Міністерство освіти і науки України

Чернігівський національний технологічний університет

Кафедра інформаційних та комп’ютерних систем

**МОДЕЛЮВАННЯ РОБОТИ РОЗВАНТАЖЕННЯ АВТОМОБІЛІВ**

Курсова робота з дисципліни “Програмна інженерія”

|  |  |  |
| --- | --- | --- |
| Виконавці  студенти гр. КС-111 |  | Легкодух С.Н.  Ярмоленко О.Ю. |
| Керівники:  доцент  асистент |  | Бивойно П.Г.  Пріла О.А. |

Чернігів ЧНТУ 2015

**ТЕХНІЧНЕ ЗАВДАННЯ**

на виконання курсової роботи з дисципліни «Програмна інженерія»

**Тема**: Моделювання розвантаження автомобілів, Варіант «TransAvto»

**Виконавці**: Легкодух С.М., Ярмоленко О.Ю., гр.КС-111.

**Опис системи**

Вантажівки прибувають на розвантажувальну станцію у випадкові моменти часу, причому інтенсивність прибуття залежить від часу доби. Час розвантаження

вантажівки випадковий. Розвантаженням вантажівки займається одна бригада вантажників. Розвантажувальна станція працює в три зміни по 8 годин. У кожну зміну на роботу виходить кілька бригад вантажників. У тому випадку, якщо бригада почала розвантажувати автомобіль, а робочий час закінчився, вантажники працюють надурочно.

**Завдання на проектування**

Створити Java застосування для імітаційного моделювання робіт

з обслуговування клієнтів, яке позволить:

– перегляд технічного завдання

– налаштування параметрів моделі, а саме: тривалість моделювання, кількість смуг, кількість місць на смузі, кількість місць на виїзді, закони розподілення для випадкових величин інтервалу прибуття клієнтів, часу обслуговування, часу виїзду, часу зайнятості дороги;

– проведення тестових запусків моделі при різних налаштуваннях з динамічною індикацією черги на обслуговування, черги на виїзд, втрачених клієнтів, та можливості виводу протоколу роботи моделі під час тестових запусків;

– проведення експериментів для отримання статичних характеристик для часу черги на розвантаження;

– реалізацію плану проведення багаторівневих експериментів з метою отримання залежності середнього розміру черги та середнього часу очікування клієнта у черзі від кількості смуг та дисперсійний і регресивний аналіз отриманих результатів;

– дослідження перехідного процесу для черги на розвантаження

**Обсяг текстової документації**

Робота обсягом 35-40 с. формату А4

**Орієнтовна трудомісткість роботи** – 36 людино-годин.

**Дата представлення роботи –** останній тиждень 2-го модуля

Керівник роботи

доцент Бивойно П.Г.

асистент Пріла О.А.

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2015 р.

РЕФЕРАТ

Курсовая работа, 52 с., рис. 25, 5 источников.

Цель разработки курсовой работы – реализовать приложение, которое моделирует работу разгрузки автомобилей.

Основным методом проектирования модулей приложения – использование UML – диаграмм.

В процессе написания приложения была разработана и создана фабрика Factory для работы с сущностями. С помощью Factory была дополнительно реализована бизнес-логика.

Приложение было реализовано с помощью языка программирования Java. В процессе разработки использованы средства разработки: инструмент сборки Ant, система контроля версий Git.

Приложение было разработано с использованием комплекта JDK версии 8.

В ходе разработки было получено приложение, моделирующее работу разгрузки автомобилей, доведенное до уровня стабильной версии. Результат разработки оформлен в виде программного проекта, приводимого в приложении к курсовой работе.

Дальнейшее развитие работы возможно в сторону усовершенствования и развития алгоритма работы.

Java, ANT, Git, JDK.

РЕФЕРАТ

Курсова робота, 52 с., Рис. 25, 5 джерел.

Мета розробки курсової роботи – реалізувати програму, яку моделює роботу розвантаження автомобілів.

Основним методом проектування модулів додатка – використання UML – діаграм.

У процесі написання додатка була розроблена і створена фабрика фабрика для роботи з сутностями. За допомогою Factory була додатково реалізована бізнес-логіка.

Додаток було реалізовано за допомогою мови програмування Java. У процесі розробки використані кошти розробки: інструмент збірки Ant, система контролю версій Git.

Додаток було розроблено з використанням комплекту JDK версії 8.

У ході розробки було отримано додаток, що моделює роботу розвантаження автомобілів, доведене до рівня стабільної версії. Результат розробки оформлений у вигляді програмного проекту, що приводиться в додатку до курсової роботи.

Подальший розвиток роботи можливе у бік удосконалення та розвитку алгоритму роботи.

JAVA, ANT, GIT, JDK.

THE ABSTRACT

Course project, 52 pp., pick. 25, 5 sources.

The purpose of developing course work – to implement an application that simulates the operation of unloading cars.

The basic method of designing application modules – use UML – diagrams.

In the process of writing applications have been designed and built factory Factory to work with entities. With Factory was further implemented business logic.

The application was implemented using the programming language Java. During the development process used development tools: instruient build Ant, a version control system Git.

The application was developed using a JDK version 8.

During the development of a desktop application has been received, modeling work brought to the unloading cars level stable version. Result development is designed as a software project, described in annex to the term paper.

Further development work is possible in the direction of improvement and development of the algorithm.

Java, ANT, Git, JDK

**ЗМІСТ**

[РЕФЕРАТ 4](#_Toc418153288)

[РЕФЕРАТ 5](#_Toc418153289)

[THE ABSTRACT 6](#_Toc418153290)

[Вступ 8](#_Toc418153291)

[1 АНАЛІЗ СИСТЕМИ, ЩО ПІДЛЯГАЄ МОДЕЛЮВАННЮ 9](#_Toc418153292)

[1.1 Опис системи 9](#_Toc418153293)

[1.2 Виділення основних абстракцій системи 9](#_Toc418153294)

[1.3 Аналіз активних абстракцій системи 11](#_Toc418153296)

[1.3.1 Бригада 11](#_Toc418153297)

[1.3.2 Авто 12](#_Toc418153298)

[1.3.3 Генератор часу 13](#_Toc418153299)

[1.4 Аналіз можливостей фреймворку Simulation для реалізації абстракцій системи 14](#_Toc418153300)

[2 РЕАЛІЗАЦІЯ СИСТЕМИ 15](#_Toc418153301)

[2.1 Реалізація шару подання 15](#_Toc418153302)

[2.1.1 Режим перегляду технічного завдання 15](#_Toc418153303)

[2.1.2 Режим тестування моделі 16](#_Toc418153304)

[2.1.3 Режим накопичення та відображення статистичних даних 17](#_Toc418153305)

[2.1.4 Режим проведення однофакторних багаторівневих експериментів 18](#_Toc418153306)

[2.1.5 Режим дослідження перехідних процесів 19](#_Toc418153307)

[2.1.6 Публічний програмний інтерфейс шару подання 20](#_Toc418153308)

[2.2 Реалізація шару моделі 20](#_Toc418153309)

[2.2.1 Клас Model 20](#_Toc418153310)

[2.3 Реалізація компонентів моделі 25](#_Toc418153311)

[2.3.1 Клас Brigada 25](#_Toc418153312)

