, що об'єднує інші абстракції і є об'єктом експериментального дослідження під час моделювання.

1.3 Аналіз активних абстракцій системи

Активними абстракціями системи є бульдозер, навантажувач та самоскид. Розглянемо їх поведінку та представимо у вигляді діаграм діяльності.

1.3.1 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.

Для роботи бульдозеру необхідні дані, перелік яких наведено у таблиці 1.3.

Таблиця 1.3 – 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

Правила дії бульдозера схематично можна представити у вигляді діаграми діяльності, рисунок 1.3.

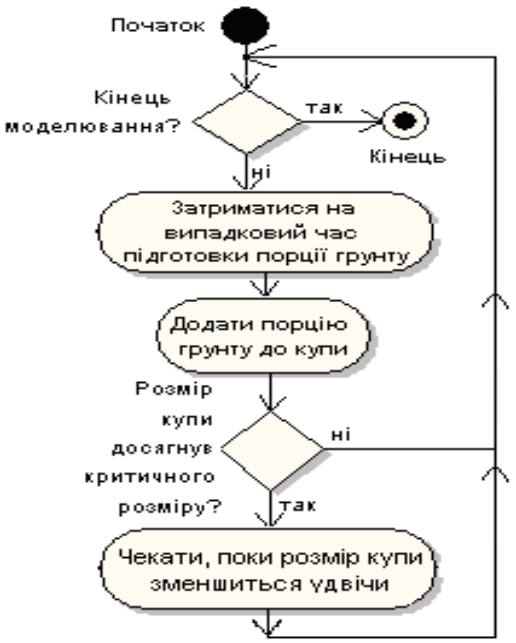

Рисунок 1.3 – Діаграма діяльності абстракції бульдозер

1.3.2 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.

Для роботи навантажувача необхідні дані, перелік яких наведено у таблиці 1.4.

Таблиця 1.4 – 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

Правила дії навантажувача схематично можна представити у вигляді діаграми діяльності, рисунок 1.4.

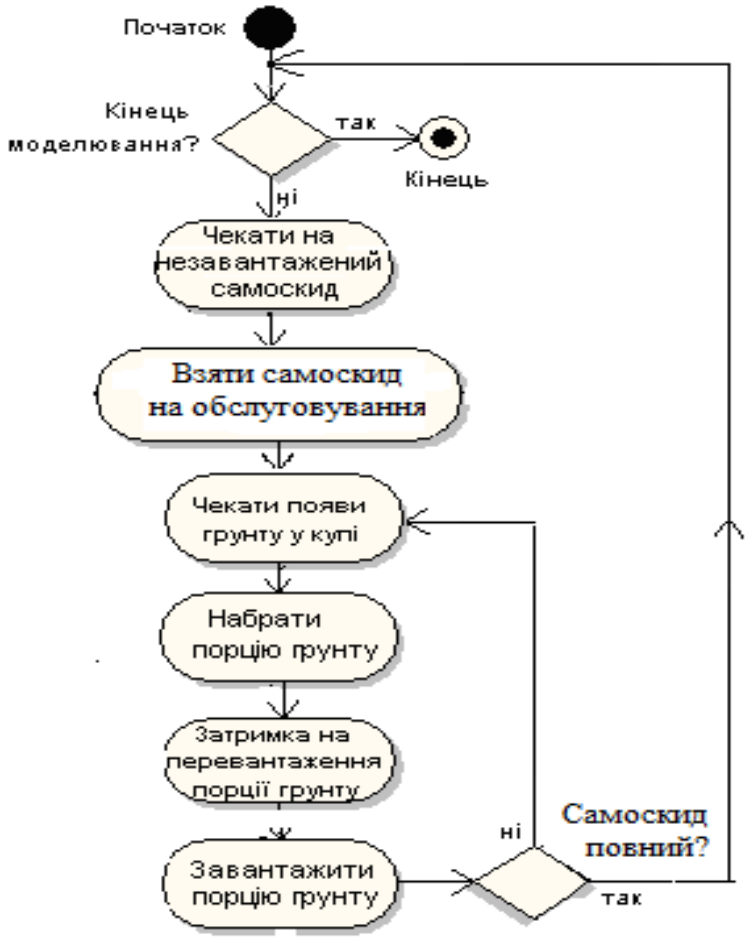

Рисунок 1.4 – Діаграма діяльності абстракції навантажувач

1.3.3 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.

Для роботи самоскиду необхідні дані, перелік яких наведено у таблиці 1.5.

Таблиця 1.5 – Атрибути абстракції Самоскид

Назва поля	Клас	Призначення поля	Джерело
queueToLoader	qusystem. QueueForTransaction	Посилання на чергу самоскидів, що чекають завантаження	Model
queueLorryOn Road	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

Правила дії самоскида схематично можна представити у вигляді діаграми діяльності, рисунок 1.5.

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.

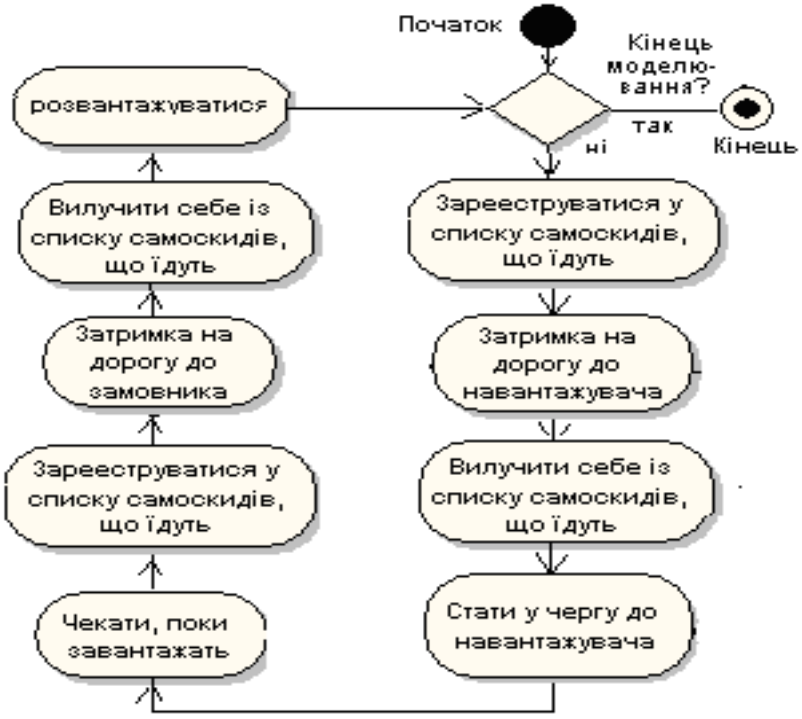

Рисунок 1.5 – Діаграма діяльності абстракції самоскид

1.4 Аналіз можливостей фреймворку Simulation для реалізації абстракцій системи

Ресурси фреймворку simulation дозволяють ефективно реалізувати абстракції системи у вигляді класів.

Перш за все розглянемо абстракції, що мають виконувати правила дії у часі. Такими абстракціями є бульдозер, навантажувач і самоскиди. Для моделювання таких абстракцій фреймворк Simulation містить абстрактний клас process.Actor, на основі якого можна створити класи, що реалізують особливості абстракцій бульдозер, навантажувач і самоскид. Для цього необхідно тільки визначити у класі-спадкоємці правила дії відповідної абстракції та її поля.

Для створення бригади самоскидів можна використати клас process.MultiActor.

Далі розглянемо абстракцію «купа ґрунту». Реалізувати цю абстракцію можна за допомогою звичайної змінної реального типу. У цьому випадку бульдозер буде додавати числа (порції ґрунту) до цієї змінної, а «навантажувач» буде зменшувати значення цієї змінної відповідно до розміру ковша навантажувача. Цей варіант досить точно відповідає реальності. Накопичувач такого типу реалізує клас process.Store. До того ж він забезпечить і динамічне відображення розміру купи ґрунту на діаграмі

Абстракції «черга самоскидів до навантажувача» та «самоскиди у дорозі», можна реалізувати як об’єкти класу queues.QueueForTransactions.

Абстракції «накопичувач інформації про довжину черги до навантажувача» та «накопичувач інформації про кількість самоскидів на шляху» можна реалізувати як об’єкти класу stat.DiscretHisto. Абстракцію «накопичувач інформації про розмір купи ґрунту» можна реалізувати як об’єкт класу stat.Histo. Ці гістограми треба передати відповідним чергам.

Абстракції типу «накопичувач інформації про час чекання або простою» можна реалізувати як об’єктикласу stat.Histo і передати відповідним акторам.

Результати цього етапу аналізу наведені у таблиці 1.6.

Таблиця 1.6 – Результати аналізу можливостей фреймворку simulation для реалізації абстракцій системи

Назва абстракції	Відповідний клас фреймворку simulation, його спадкоємець, або новий клас
1	2
Бульдозер	**Клас Buldo, спадкоємець класу Actor**
Навантажувач	**Клас Loader, спадкоємець класу Actor**
Самоскид	**Клас Lorry, спадкоємець класу Actor**
Бригада самоскидів	Клас MultiActor
Купа ґрунту	**Клас Store**

Продовження таблиці 1.6

1	2
Черга самоскидів до навантажувача	Клас QueueForTransaction
Самоскиди, що їдуть	Клас QueueForTransaction
Накопичувач інформації про розмір купи грунту	Клас Histo
Накопичувач інформації про розмір черги до навантажувача	Клас DiscretHisto
Накопичувач інформації про кількість самоскидів на шляху	Клас DiscretHisto
Накопичувач інформації про час простою бульдозера	Клас Histo
Накопичувач інформації про час простою навантажувача	Клас Histo
Накопичувач інформації про час чекання самоскидів	Клас Histo

2 РЕАЛІЗАЦІЯ СИСТЕМИ

2.1 Реалізація шару подання

Основою шару подання є інтерфейс користувача, який представлено на рисунках 2.1 – 2.5. Інтерфейс був створений відповідно до завдань, що були визначені вище. Інтерфейс спроектовано як сукупність декількох основних панелей.

Основою інтерфейсу є компонент SplitPanel.

Ліву частину цього компоненту займає панель для розміщення елементів налаштування моделі і присутня на екрані у всіх режимах роботи. У якості менеджера компоновки цієї панедлі вибрано GridLayout.

Праворуч розташований компонент TabbedPane, на закладках якого розташовані панелі, що з'являються після вибору відповідного режиму роботи.

2.1.1 Режим перегляду технічного завдання

Перша закладка, рисунок 2.1, містить текст технічного завдання у вигляді відображення HTML файлу. Сам файл tz.htm розташовано у папці з текстами класів застосування. Для розміщення тексту використовується компонент JTextPane, розташований на ScrollPane.