[2.3.2 Клас Factory 26](#_Toc418153313)

[2.3.3 Клас GeneratorAuto 27](#_Toc418153314)

[3 РЕЗУЛЬТАТИ ВИКОНАННЯ ПРОЕКТУ 29](#_Toc418153315)

[3.1 Результати тестування програми 29](#_Toc418153316)

[3.1.1 Режим «Тест моделі» 29](#_Toc418153317)

[3.1.2 Режим «Статистика» 30](#_Toc418153318)

[3.1.3 Режим «Однофакторні багаторівневі експерименти» 32](#_Toc418153319)

[3.1.4 Режим «Дослідження перехідних процесів» 33](#_Toc418153320)

[3.2 Система контролю версіями Git 35](#_Toc418153321)

[3.3 Система збирання проектів 36](#_Toc418153322)

Вступ

Дана курсова робота переслідує такі цілі:

* зміцнення знання, які були отримані при вивченні дисципліни «Моделювання»;
* удосконалення навичок проведення об'єктного аналізу системи, об'єктно-орієнтованого проектування та програмування;
* отримання навичок командної розробки програмних проектів;

У рамках курсової роботи необхідно створити програму, яка дозволить дослідити дискретно – динамічну систему шляхом імітаційного моделювання та визначити характеристики цієї системи.

Для виконання роботи необхідно розглянути питання експериментального дослідження, проблеми планування експериментів, регресивного та дисперсійного аналізу статистичних даних, а так само методика дослідження перехідних процесів , необхідних для курсової роботи.

При виконанні курсової роботи передбачається – створити Java застосування для імітаційного моделювання роботи розвантаження автомобілів, яке дозволить:

* налаштування параметрів моделі;
* проведення тестових запусків моделі при різних налаштуваннях та
* можливість виведення протоколу роботи моделі під час тестових запусків;
* проведення експериментів для отримання статистичних характеристик.

При розробці курсового проекту будуть використовуватись:

* + система контролю версій Git;
  + система збирання проекту Ant;

# АНАЛІЗ СИСТЕМИ, ЩО ПІДЛЯГАЄ МОДЕЛЮВАННЮ

## Опис системи

Вантажівки прибувають на розвантажувальну станцію у випадкові моменти часу, причому інтенсивність прибуття залежить від часу доби. Час розвантаження

вантажівки випадковий. Розвантаженням вантажівки займається одна бригада вантажників. Розвантажувальна станція працює в три зміни по 8 годин. У кожну зміну на роботу виходить кілька бригад вантажників. У тому випадку, якщо бригада почала розвантажувати автомобіль, а робочий час закінчився, вантажники працюють надурочно.



Рисунок 1.1 – Система розвантаження автомобілів

## Виділення основних абстракцій системи

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

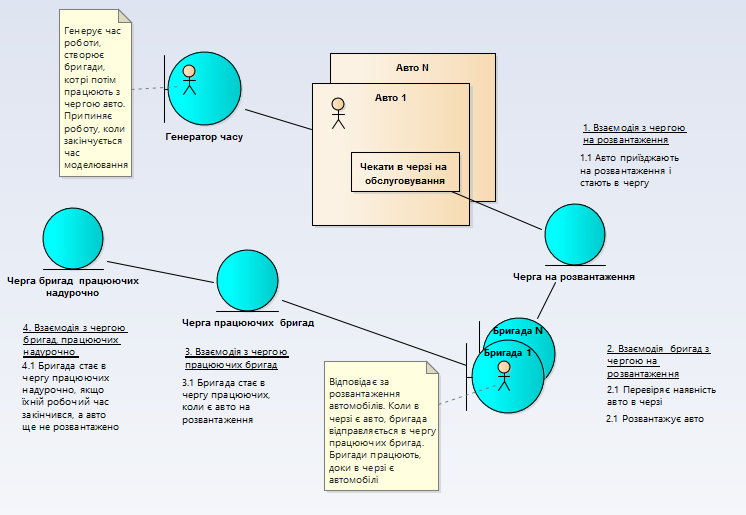


Рисунок 1.2 – Діаграма бізнес процесів у системі

Аналізуючи опис системи, наведений у технічному завданні і наведену діаграму, перш за все можна виділити таку абстракцію, як «Бригада». Характеристикою цієї абстракції є час розвантаження авто. Цей параметр є індикатором деяких процесів, що відбуваються у системі, бо його значення є результатом взаємодії інших елементів системи, а саме впливає на «чергу на розвантаження».

Абстракція «черга на розвантаження» моделює чергу авто, які чекають на обслуговування. Найбільш вірогідно, що ця величина підпорядковується нормальному закону розподілення.

Абстракція «Авто» моделює чергу автомобілів на обслуговування, які чекають у черзі на розвантаження. Абстракція «Авто» у системі призначена для отримання автомобілів, які приїжджають на обслуговування.

Перед роботою абстракція «Бригада» робить перевірку на наявність авто, при наявності авто, бригада виконує свою роботу, розвантажує автомобілі. Якщо авто немає, бригада чекає появи авто. При цьому робота бригади затримується на деякий час.

Абстракція «Черга працюючих бригад» моделює чергу бригад, які працюють з авто на розвантаження.

Абстракція «Черга бригад працюючих надурочно» моделює чергу бригад, які працюють з авто на розвантаження, але їхній робочий час закінчився.

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

|  |  |
| --- | --- |
| Таблиця 1.1 – Абстракції системи, що підлягає моделюванню | |
| Абстракція | Перелік завдань |
| Черга на розвантаження | Місце, де знаходяться автомобілі, що чекають на розвантаження. Забезпечити динамічну індикацію змін свого розміру у часі. |
| Бригада | Моделює обслуговування черги автомобілів. |
| Авто | Створює через випадкові інтервали часу авто, які стають у чергу на розвантаження. Відповідно автомобілі, потрапляють до черги на розвантаження. |
| Черга працюючих бригад | Місце, де працюють бригади, що отримали авто на розвантаження. Забезпечити динамічну індикацію змін свого розміру у часі. |
| Черга бригад працюючих надурочно | Місце, де працюють бригади, що отримали авто на розвантаження, але не розвантажили авто за свою зміну. Забезпечити динамічну індикацію змін свого розміру у часі. |

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

## Аналіз активних абстракцій системи

Активними абстракціями системи є бригада, авто, генератор часу. Розглянемо їх поведінку та представимо у вигляді діаграм діяльності.

### Бригада

Бригада пов'язана з чергою на розвантаження. Завданням бригади є розвантаження прибувших автомобілів.

Якщо автомобіль приїхав на розвантаження, він потрапляє до черги на розвантаження. Перед розвантаженням проводиться перевірка на наявність авто в черзі. Якщо воно є, то бригада розвантажує авто певний проміжок часу, який задається випадково.

Після обслуговування авто, цикл повторяється знову.