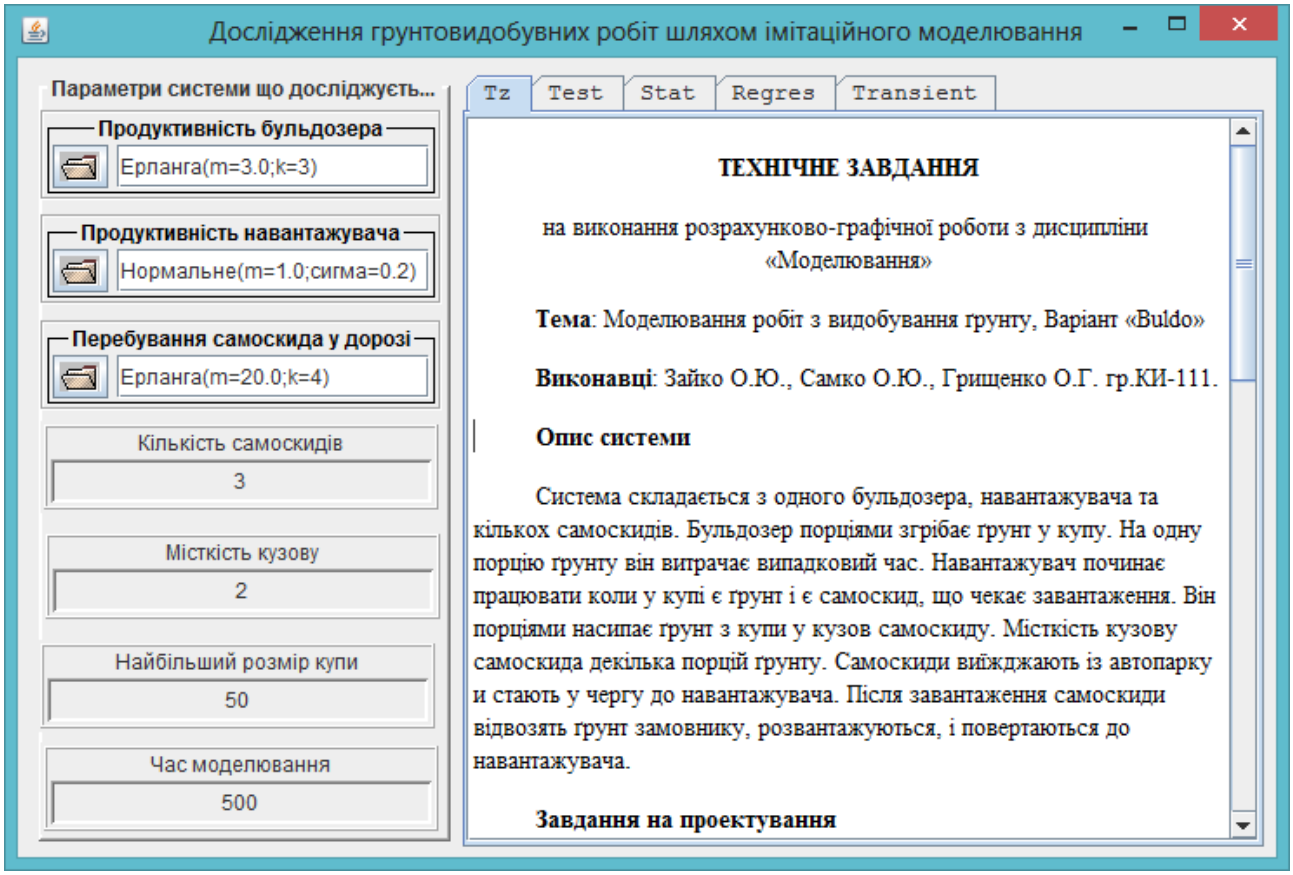

Рисунок 2.1 – Інтерфейс користувача моделі у режимі перегляду технічного завдання

Відображення файлу реалізується кодом, що наведено у лістингу 2.1.

Лістинг 2. 1 – Код для відображення тексту файлу на панелі

```
String str="/buldo2014/tz.htm";
URL url = getClass().getResource(str);
try {
   jTextPane.setPage(url);
} catch (IOException e) {
   System.err.println("Problems with file "+str);
}
```

2.1.2 Режим тестування моделі

Друга закладка, рисунок 2.2, використовується для тестування роботи моделі із динамічною індикацією зміни розмірів черг у часі та виведенням протоколу роботи моделі.

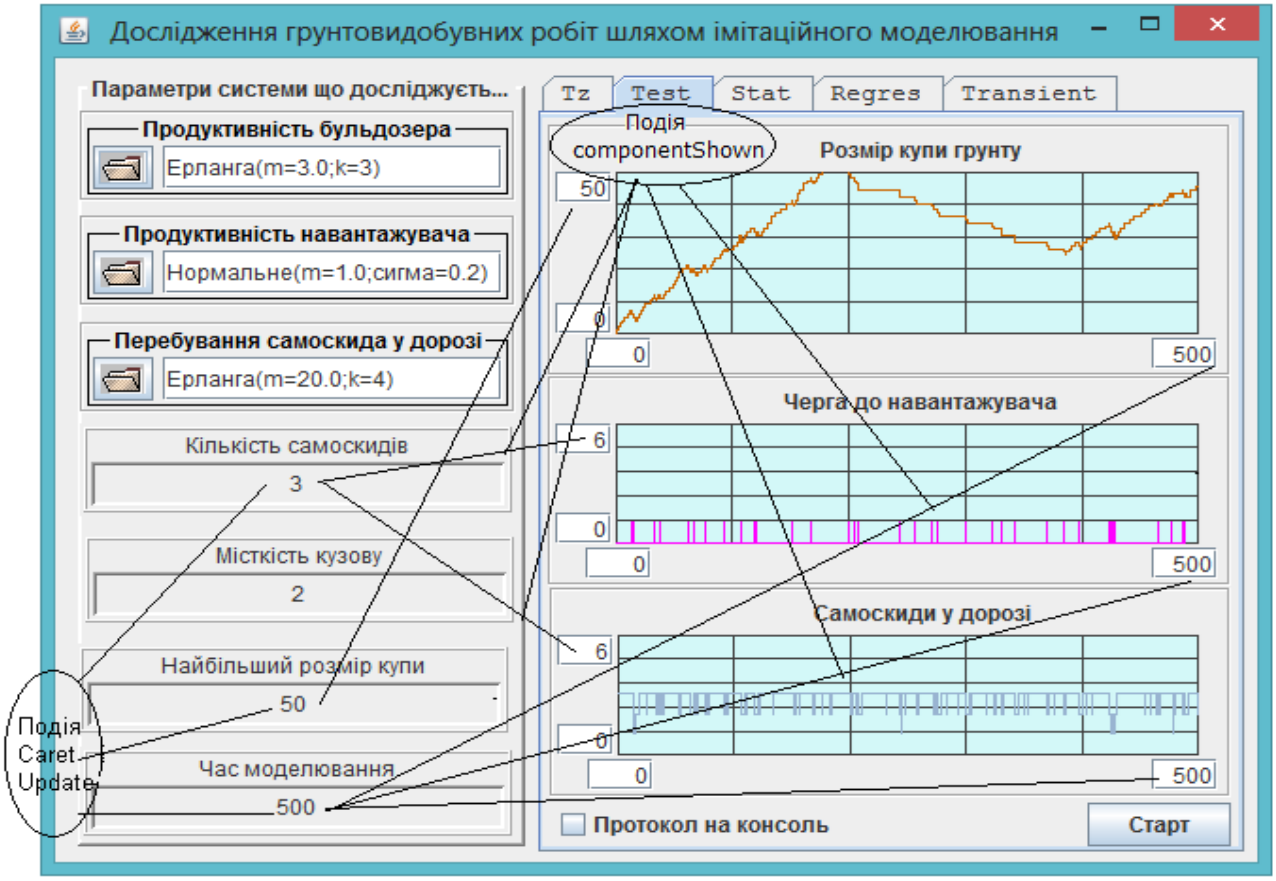

Рисунок 2.2 – Інтерфейс користувача моделі у режимі тестування

Для панелі застосовано менеджер компоновки GridBagLayout, що надає багато можливостей для розташування компонентів на панелі та дозволяє змінювати розміри панелі з не порушуючи її дизайн.

Для того, щоб налаштування діаграм відповідали налаштуванням моделі, події caretUpdate компонентів, що використовуються для налаштувань параметрів моделі, пов’язані з налаштуваннями діаграм.

З цією ж метою використовується і подія панелі componentShown.

Відповідні зв’язки схематично показані на рисунку 2.2.

Для запуску моделі у режимі тестування використовується кнопка «Старт», з якою пов’язано виклик методу startTest().

Текст методу наведено у лістингу 2.2.

Лістинг 2. 2 - Метод запуску процесу моделювання у режимі тестування

```
private void startTest() {
    getJButtonStart().setEnabled(false);
    getDiagramHeepSize().clear();
    getDiagramQueueToLoader().clear();
    getDiagramLorryOnRoad().clear();
    Dispatcher dispatcher = new Dispatcher();
    dispatcher.addDispatcherFinishListener(
        ()->getJButtonStart().setEnabled(true));
    IModelFactory factory = (d)-> new BuldModel(d, this);
    BuldModel model =(BuldModel) factory.createModel(dispatcher);
    model.initForTest();
    dispatcher.start();
}
```

У методі діаграми готуються до виводу графіків, створюються диспетчер, фабрика моделей та модель. Далі модель готується до роботи у режимі тестування, після чого стартує диспетчер. Кнопка «Старт» блокується на період моделювання, а розблоковує її об’єкт, що створено за допомогою лямбда функції, який реагує на завершення роботи диспетчера.

2.1.3 Режим накопичення та відображення статистичних даних

Режим “Stat” використовується для збирання та виведення на екран статистичних даних про роботу моделі.

Для реалізції цього режиму в моделі мають використовуватися об’єкти типу IHisto, посилання на які слід передати чергам та акторам.

Для відображення статистичних даних, рисунок 2.3, на закладці встановлено компонент типу StatistcsManager. Цьому компоненту після створення необхідно передати посилання на фабрику моделей. Для цього використовується така інструкція:

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

Тут використання лямбда функції дозволяє не створювати клас для фабрики моделей з методом createModel(Dispatcher d), який буде викликатися компонентом statisticsManager, з передачею у цей метод свого диспетчера;

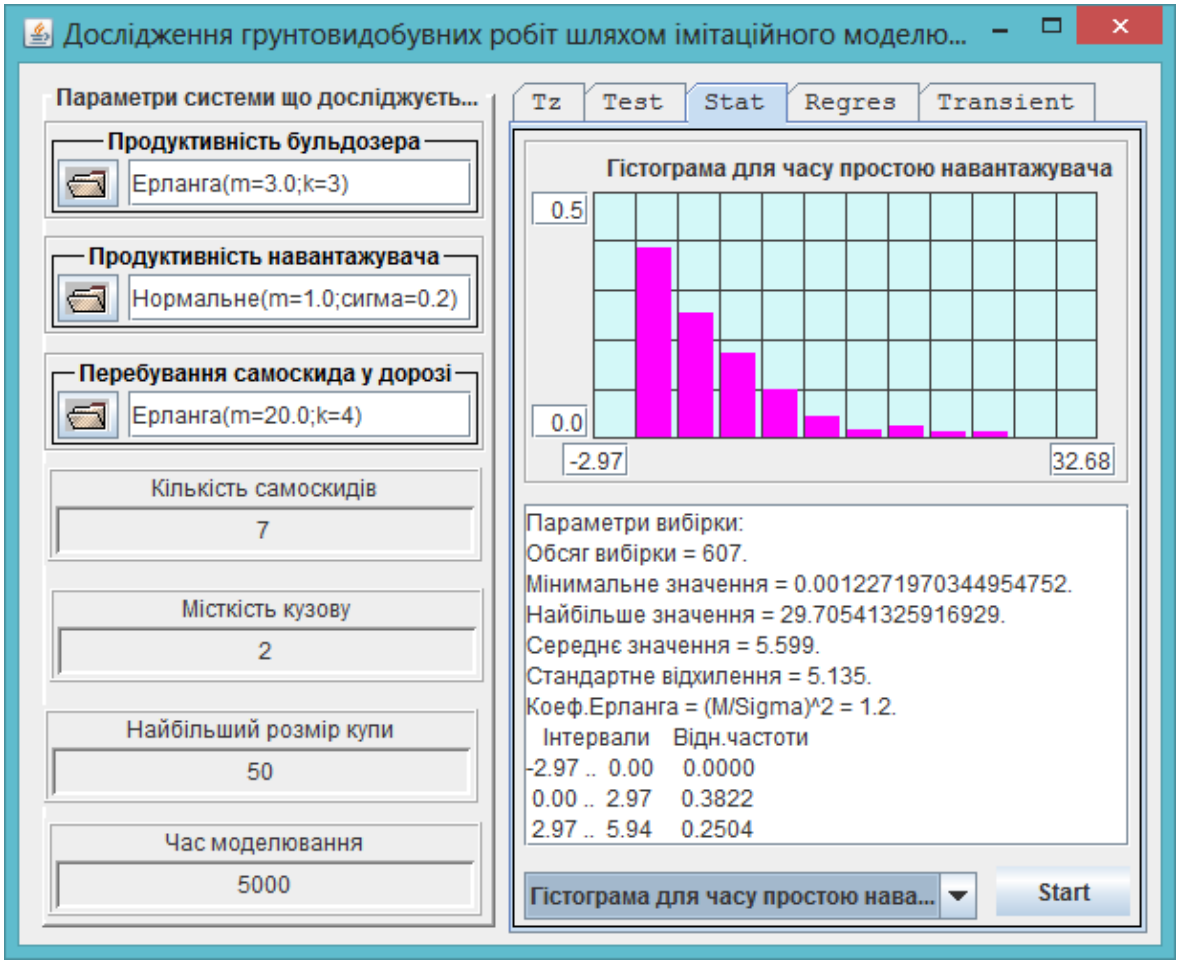

Рисунок 2.3 – Інтерфейс користувача моделі у режимі відображення статистичних даних щодо роботу моделі

Компонет ExperimentManager вимагає також, щоб модель реалізовала інтерфейс IStatisticsable.

2.1.4 Режим проведення однофакторних багаторівневих експериментів

У режимі “Regres”, рисунок 2.4, з моделлю можна проводити одноріневі та багаторівневі однофакторні експерименти по вивченню впливу заданого фактору на показники роботи моделі. Реалізація цього завдання покладена на компонет типу ExperimentManager.

Компоненту ExperimentManager після створення необхідно передати посилання на фабрику моделей. Для цього використовується така інструкція:

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

Для співпраці з цим компонентом модель має реалізувати інтерфейс IExperimentable.

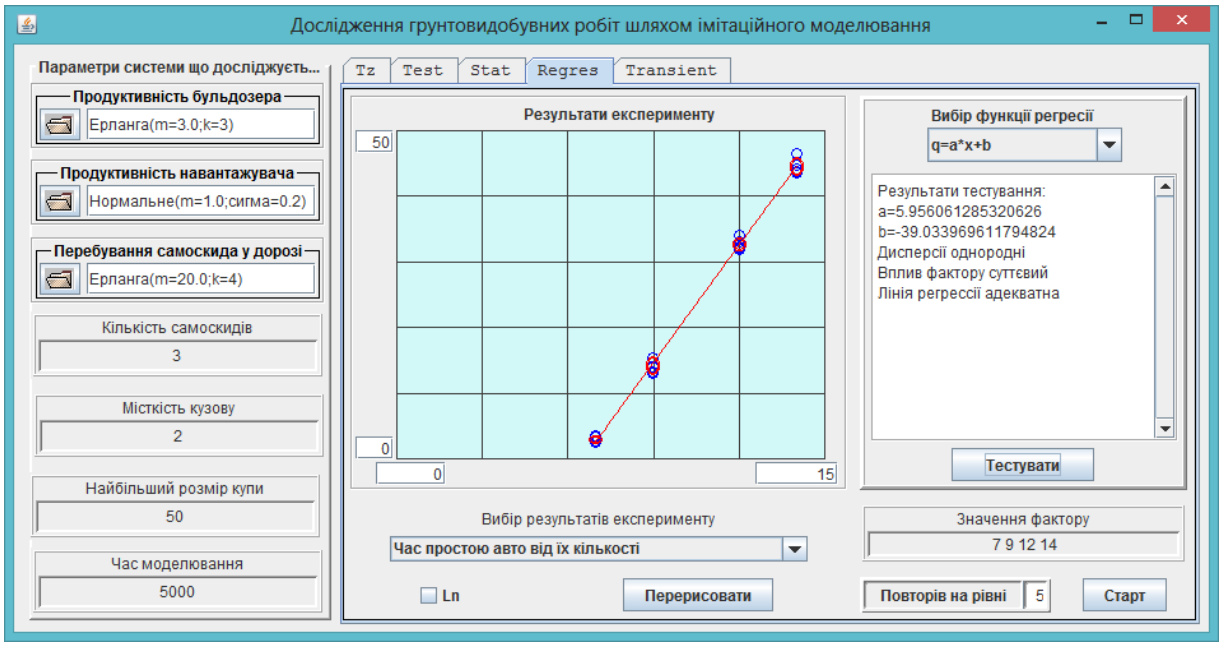

Рисунок 2.4 – Інтерфейс користувача моделі у режимі проведення багаторівневих однофакторних експериментів

2.1.5 Режим дослідження перехідних процесів

Режим “Transient” використовується для дослідження перехідних процесів у чегах моделі. Реалізація цього завдання покладена на компонет типу TransProcessManager, рисунок 2.5.

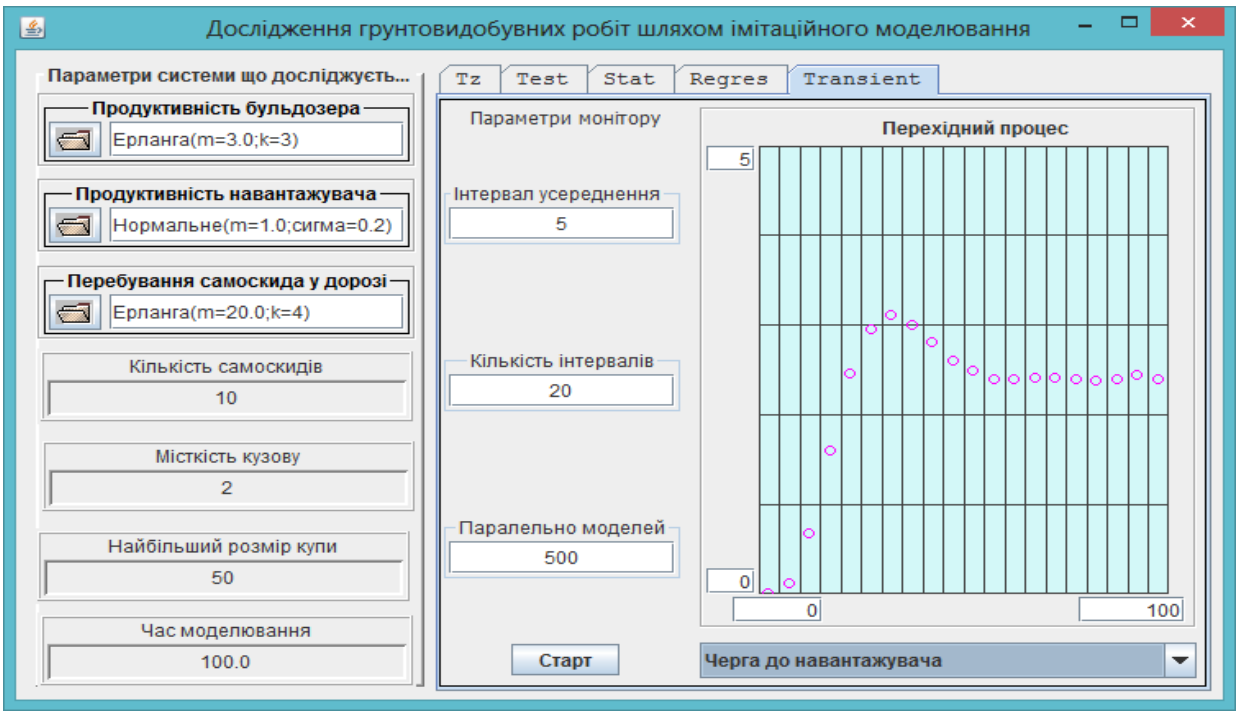

Рисунок 2.5 – Інтерфейс користувача моделі у режимі дослідження перехідних процесів

Компоненту TransProcessManager після створення необхідно передати посилання на фабрику моделей. Для цього використовується така інструкція:

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

Для співпраці з компонентом модель має реалізувати інтерфейс ITransPrcesable.

2.1.6 Публічний програмний інтерфейс шару подання

Ще одна важлива функція шару подання – надання доступу до своїх компонент іншим класам. Перелік відповідних публічних методів наведено на рисунку 2.6.

Рисунок 2.6 – Публічний інтерфейс шару подання

2.2 Реалізація шару моделі

Цей шар складається тільки з класу моделі.

2.2.1 Клас BuldModel

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

Перелік атрибутів моделі наведено у лістингу 2.3.

Лістинг 2.3 - Перелік атрибутів моделі

```
//Посилання на диспетчера
private Dispatcher dispatcher;
//Посилання на візуальну частину
private GUI gui;
//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
private DiscretHisto histoForQueueToLoader;
// Гістограма для часу простою бульдозера
private Histo histoBuldo;
// Гістограма для часу простою навантажувача
private Histo histoLoader;
// Гістограма для часу простою самоскида
private Histo histoLorry;
// Гістограма для розмірів купи
private Histo histoHeap;
```

Для створення моделі використовується конструктор з двома параметрами, який забезпечує гарантовану передачу моделі посилань на візуальну частину і диспетчера. Решта компонентів створюється безпосередньо у моделі. Конструктор наведено у лістингу 2.4.

Конструктор забезпечує також передачу акторів моделі до стартового списку диспетчера за допомогою методу componentsToStartList().

Лістинг 2.4 - Конструктор моделі

```
public BuldModel(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();
}
```

Такий конструктор можна вважати стандартним для виконання РГР або курсового проекту. Натомість, зміст методу componentsToStartList() залежіть від конкретного завдання. У нашому випадку цей метод має вигляд, представлений у лістингу 2.5.

Лістинг 2.5 - 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.

Як приклад методу для створення актора наведемо метод getBuldo(), лістинг 2.6. Для створення усіх акторів ми будемо використовувати однаковий підхід, що полягає у використанні конструктора з параметрами, які передають посилання на візуальну частину та на модель. Маючи ці посилання, створений об’єкт отримує доступ до інформації, що потрібна йому для функціонування.

Лістинг 2.6 - Метод відкладеного створення об’єкту бульдозер

```
public Buldo getBuldo() {
    if (buldo == null) {
        buldo = new Buldo("Бульдозер", gui, this);
    }
    return buldo;
}
```

Об’єкти класу MultiActor створюються дещо інакше. Такому об’єкту потрібно передати посилання на зразок, що буде клонуватися, та задати кількість клонів. Наводимо тут метод створення бригади авто, лістинг 2.7.

Лістинг 2.7 – Метод відкладеного створення бригади самоскидів

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

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 2.8.

Лістинг 2.8 – 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;
}
```

Об’єкти для гістограм, зазвичай, не потребують додаткових налаштувань. Як приклад, у лістингу 2.9 наведено метод доступу до гістограми для розмірів купи.

Лістинг 2.9 – Метод відкладеного створення гістограми

```
private Histo getHistoHeap() {
    if (histoHeap == null)
        histoHeap = new Histo();
    return histoHeap;
}
```

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

Так у методі initForTest() діаграмам передаються посилання на об’єкти класу Painter, що забезпечує відображення цих черг на гістограмах.

У інших режимах виникає потреба реалізації методів відповідних інтерфейсів. Якщо методи ініціалізації з цих інтерфейсів передають якісь налаштування компонентам моделі, то ці налаштування доцільно перенести до шару подання звідки компоненти їх можуть отримати.

2.2.1.1 Реалізація інтерфейсу IStatisticsable

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

Метод getStatistics(), лістинг 2.10, повертає асоціативний масив, який містить гістограми з накопиченої статистикою. На підставі цих даних компонент StatistcsManager формує результати.

Лістинг 2.10 – 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 для часу простою навантажувача", getHistoLoader());
    return map;
}
```

2.2.1.2 Реалізація інтерфейсу IExperimentable

Метод initForExperiment(double) цього інтерфейсу, лістинг 2.11, забезпечує передачу моделі значення фактору, вплив якого вивчається.

Лістинг 2.11 – Метод для передачі моделі значення фактору

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

Через метод getResultOfExperiment() цього інтерфейсу, лістинг 2.12, модель повертає компоненту результати кожного експерименту.