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

|  |  |  |  |
| --- | --- | --- | --- |
| Таблиця 1.3 – Атрибути абстракції бригада | | | |
| Назва поля | Клас | Призначення поля | Джерело |
| isAvtoOrFinishWork | BooleanSupplier | Перевірка на наявність авто в черзі на розвантаження або закінчення часу моделювання | Model |
| quequeToBrigada | QueueForTransactions | Черга на розвантаження | Model |

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

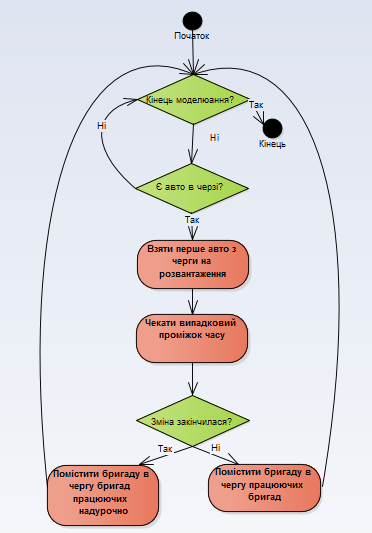


Рисунок 1.3 – Діаграма діяльності абстракції «Бригада»

### Авто

Після генерації автомобіля, він може потрапити до черги на розвантаження.

Після цього цикл повторюється знову. Працює авто впродовж усього часу моделювання.

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

|  |  |  |  |
| --- | --- | --- | --- |
| Таблиця 1.4 – Атрибути абстракції авто | | | |
| Назва поля | Клас | Призначення поля | Джерело |
| queque | QueueForTransactions | Черга до бригади на розвантаження | Model |
| rnd | ChooseRandom | Посилання на генератор випадкових чисел | Gui |
| isAvto | BooleanSupplier | Відображає стан автомобілів | Model |
| timer | Timer | Інтенсивність прибуття авто вдень і вночі | Model |

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

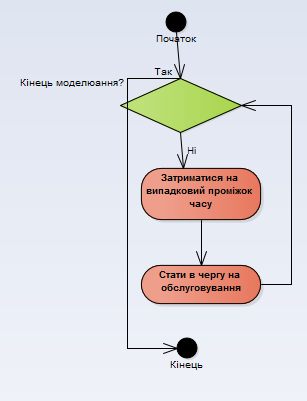


Рисунок 1.3 – Діаграма діяльності абстракції «авто»

### Генератор часу

Головне завдання цієї абстракції – керувати робочим часмом, створювати нові бригади. Цим самим імітується їхня робота по змінах. Правила дії генератора часу схематично можна представити у вигляді діаграми діяльності, рисунок 1.5.

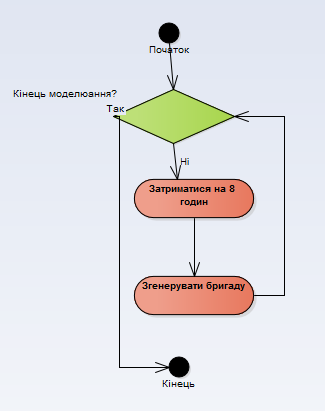


Рисунок 1.5 – Діаграма діяльності абстракції «генератор часу»

## 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 авто і бригада. Для моделювання таких абстракцій фреймворк Simulation містить абстрактний клас process.Actor, на основі якого можна створити класи, що реалізують особливості абстракцій авто і бригада. Для цього необхідно тільки визначити у класі-спадкоємці правила дії відповідної абстракції та її поля.

Абстракції «черга на розвантаження», «черга працюючих бригад» та «черга бригад працюючих надурочно», можна реалізувати як об’єкти класу queues.QueueForTransactions.

Результати цього етапу аналізу наведені у таблиці 1.5.

|  |  |
| --- | --- |
| Таблиця 1.5 – Результати аналізу можливостей фреймворку simulation для реалізації абстракцій системи | |
| Назва абстракції | Відповідний клас фреймворку simulation, його спадкоємець, або новий клас |
| Бригада | Клас Бригада, спадкоємець класу Actor |
| Авто | Клас Авто, спадкоємець класу Actor |
| Черга працюючих бригад | Клас QueueForTransaction |
| Черга на розвантаження | Клас QueueForTransaction |
| Черга бригад працюючих надурочно | Клас QueueForTransaction |

# РЕАЛІЗАЦІЯ СИСТЕМИ

## Реалізація шару подання

Основою шару подання є інтерфейс користувача, який представлено на рисунках 2.1, 2.2. Інтерфейс був створений відповідно до завдань, що були визначені вище. Інтерфейс спроектовано як сукупність трьох основних панелей.

Основою інтерфейсу є компонент Panel.

Ліву частину цієї панелі займає панель для розміщення компонентів для налаштування моделі і присутня на екрані у всіх режимах роботи.. У якості менеджера компоновки вибрано AbsoluteLayout.

Праворуч розташований компонент TabbedPane, на закладках якого одна під одною розташовані панелі, що з’являються після вибору відповідного режиму роботи. Для панелей також застосовано менеджер компоновки AbsoluteLayout.

### Режим перегляду технічного завдання

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

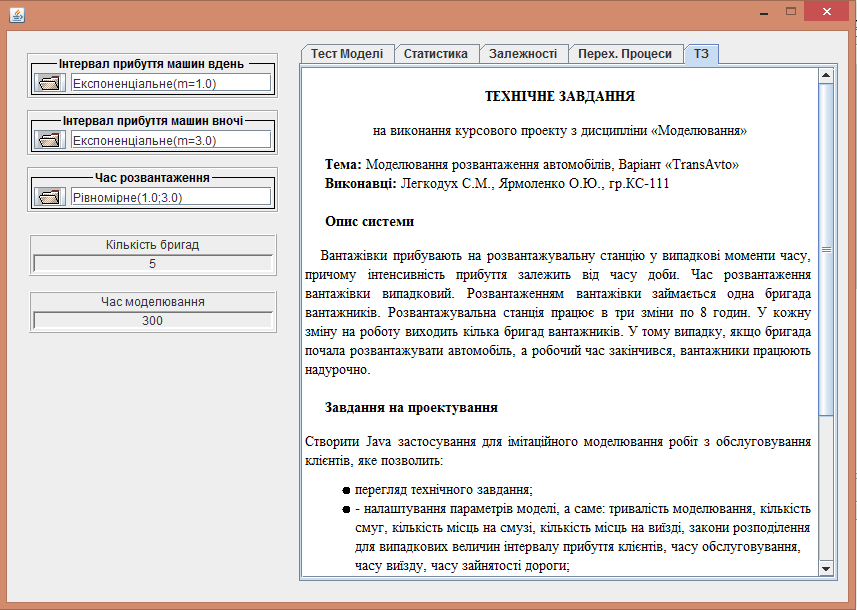


Рисунок 2.1 – Інтерфейс користувача моделі у режимі перегляду технічного завдання

Відображення файлу реалізується кодом, що наведено у лістингу 2.1.