Лістинг 2.12 – 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;
}
```

2.2.1.3 Реалізація інтерфейсу ITransProcesable

Метод initForTrans(double) цього інтерфейсу, лістинг 2.13, 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.

Лістинг 2.13 – Реалізація методу підготовки до дослідження перехідних процесів

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

Через метод getTransResult() цього інтерфейсу, лістинг 2.15, компонент отримує середнє значення черги для кожного інтервалу накопичення даних.

Лістинг 2.14 – Реалізація методу повернення результатів накопичення

```
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;
}
```

2.2.1.4 Перелік публічних методів моделі

Перелік публічних методів моделі, що утворюють її інтерфейс наведено на рисунку 2.7.

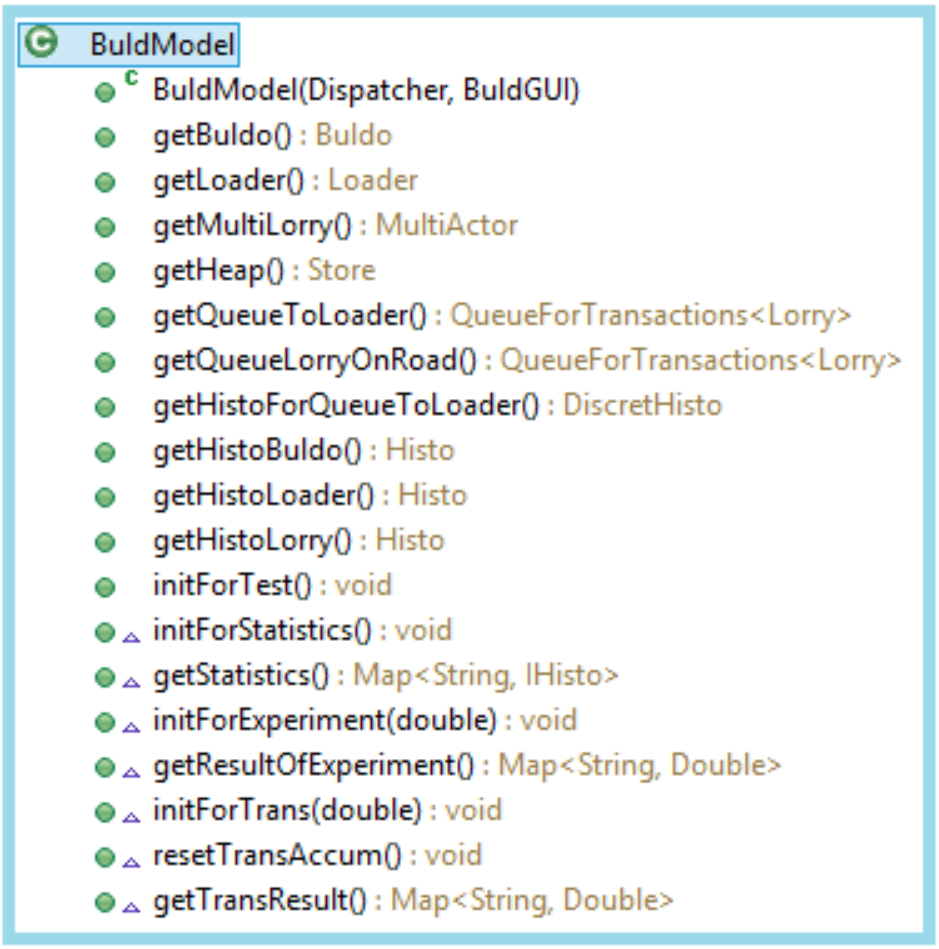

Рисунок 2.7 – Публічний інтерфейс моделі

2.3 Реалізація компонентів моделі

Класи, що потребують реалізації у даному проекті – це класи для бульдозера, навантажувача та клас для самоскида. Для створення об'єктів цих класів використовуються однотипні конструктори, через які передаються посилання на візуальну частину та модель. На підставі цих посилань у конструкторі ініціалізуються усі необхідні поля. Таким чином, атрибути акторів приймають свої значення у момент створення актора. У тих випадках, коли налаштування актора потрібно змінити після їх створення, слід для цього створити відповідні методи.

Поля, що містять умови у вигляді лямбда функцій у конструкторі визначати недоцільно. Справа у тому, що ці функції можуть містити посилання на створюваний об’єкт, якого в цей час ще не існує. Тому лямбда функції для умов, якщо вони є, створюються в момент запуску правил дії акторів, через метод initConditions()

2.3.1 Клас Buldo

Об’єкти цього класу представляють абстракцію бульдозер. У лістингу 5.14 наведено текст класу з коментарями.

Лістинг 5.14 – Клас Buldo

```
public class Buldo extends Actor {
   // Посилання на купу грунту, до якої бульдозер нагрібає 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;
   // Умова відновлення роботи бульдозера після зупинки.
   private BooleanSupplier heapHalfSize;

   // Конструктор, у якому ініціалізуються поля об'єкту
   // через посилання на модель  та візуальну частину
   public Buldo(String name, BuldGUI gui, BuldModel model) {
      setNameForProtocol(name);
      heap = model.getHeap();
      heapMaxSize = gui.getChooseDataHeapMaxSize().getDouble();
      finishTime = gui.getChooseDataFinishTime().getDouble();
      rnd = gui.getRndBuldo();
      setHistoForActorWaitingTime(model.getHistoBuldo());
   }
   // Правила дії бульдозера.
   protected void rule() throws DispatdherFinishException{
      // Ініціалізація умови
      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");
         }
      }
      public void setFinishTime(double finishTime) {
         this.finishTime =finishTime;
      }
}
```

2.3.1 Клас Loader

Об’єкти цього класу представляють абстракцію навантажувач. У лістингу 5.15 наведено текст класу з коментарями.

Лістинг 5.15 - Клас Loader

```
//Клас для абстракції Навантажувач
public class Loader extends BaseBuldActor {
   // 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;
   // Умова наявності самоскида у черзі
   private BooleanSupplier isLorry;
   // Умова наявності грунту у купіі
   private BooleanSupplier heapSize;
   // Конструктор
```

```
    public Loader(String name, BuldGUI gui, BuldModel model) {
    setNameForProtocol(name);
    heap = model.getHeap();
    queueToLoader = model.getQueueToLoader();
    finishTime = gui.getChooseDataFinishTime().getDouble();
    rnd = gui.getRndLoader();
    setHistoForActorWaitingTime(model.getHistoLoader());
}

// Правила дії навантажувача.
protected void rule() throws DispatdherFinishException {
    isLorry = () -> queueToLoader.size() > 0;
    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());
        }
```

```
        }
    }
    public void setFinishTime(double finishTime) {
        this.finishTime =finishTime;
    }
}
```

2.3.2 Клас Lorry

Об’єкти цього класу представляють абстракцію самоскид. У лістингу 5.15 наведено текст класу з коментарями.

Лістинг 5. 16 - Клас Lorry

```
public class Lorry extends BaseBuldActor {
    // Тривалість роботи самосвалу
    private double finishTime;
    // Черга самосвалів у дорозі
    private QueueForTransactions<Lorry> queueLorryOnRoad;
    // Черга самосвалів 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;
    //Умова завантаженості самоскида
    private BooleanSupplier isBodyFull;

    //Конструктор
    public Lorry(String name, BuldGUI gui, BuldModel model) {
        setNameForProtocol(name);
        this.queueToLoader = model.getQueueToLoader();
        this.queueLorryOnRoad = model.getQueueLorryOnRoad();
        this.finishTime = gui.getChooseDataFinishTime().getDouble();
        this.bodySize = gui.getChooseDataBodySize().getInt();
```

```
        this.rnd = gui.getRndLorry();
        this.setHistoForActorWaitingTime(model.getHistoLorry());
    }

    // Правила дії абстракції "Самоскид".
    public void rule() throws DispatdherFinishException {
        this.isBodyFull = () -> isFull();
        // Цикл правил дії самосвалу
        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
            // реєструється у списку самоскидів, що їдуть
            queueLorryOnRoad.add(this);
            getDispatcher().printToProtocol(
                    getNameForProtocol() + " поїхав розвантажуватися");
            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 самоскиду
    public void addPortion() {
        bodyLoad++;
        getDispatcher().printToProtocol(
                getNameForProtocol() + "- у кузові стало " + bodyLoad);
    }

    // Метод перевірки завантаженості самоскиду
    public boolean isFull() {
        return bodyLoad >= bodySize;
    }
    public void setFinishTime(double finishTime) {
        this.finishTime =finishTime;
    }
}
```

3 РЕЗУЛЬТАТИ ТЕСТУВАННЯ ПРОГРАМИ

Тестування програми проводилося з метою з'ясування її працездатності, адекватної реакції на зміну налаштувань та підтвердження можливості її використання для дослідження реальної системи.

3.1 Режим «Тест моделі»

У процесі тестування вивчалися реакції системи на зміну усіх налаштувань моделі.

На всі зміни налаштувань модель реагувала адекватно. У разі зміни налаштувань критичного розміру купи, кількості самоскидів та часу моделювання, відповідно до нових значень цих параметрів змінювалися і налаштування діаграм.

Шляхом експериментів з моделлю було підібрано налаштування моделі, що забезпечували такий режим роботи системи, при якому черги самоскидів до навантажувача були невеликими і розмір купи не збільшувався до критичного.

Результати роботи моделі у цьому режимі показано на рисунку 3.1.

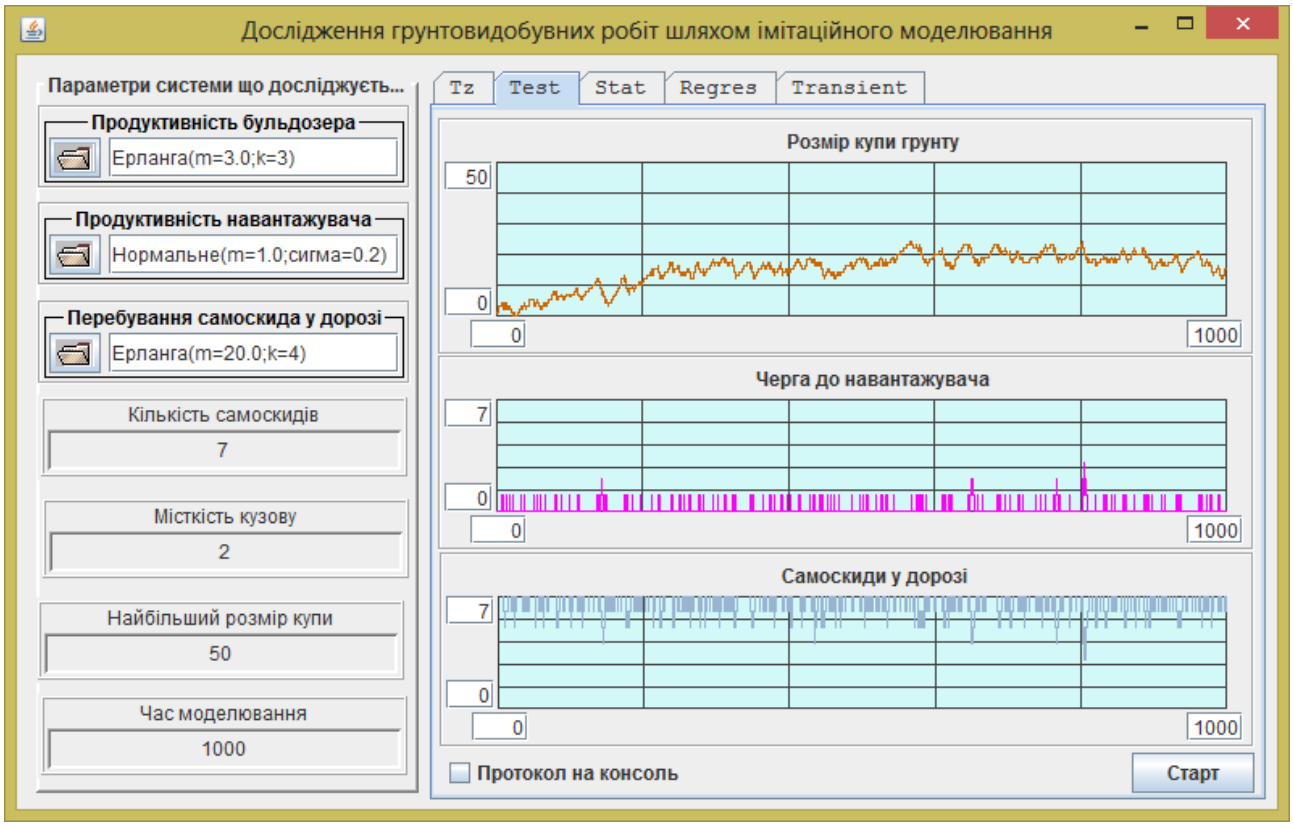

Рисунок 3.1 – Результати роботи моделі у нормальному режимі

На рисунку 3.2 показано результати моделювання системи при недостатній кількості самоскидів. У цьому режимі купа швидко досягає критичного розміру і бульдозер простоює.

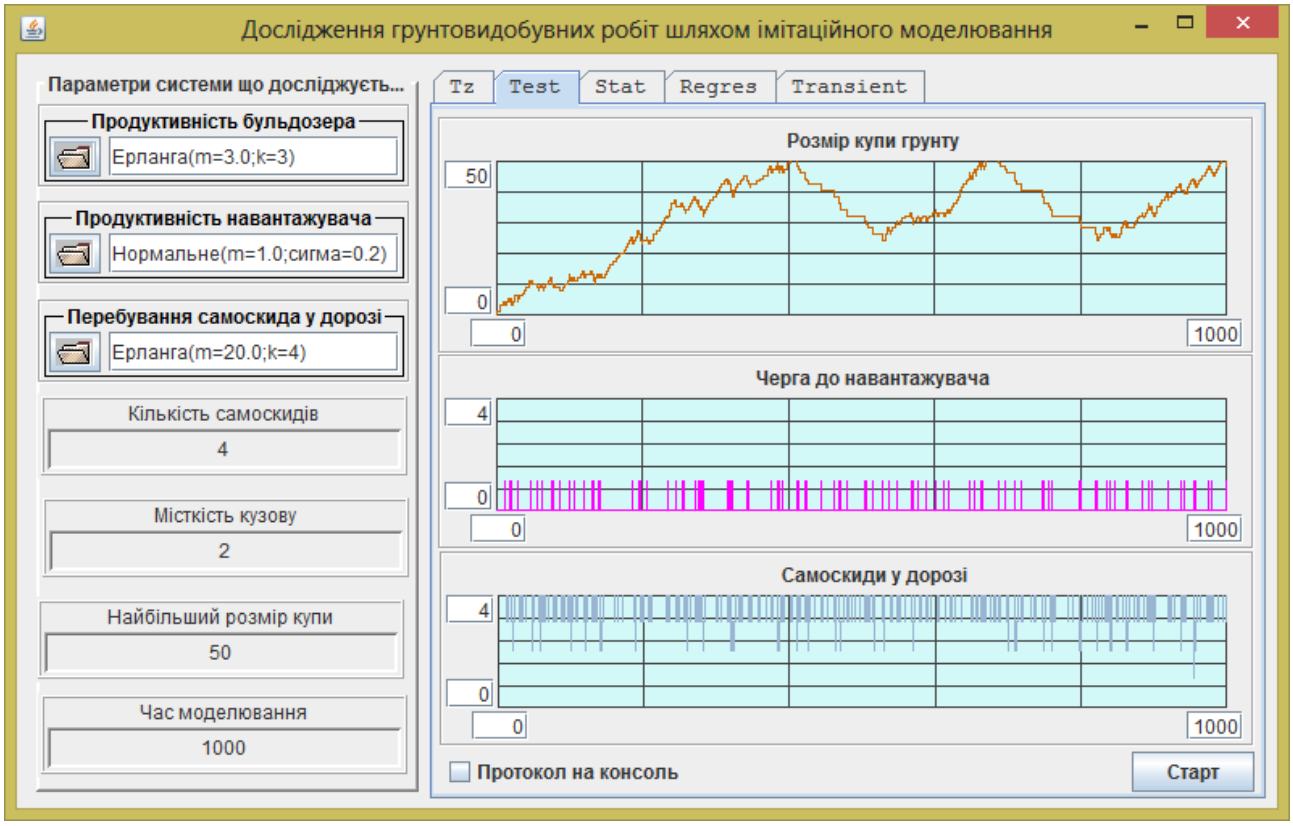

Рисунок 3.2 – Результати роботи моделі при недостатній кількості самоскидів

3.2 Режим «Статистика»

У процесі тестування проводився аналіз працездатності застосування у режимі статистики та оцінювалося значення часу моделювання, необхідного для одержання прийнятних за точністю статистичних даних.

Тестування працездатності тестування довело його придатність для отримання статистичних даних про довжину черги.