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

|  |
| --- |
| **private** **void** TZ() {  String str = "tz.html";  URL url = getClass().getResource(str);  **try** {  textPane.setPage(url);  textPane.setEditable(**false**);  textPane.setHighlighter(**null**);    } **catch** (IOException e) {  e.printStackTrace();  System.***err***.println("Problems with file " + str);  }  } |

### 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.

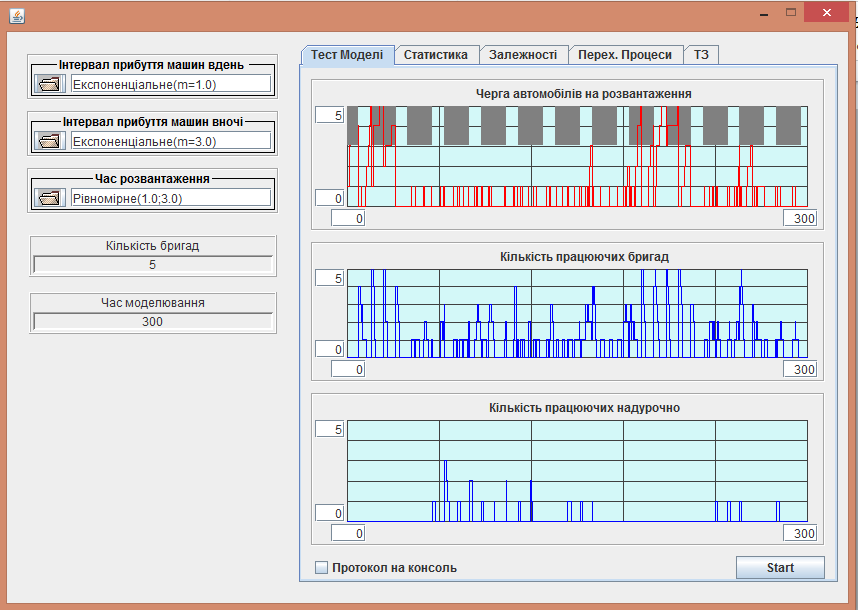


Рисунок 2.2 – Інтерфейс користувача моделі у режимі тестування

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

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

З цією ж метою використовується і подія панелі componentShown.

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

Текст методу наведено у лістингу 2.2.

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

|  |
| --- |
| **private** **void** startTest() {      Dispatcher dispatcher = **new** Dispatcher();  dispatcher.addDispatcherFinishListener(  ()->btnStart.setEnabled(**true**));  IModelFactory factory = (d)-> **new** Model(d, **this**);  Model model = (Model) factory.createModel(dispatcher);  model.initForTest();  dispatcher.start();  btnStart.setEnabled(**false**);  diagramQueueForRozvantazh.clear();  diagramCountWorkNadurochno.clear();  diagramCountWorkBrigad.clear();    } |

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 «Start» блокується на період моделювання, а розблоковує її об’єкт, що створено за допомогою лямбда функції, який реагує на завершення роботи диспетчера.

### Режим накопичення та відображення статистичних даних

Режим “Статистика ” 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.

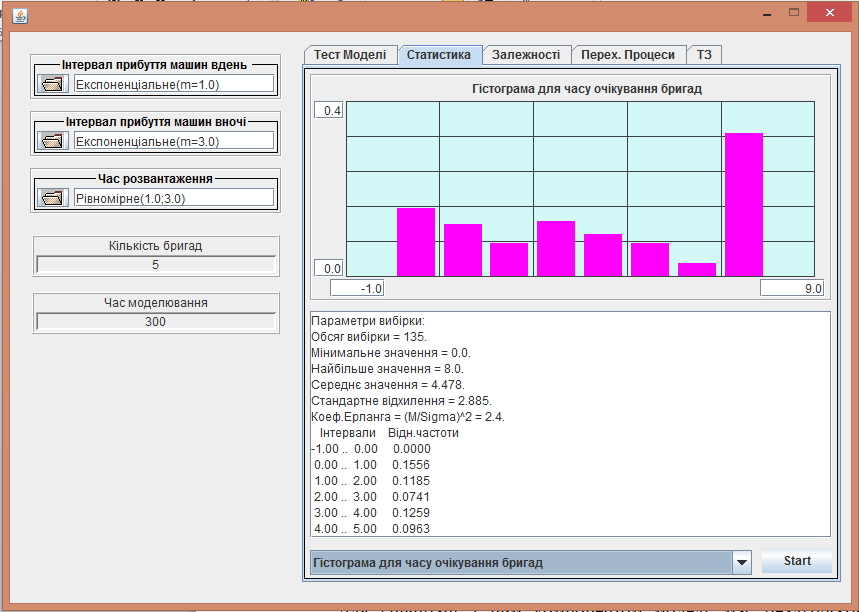


Рисунок 2.3 – Інтерфейс користувача моделі у режимі відображення статистичних даних про роботу моделі

### Режим проведення однофакторних багаторівневих експериментів

У режимі “Залежність”, рисунок 2.4, з моделлю можна проводити однорівневі 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.

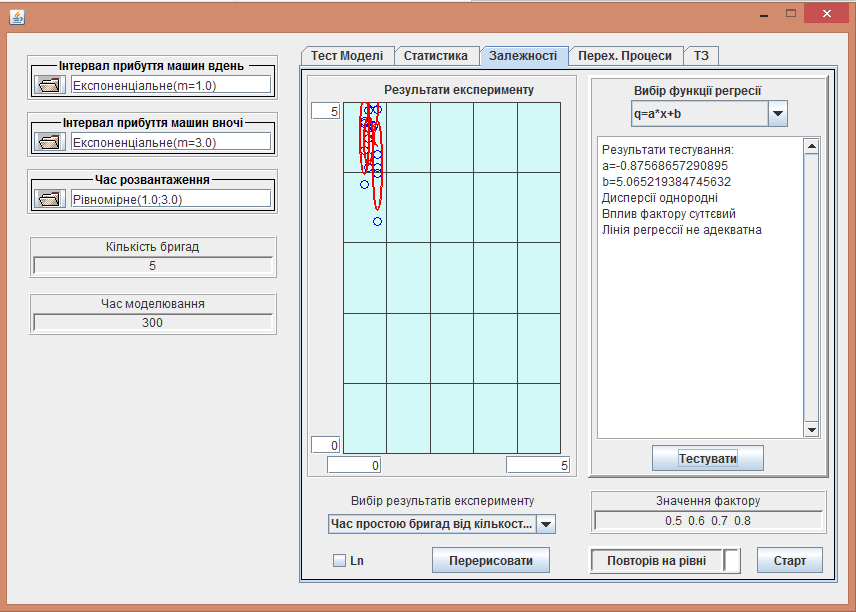


Рисунок 2.4 – Інтерфейс користувача моделі у режимі проведення багаторівневих однофакторних експериментів

### 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.

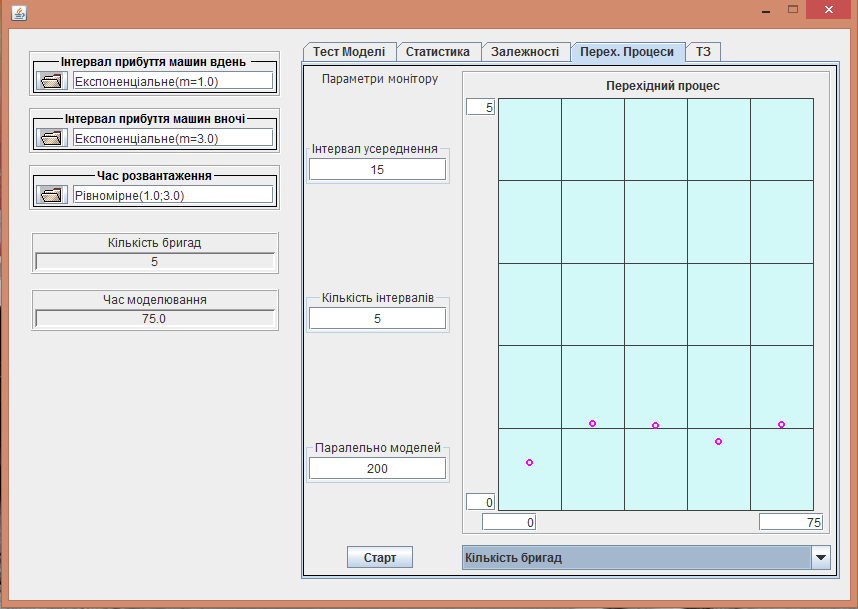


Рисунок 2.5 – Інтерфейс користувача моделі у режимі дослідження перехідних процесів

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.

### 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.

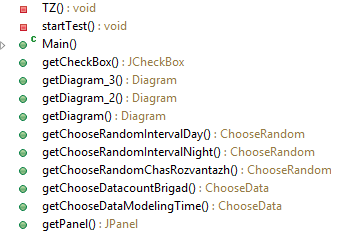


Рисунок 2.6 – Публічний інтерфейс шару подання

## Реалізація шару моделі

Цей шар складається тільки з класу моделі.

### Клас Model

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

|  |
| --- |
| **private** Dispatcher dispatcher;  **private** Main gui;  **private** Timer timer;  **private** GeneratorAvto avto;  **private** QueueForTransactions<Double> queueAvto;  **private** QueueForTransactions<Brigada> queueBrigad;  **private** QueueForTransactions<Brigada> queueBrigadNadurochno;  **private** Histo histoForBrigadWait;  **private** DiscretHisto dHistoAvtoServiceTime;  **private** **double** finishTime; |

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** Model(Dispatcher d, Main g) {  // **TODO** Auto-generated constructor stub  **if** (d == **null** || g == **null**) {  System.***out***.println("Не визначено диспетчера або GUI для RgrModel");  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()

|  |
| --- |
| **private** **void** componentsToStartList() {  // **TODO** Auto-generated method stub  // Передаємо акторів диспетчеру  dispatcher.addStartingActor(getTimer());  dispatcher.addStartingActor(getAvto());    } |

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 getGenAuto(), 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 створення об’єкту авто

|  |
| --- |
| **public** GeneratorAvto getAvto(){  **if**( avto ==**null**){  avto = **new** GeneratorAvto("Avto", gui, **this**);  }  **return** avto;  } |

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 Бригад, лістинг 2.7.

Лістинг 2. 7 – Метод створення бригад

|  |
| --- |
| **public** MultiActor getMultiCassir() {  **if** (multiCassir == **null**) {  multiCassir = **new** MultiActor();  multiCassir.setNameForProtocol("MultiActor dlia brigadu kassirov");  multiCassir.setOriginal(**new** Cassir("Kassir", gui, **this**));  multiCassir.setNumberOfClones(gui.getChooseData\_1().getInt());  }  **return** multiCassir;  } |

Об’єкти для черг створюються також інакше. У класі QueueForTransactions відсутні конструктори з параметрами. Тому посилання, що необхідні чергам слід передавати через методи set…(). Для прикладу наведемо методи getQueueToBrigad (), лістинг 2.8.

Лістинг 2.8 – Метод створення черги до бригад

|  |
| --- |
| **public** QueueForTransactions getQueueBrigad() {  **if**(queueBrigad == **null**){  queueBrigad = **new** QueueForTransactions<Brigada>("QueueBrigad",dispatcher);  }  **return** queueBrigad;  } |

Лістинг 2.9 – Метод створення черги надурочних робіт

|  |
| --- |
| **public** QueueForTransactions getQueueBrigadNadurochno() {  **if**(queueBrigadNadurochno == **null**){  queueBrigadNadurochno = **new** QueueForTransactions<Brigada>("QueueBrigadNadurochno",dispatcher);  }  **return** queueBrigadNadurochno;  } |

Аналогічно створюються об’єкти для гістограм. Як приклад методу для створення об’єкта для гістограм наведемо метод getHistoAvtoServiceTime (), лістинг 2.11.

Лістинг 2.11 – Метод створення об’єкта для гістограми

|  |
| --- |
| **public** DiscretHisto getHistoAvtoServiceTime() {  // **TODO** Auto-generated method stub  **if** (dHistoAvtoServiceTime == **null**) {  dHistoAvtoServiceTime = **new** DiscretHisto();  }  **return** dHistoAvtoServiceTime;  } |

Завершується робота над бригадою 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 методі initForStat() блокується виведення протоколу на консоль.

У інших режимах може виникнути потреба реалізації методів деяких 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.

#### **Реалізація інтерфейсу IStatisticsable**

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

Метод getStatistics(), лістинг 2.12, повертає асоціативний масив, який містить гістограми з накопиченої статистикою. На підставі цих даних компонент StatistcsManager формує результати.

Лістинг 2.12 – Метод, що повертає статистичні дані про роботу моделі

|  |
| --- |
| **public** Map<String, IHisto> getStatistics() {  // **TODO** Auto-generated method stub  Map<String, IHisto> map = **new** HashMap<>();  map.put("Гістограма для часу очікування бригад",getHistoBrigadWait());  map.put("Гістограма для часу обслуговування Авто", getHistoAvtoServiceTime());    **return** map;  } |

#### **Реалізація інтерфейсу IExperimentable**

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

Лістинг 2.13 – Метод для передачі моделі значення фактору

|  |
| --- |
| **public** **void** initForExperiment(**double** factory) {  // **TODO** Auto-generated method stub  // queueBrigad.setNumberOfClones((int) factory);  } |

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

Лістинг 2.14 – Метод для повернення результатів експерименту

|  |
| --- |
| **public** Map<String, Double> getResultOfExperiment() {  // **TODO** Auto-generated method stub  Map<String, Double> resultMap = **new** HashMap<>();  resultMap.put("Час простою авто від їх кількості", getHistoAvtoServiceTime()  .getAverage());  resultMap.put("Час простою бригад від кількості авто", getHistoBrigadWait()  .average());    **return** resultMap;  } |

#### **Реалізація інтерфейсу ITransProcesable**

Метод initForTrans(double) цього інтерфейсу, лістинг 2.15, 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 2.15 – Реалізація методу підготовки до дослідження перехідних процесів

|  |
| --- |
| **public** **void** initForTrans(**double** finishTime) {  // **TODO** Auto-generated method stub  getAvto().setFinishTime(finishTime);  getTimer().setFinishTime(finishTime);  gui.getChooseDataModelingTime().setDouble(finishTime);  } |

Метод resetTransAccum() цього інтерфейсу, лістинг 2.16, компонент використовує для ініціалізації накопичувачів інформації про середнє значення черги на початку кожого інтервалу накопичення даних.

Лістинг 2.16 – Реалізація методу ініціалізації накопичувачів

|  |
| --- |
| **public** **void** resetTransAccum() {  // **TODO** Auto-generated method stub  getQueueAvto().resetAccum();  getQueueBrigad().resetAccum();  getQueueBrigadNadurochno().resetAccum();  } |

Через метод getTransResult() цього інтерфейсу, лістинг 2.17, компонент отримує середнє значення черги для кожого інтервалу накопичення даних.