На рисунках 3.3 – 3.5 наведені результати отримання статистичних характеристик для довжини черги для часу моделювання, що дорівнював 100.

Як видно, вигляд гістограм, які представляють емпіричний закон розподілу для цієї випадкової величини у кожному з експериментів мав різний вигляд. Суттєво відрізнялися і значення середньої довжини черги, від 0.46 до 1.25. Але результати при цьому з'являлися миттєво.

На рисунках 3.6 – 3.8 наведені результати отримання статистичних характеристик для довжини черги для часу моделювання, що дорівнював 100000. У цьому випадку вигляд гістограм майже однаковий, а значення середньої довжини черги змінюються від 0.069 до 0.082. Але на експеримент вже витрачалося 8 секунд. Збільшення часу моделювання ще у 10 разів пропорційно збільшувало час на проведення експерименту.

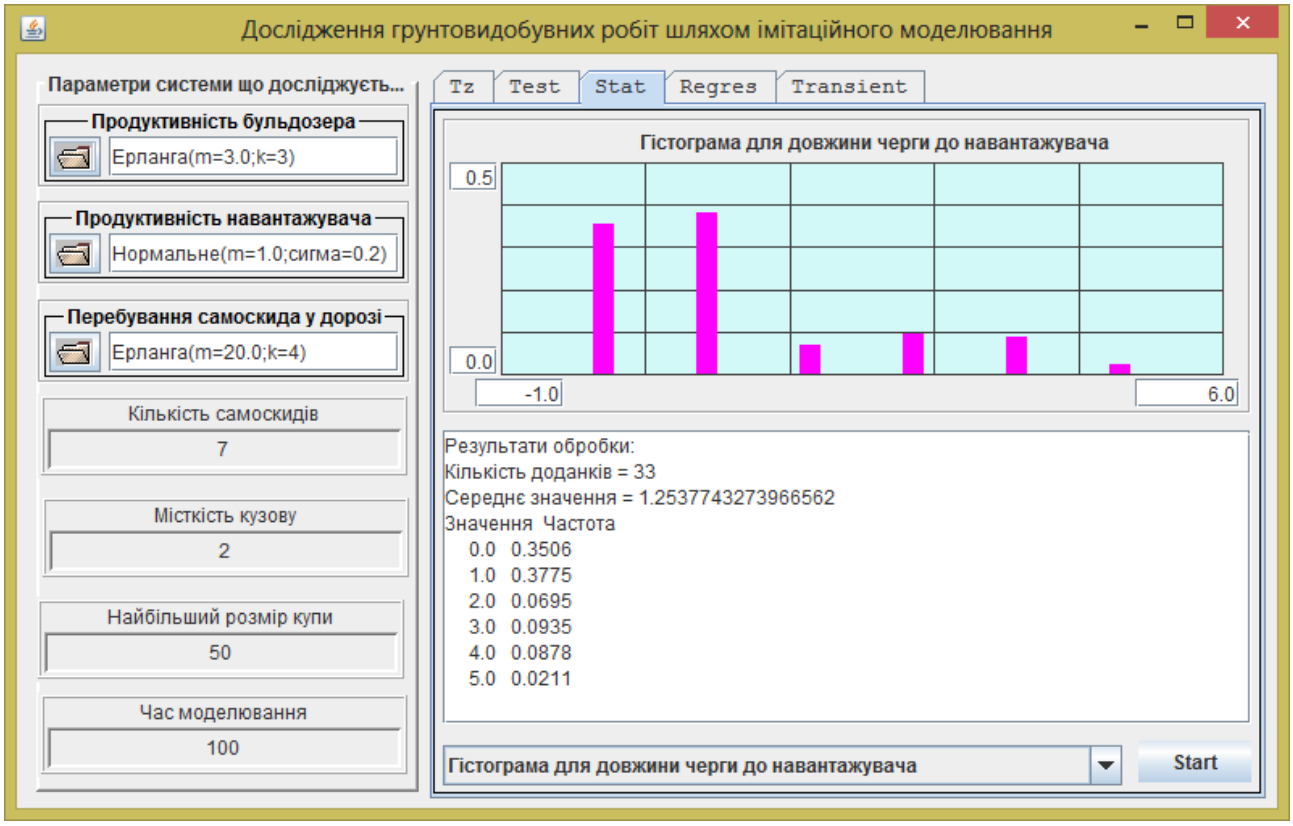

Рисунок 3.3 – Результати роботи моделі при недостатньому часі моделювання

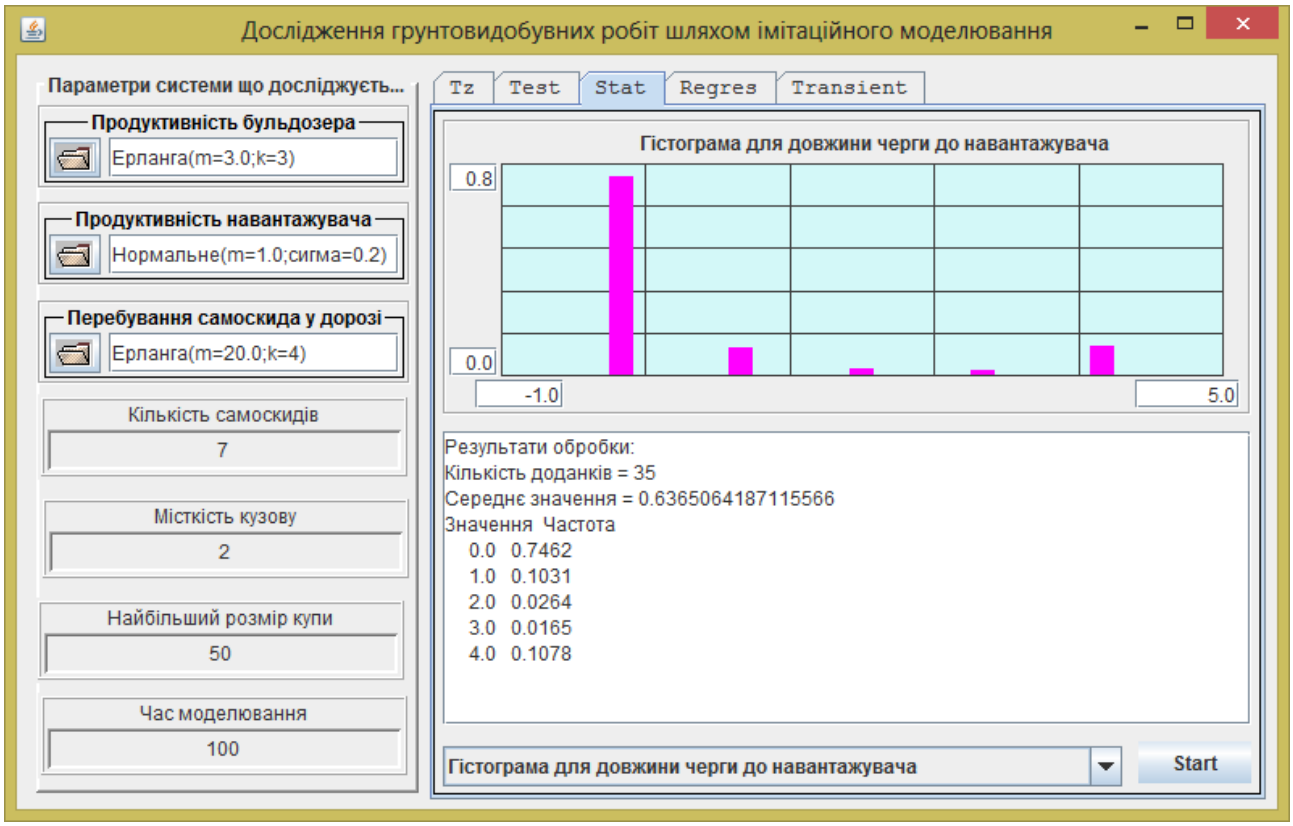

Рисунок 3.4 – Результати роботи моделі при недостатньому часі моделювання

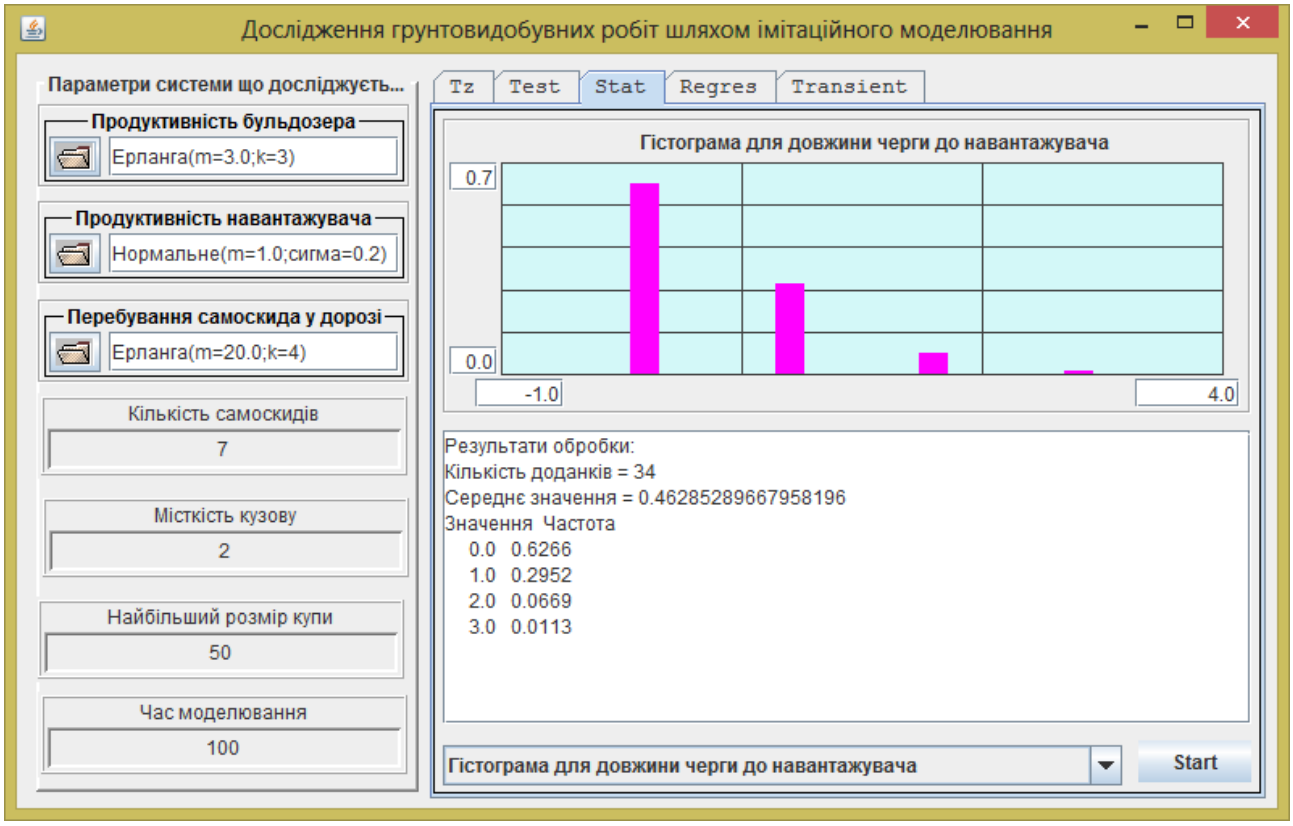

Рисунок 3.5 – Результати роботи моделі при недостатньому часі моделювання

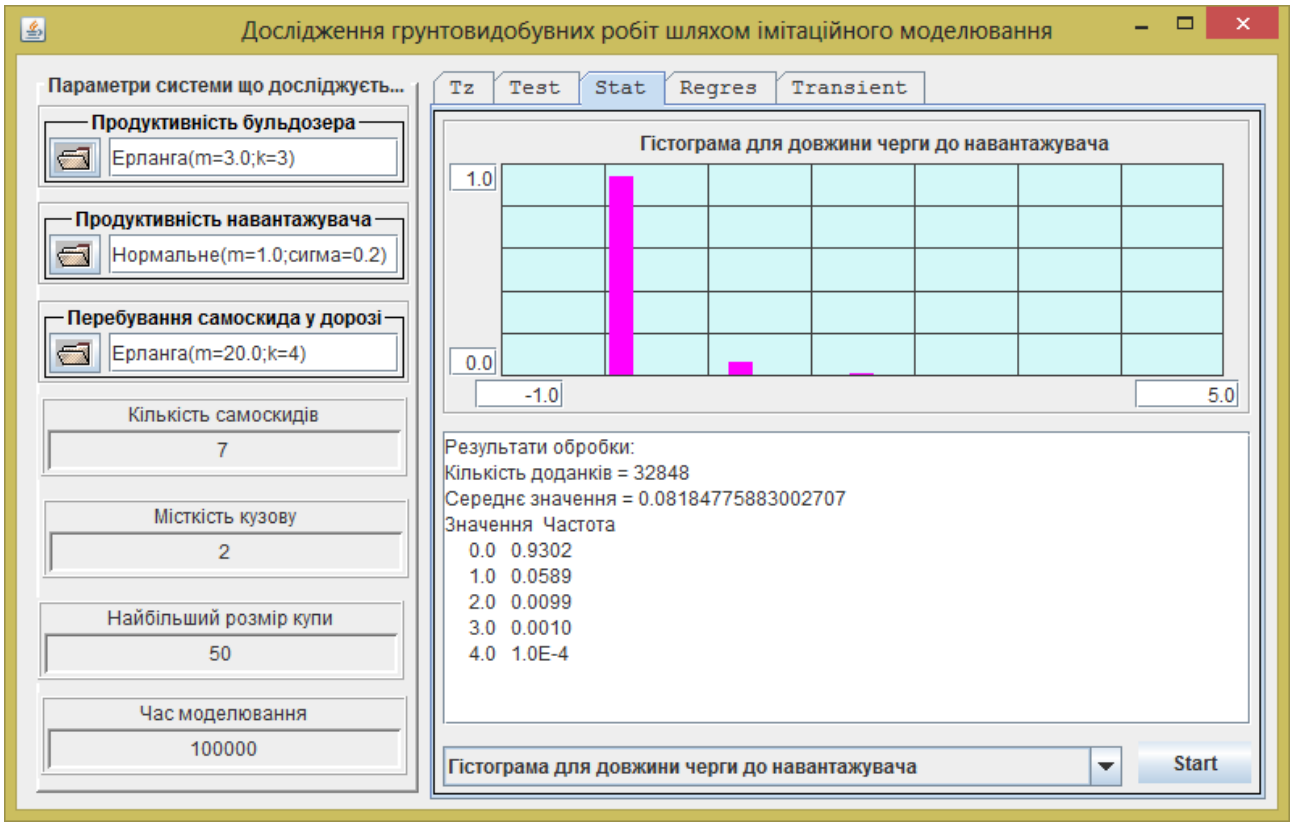

Рисунок 3.6 – Результати роботи моделі при достатньому часі моделювання

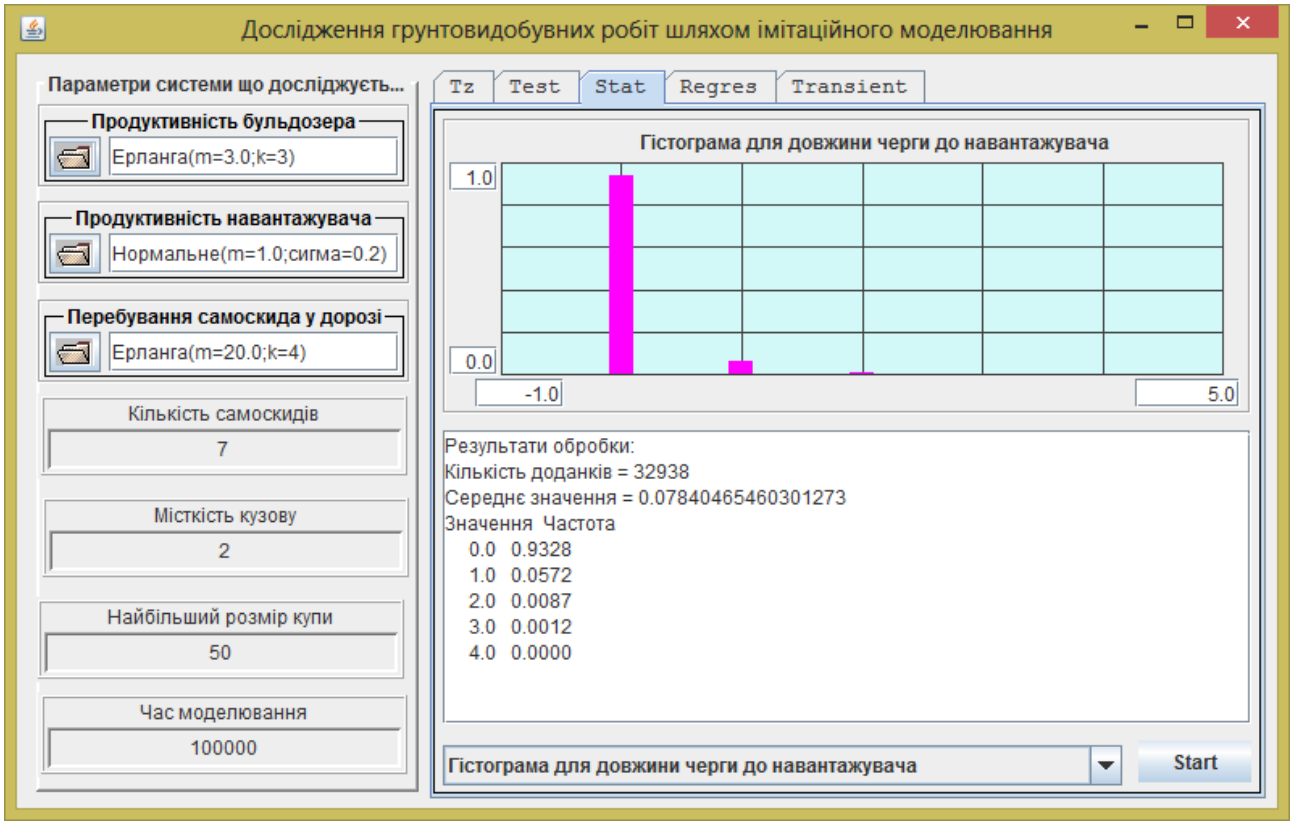

Рисунок 3.7 – Результати роботи моделі при достатньому часі моделювання

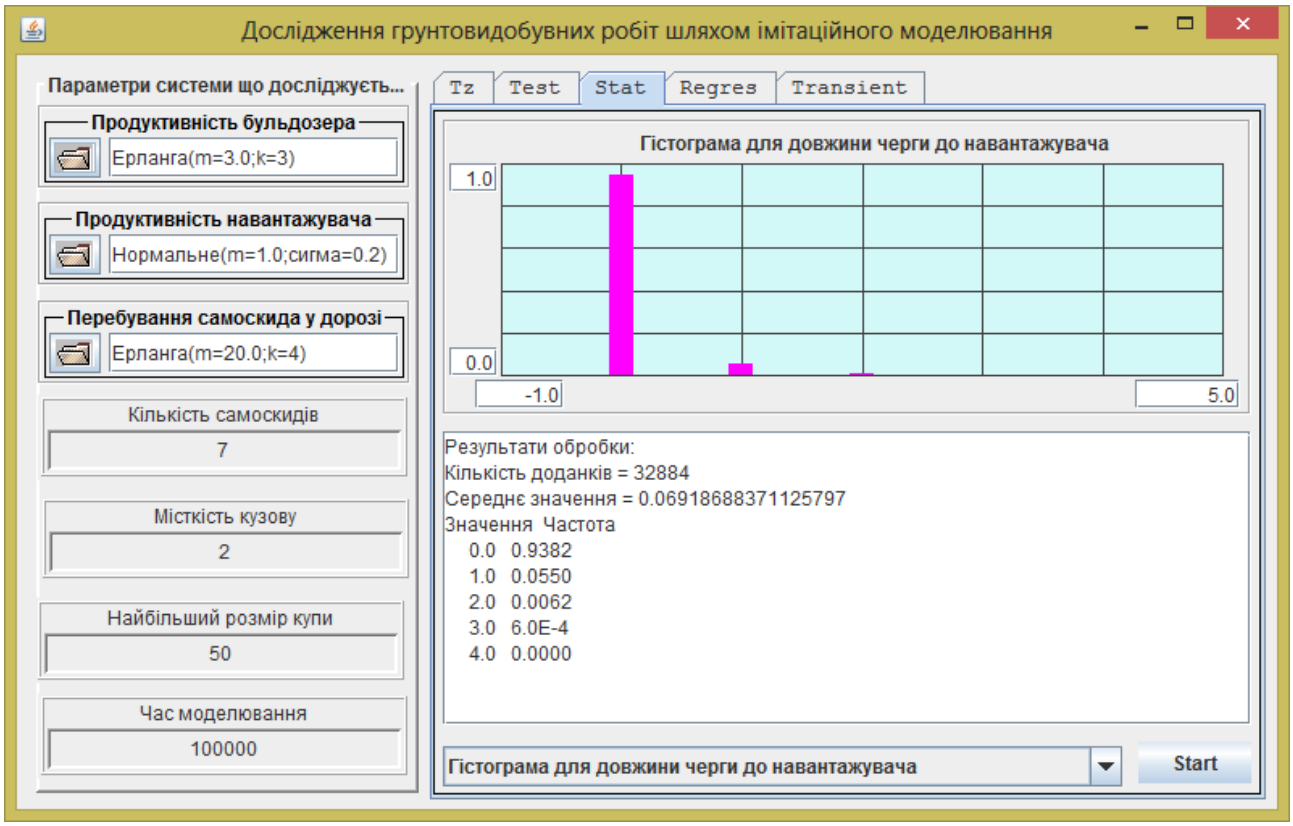

Рисунок 3.8 – Результати роботи моделі при достатньому часі моделювання

3.3 Режим «Однофакторні багаторівневі експерименти»

У процесі тестування проводився аналіз працездатності застосування у режимі «Однофакторні багаторівневі експерименти» та вивчався вплив кількості самоскидів на показники роботи моделі.