Лістинг 2.17 – Реалізація методу повернення результатів накопичення

|  |
| --- |
| **public** Map<String, Double> getTransResult() {  // **TODO** Auto-generated method stub  Map<String, Double> transMap = **new** HashMap<>();  transMap.put("Черга Авто", getQueueAvto().getAccumAverage());  transMap.put("Кількість бригад", getQueueBrigad().getAccumAverage());  transMap.put("Бригади працюючі надурочно", getQueueBrigadNadurochno().getAccumAverage());  **return** transMap;  } |

#### **Перелік публічних методів моделі**

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

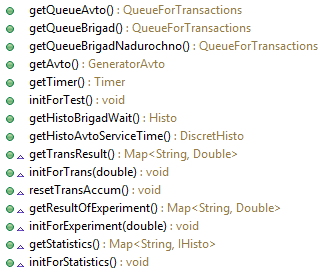


Рисунок 2.7 – Публічний інтерфейс моделі

## Реалізація компонентів моделі

Класи, що потребують реалізації у даному проекті – це класи для дороги, бригади, авто та генератор авто. 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()

### Клас Brigada

Об’єкти цього класу представляють абстракцію бригади. У лістингу 2.18 наведено текст класу з коментарями.

Лістинг 2.18 – Клас Cassir

|  |
| --- |
| **package** kursa4;  **import** java.util.function.BooleanSupplier;  **import** process.Actor;  **import** process.DispatcherFinishException;  **import** process.QueueForTransactions;  **import** rnd.Randomable;  **import** widgets.ChooseRandom;  **public** **class** Brigada **extends** Actor{      **private** BooleanSupplier isAvtoOrFinishWork;  **private** BooleanSupplier finishWork;  **private** Randomable rnd;  **private** Main gui;  **private** Model model;  **private** QueueForTransactions<Double> quequeToBrigada;      **public** Brigada(String string, Main gui, Model model) {  // **TODO** Auto-generated constructor stub  **this**.gui = gui;  **this**.model = model;  quequeToBrigada = model.getQueueAvto();  rnd = gui.getChooseRandomChasRozvantazh();  setHistoForActorWaitingTime(model.getHistoBrigadWait());    }  @Override  **protected** **void** rule() {  **double** nowTime = getDispatcher().getCurrentTime();  finishWork = ()->nowTime+8==getDispatcher().getCurrentTime();  isAvtoOrFinishWork = ()-> quequeToBrigada.size()>0||finishWork.getAsBoolean();  // **TODO** Auto-generated method stub    // стартувати генератор бригад  **try** {  waitForCondition(isAvtoOrFinishWork, "Має бути вантажівка в черзі");  //працюючих  } **catch** (DispatcherFinishException e) {  // **TODO** Auto-generated catch block  **return**;  }  **if**(finishWork.getAsBoolean()){  **return**;  }  // Забираємо вантажівку на обслуговування  **double** bt = quequeToBrigada.removeFirst();    model.getQueueBrigad().add(**this**);  **double** holdTime = rnd.next();  holdForTimeOrWaitForCondition(holdTime, finishWork, "чекаємо кінця зміни");  model.getQueueBrigad().remove(**this**);  **if**(finishWork.getAsBoolean()){    //стати в чергу надурочно  model.getQueueBrigadNadurochno().add(**this**);  getDispatcher().printToProtocol(getNameForProtocol() + " стає в чергу надурочно");  **double** nadTime = getActivateTime()-getDispatcher().getCurrentTime();    holdForTime(nadTime);  model.getQueueBrigadNadurochno().remove(**this**);  }    **double** time = getDispatcher().getCurrentTime()-bt;  model.getHistoAvtoServiceTime().add(time);  }    } |

### Клас Factory

У лістингу 2.19 наведено текст класу з коментарями.

Лістинг 2. 19 – Клас Factory

|  |
| --- |
| **package** kursa4;  **import** process.Dispatcher;  **import** process.IModelFactory;  **public** **class** Factory **implements** IModelFactory {  **private** Main gui;  **public** Factory(Main gui) {  **this**.gui = gui;  }  **public** Model createModel(Dispatcher dispatcher) {  Model newModel = **new** Model(dispatcher, gui);  **return** newModel;  }  } |

### Клас GeneratorAuto

Об’єкти цього класу представляють абстракцію генератора автомобілів. У лістингу 2.21 наведено текст класу з коментарями.

Лістинг 2. 21 – Клас GeneratorAuto

|  |
| --- |
| package kursa4;  import java.util.function.BooleanSupplier;  import process.Actor;  import process.QueueForTransactions;  import widgets.ChooseRandom;  public class GeneratorAvto extends Actor {    private Timer timer;  private ChooseRandom rndDay;  private ChooseRandom rndNight;  private QueueForTransactions queue;  private BooleanSupplier isAvto;  private double finishTime;  /////////////////  //konstructor  /////////////////  public GeneratorAvto(String string, Main gui, Model model) {  // TODO Auto-generated constructor stub  timer = model.getTimer();  rndDay = gui.getChooseRandomIntervalDay();  rndNight = gui.getChooseRandomIntervalNight();  queue = model.getQueueAvto();  finishTime = gui.getChooseDataModelingTime().getDouble();  }  /////////////////  //rule  //////////////////  @Override  protected void rule() {  ChooseRandom rnd = null;  isAvto = ()->queue.size()>0;  while(getDispatcher().getCurrentTime() <= finishTime){  if(timer.getTime()>=8 && timer.getTime()<=16){  rnd = rndDay;}    else{  rnd = rndNight;  }  holdForTime(rnd.next());  getDispatcher().printToProtocol(" " + getNameForProtocol() + " додає час");  queue.add(getDispatcher().getCurrentTime());  }  }  public void setFinishTime(double finishTime2) {  // TODO Auto-generated method stub  this.finishTime =finishTime;  }    } |

# РЕЗУЛЬТАТИ ВИКОНАННЯ ПРОЕКТУ

## 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.

### 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 кількості смуг, кількості місць на смузі, кількості місць на смузі 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 автомобілів до бригад були невеликими,кількість бригад надурочно не значна.

Результати роботи моделі у цьому режимі показано на рисунку 3.1.

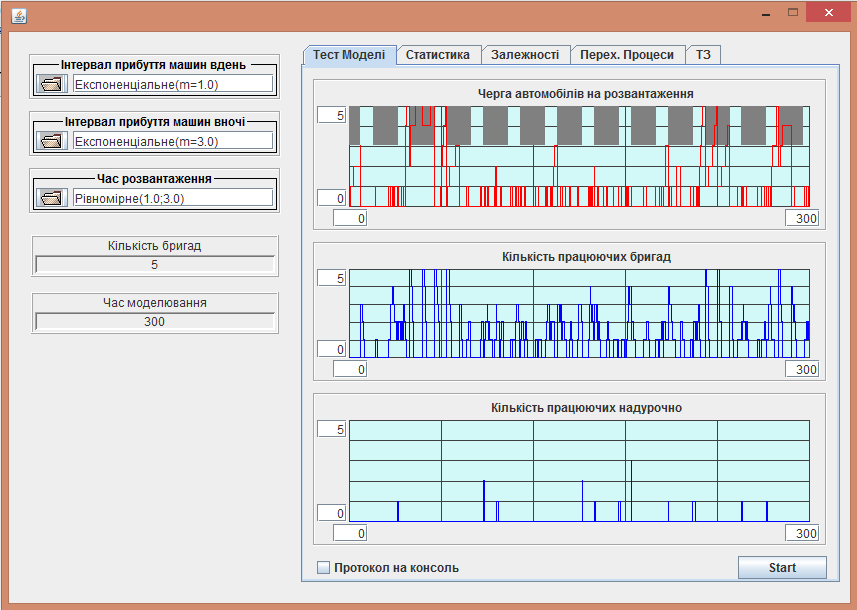


Рисунок 3.1 – Результати роботи моделі у нормальному режимі

На рисунку 3.2 показано результати моделювання системи при недостатній кількості бригад. У цьому режимі кількість авто швидко зростає і може досягнути критичного розміру. Авто, які хотіли про їхати, але не мають змоги це зробити, потрапляють до черги втрачених клієнтів.

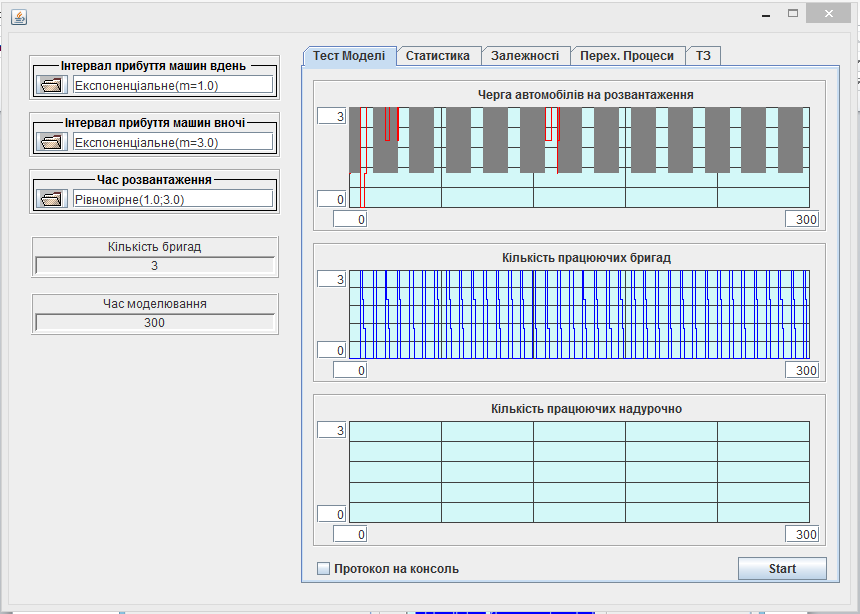


Рисунок 3.2 – Результати роботи моделі при недостатній кількості бригад

### 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 довело його придатність для отримання статистичних даних про довжину черги.

На рисунках 3.4 – 3.6 наведені результати отримання статистичних характеристик для довжини черги для часу нормального, недостатнього та стандартного часу моделювання.

Час моделювання будимо вимірювати в хвилинах.