Тестування працездатності тестування довело його придатність для вивчення впливу фактору, що досліджувався.

На рисунках 3.9 – 3.11 наведені результати відповідних екпериментів.

Виявилося, що оптимальним числом для кількісті самоскидів є число 7.

Якщо кількість самоскідів менша, то зростає час простою бульдозера та навантажувача.

Якщо кількість самоскидів збільшувати то збільшується розмір черги сасоскидів і відповідно збільшується час їх простою.

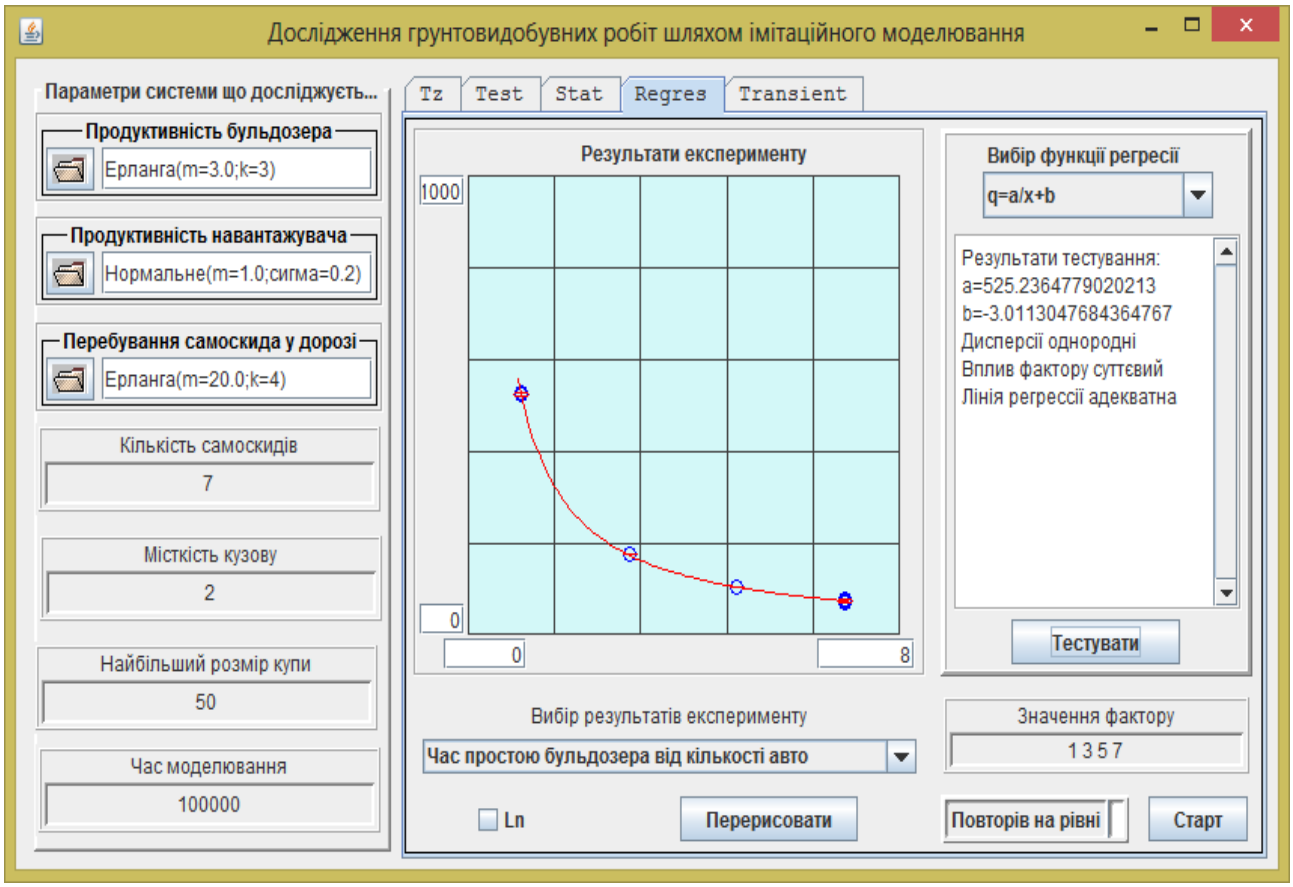

Рисунок 3.9 – Результати дослідження впливу кількості самоскидів на час простою бульдозера

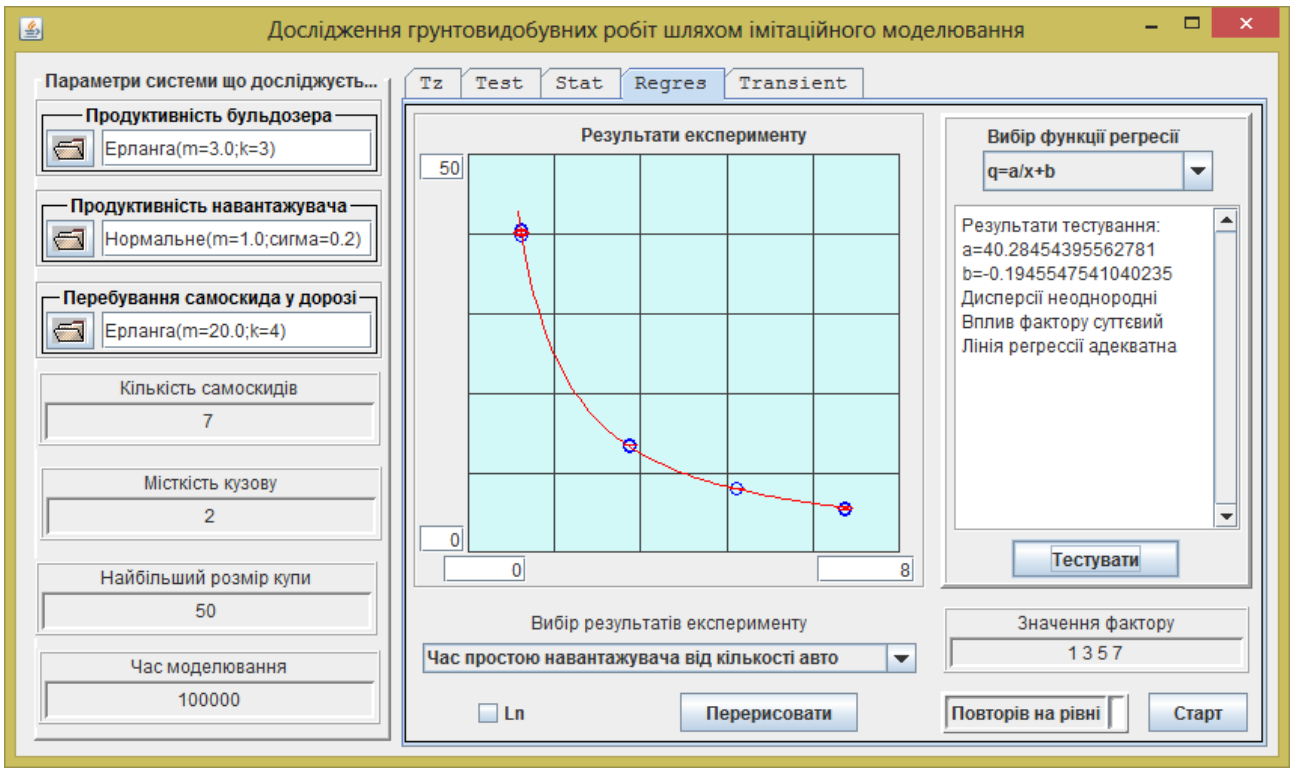

Рисунок 3.10 – Результати дослідження впливу кількості самоскидів на час простою навантажувача

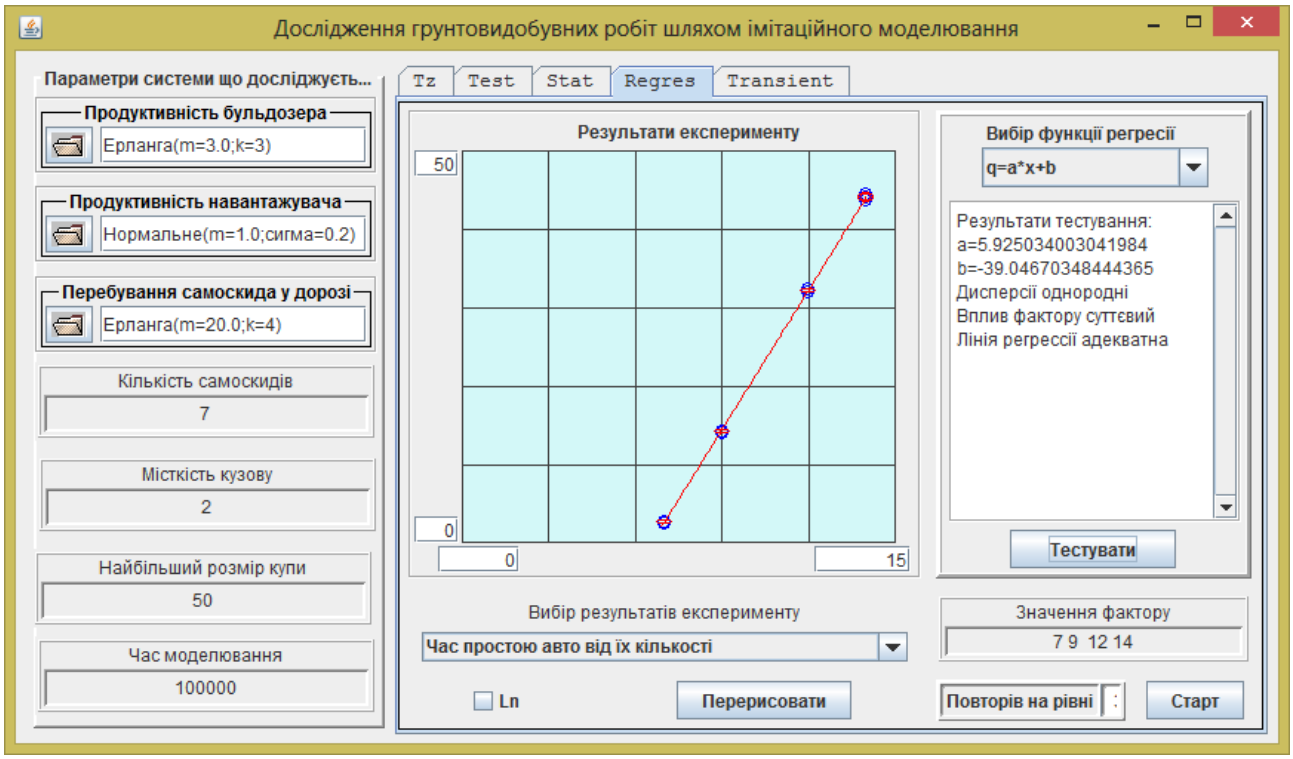

Рисунок 3.11 – Результати дослідження впливу кількості самоскидів на час їх простою у черзі

3.4 Режим «Дослідження перехідних процесів»

На рисунках 3.12 – 3.15 наведені результати відповідних екпериментів.

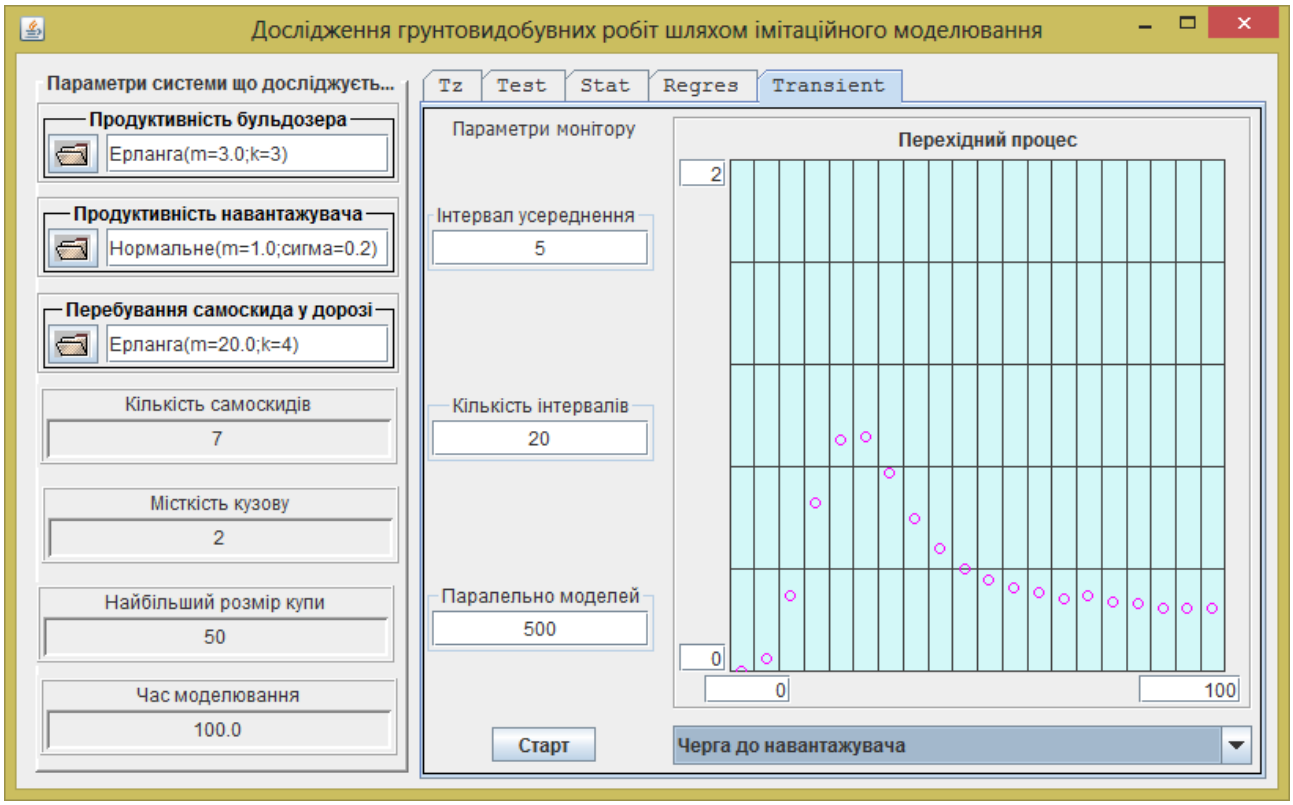

Рисунок 3.12 –Перехідний процес для черги до навантажувача

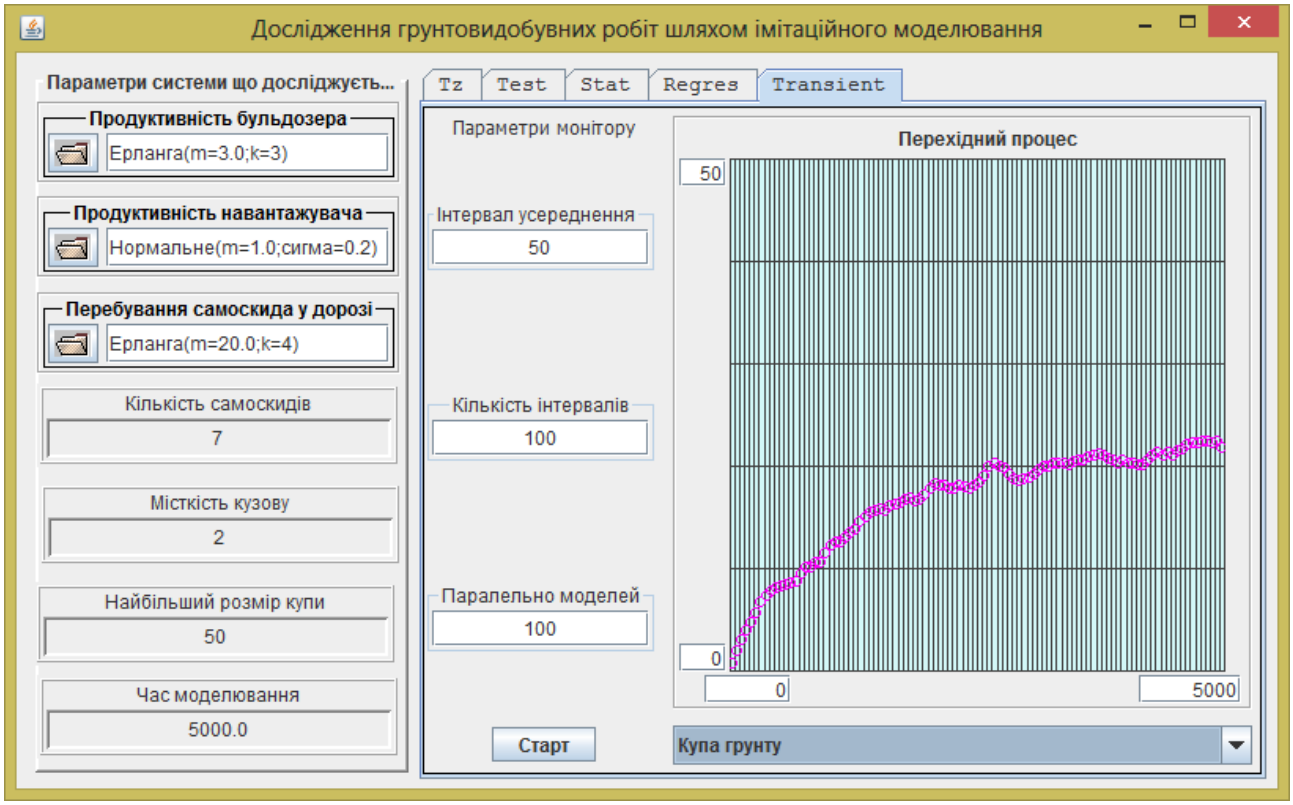

Рисунок 3.12 –Перехідний процес для купи грунту

У процесі тестування проводився аналіз працездатності застосування у режимі «Дослідження перехідних процесів» та визначалися показники перехідних процесів при оптимальній кількості самоскидів.

Тестування працездатності тестування довело придатність застосування для вивчення перехідних процесів у чергах системи.

Виявилося, що перехідний процес для черги самоскидів триває близько 100 одиниць часу.

Натомість перехідний процес для купи грунту триває досить довго, принаймі 5000 одиниць часу.

ВИСНОВКИ

У процесі виконання розрахунково-графічної роботи було проведено об'єктно-орієнтований аналіз предметної області та визначені абстракції досліджуваної системи.

Аналіз можливостей фреймворку SimulationJava показав, що його засоби суттєво зменшують обсяг роботи по створенню системи моделювання. У процесі реалізації програми класи для активних компонент моделі створювалися шляхом успадкування класу Actor. Для створення черг, накопичувачів та гістограми використовувалися класи фреймворку. Для налаштувань моделі та відображення результатів моделювання у візуальній частини програми також використовувалися компоненти фреймврку.

З точки зору архітектури програма розроблялася як сукупність трьох шарів. Це дозволило розподілити роботу над програмою між трьома членами команди і прискорити її створення.

Тестування програми довело її працездатність і придатність для отримання статистичних характеристик досліджуваної системи.

Рекомендована література

1. Структура і оформлення кваліфікаційних та курсових робіт. Методичні вказівки для студентів професійного спрямування «Комп'ютерна інженерія» / /Укл.: А.І.Вервейко, С.О., Нестеренко, Є.В.Нікітенко – Чернігів: ЧДТУ, 2001. – 28с.
2. Томашевський В.М. Моделювання систем. – К.: Видавнича група BHV, 2005. -352 с.:іл.
3. Лоу А.М., Кельтон В.Д. Имитационное моделирование. - 3-е изд. - СПб.: Питер, 2004. – 847с.
4. РГР з моделювання. Методичні вказівки до виконання розрахунково-графічних робіт з дисципліни «Моделювання» для студентів напряму підготовки 6.050102 – „Комп'ютерна інженерія". /Укл.: Бивойно П.Г., Пріла О.А, Бивойно Т.П. – Чернігів: ЧДТУ, 2012. – 64 с.
5. Імітаційне моделювання паралельних процесів. Методичні вказівки до лабораторного практикуму та самостійної роботи з дисципліни "Моделювання" для студентів напряму підготовки 6.050102 "Комп'ютерна інженерія". Укладачі Бивойно П.Г., Павловський В.І. - Чернігів, ЧДТУ, 2008. - 54 с.