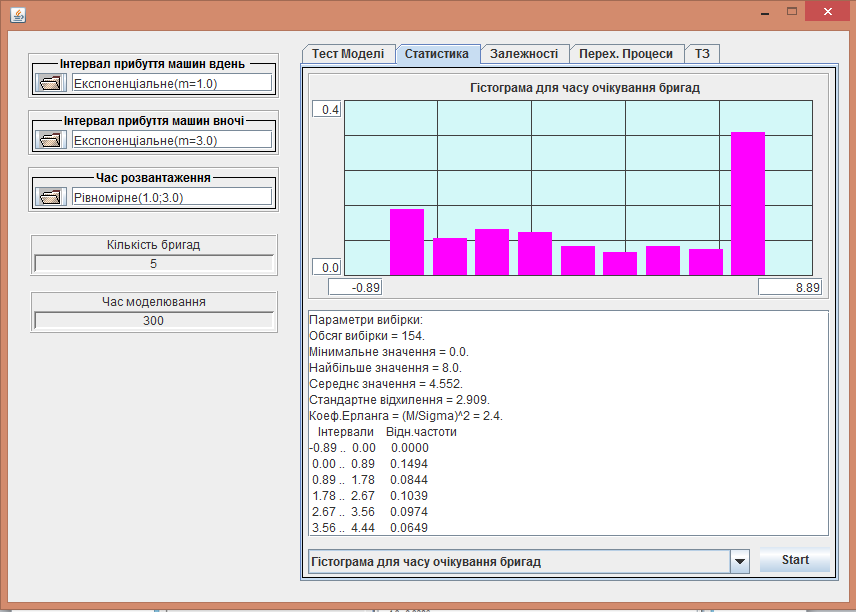


Рисунок 3.4 – Результати роботи моделі при стандартному часі моделювання

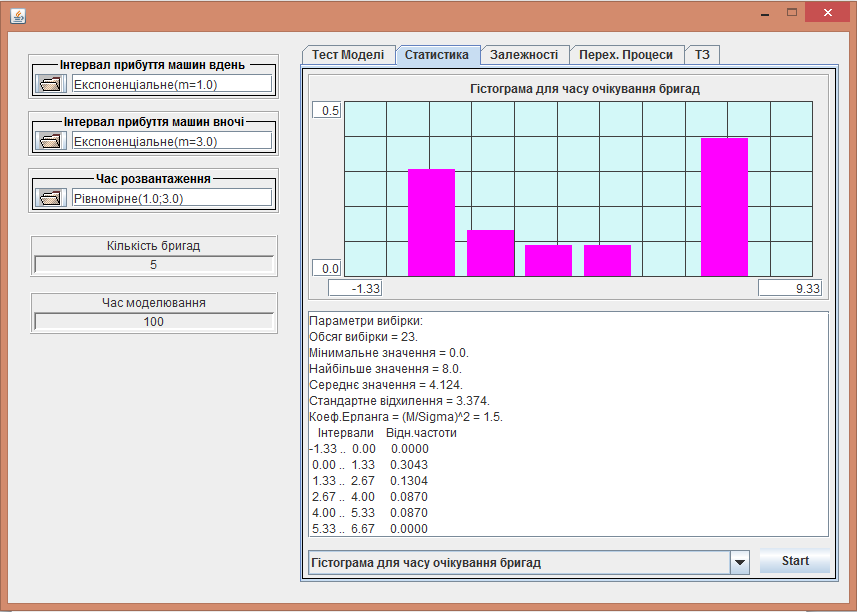


Рисунок 3.5 – Результати роботи моделі при недостатньому часі моделювання

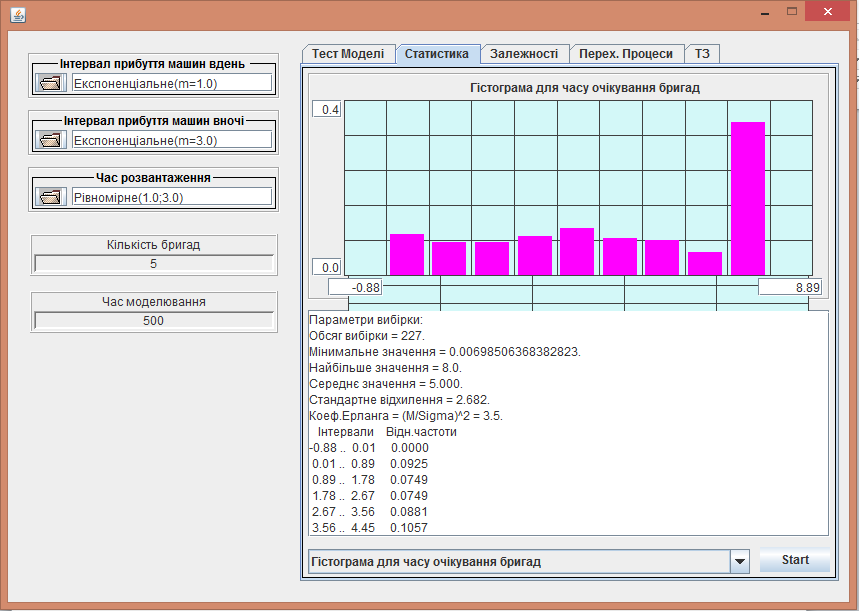


Рисунок 3.6– Результати роботи моделі при достатньому часі моделювання

### Режим «Однофакторні багаторівневі експерименти»

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

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

На рисунках 3.9 – 3.10 наведені результати відповідних екпериментів.

Виявилося, що оптимальним числом для кількісті бригад є число 5.

Якщо кількість бригад менша, то зростає час простою автомобіля.

Якщо кількість бригад збільшується то зменшується розмір черги автомобілів і відповідно зменшується час їх простою.

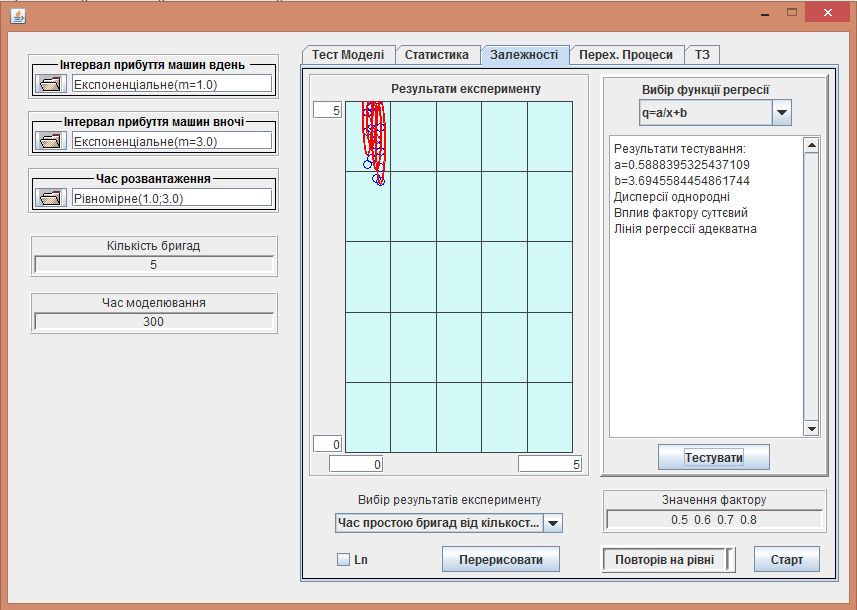


Рисунок 3.9 – Результати дослідження впливу кількості бригад

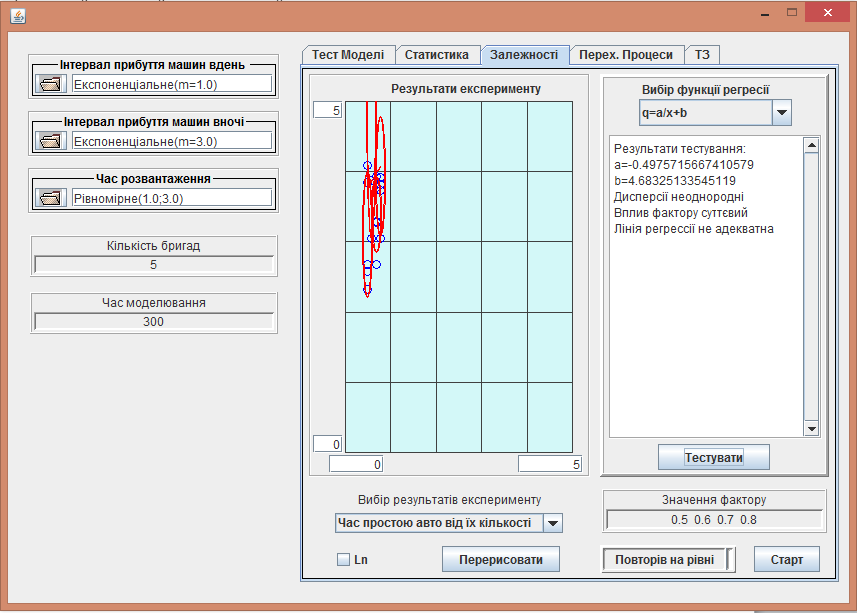


Рисунок 3.10 – Результати дослідження впливу довжини черги на виїзд на час простою автомобіля від їх кількості

### Режим «Дослідження перехідних процесів»

На рисунках 3.11 – 3.13 наведені результати відповідних експериментів.

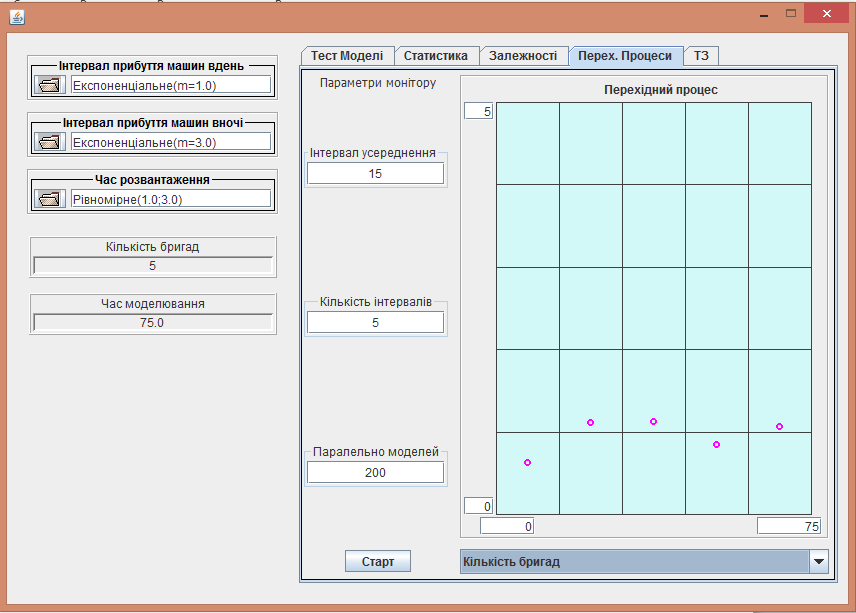


Рисунок 3.11 – Перехідний процес для кількості бригад

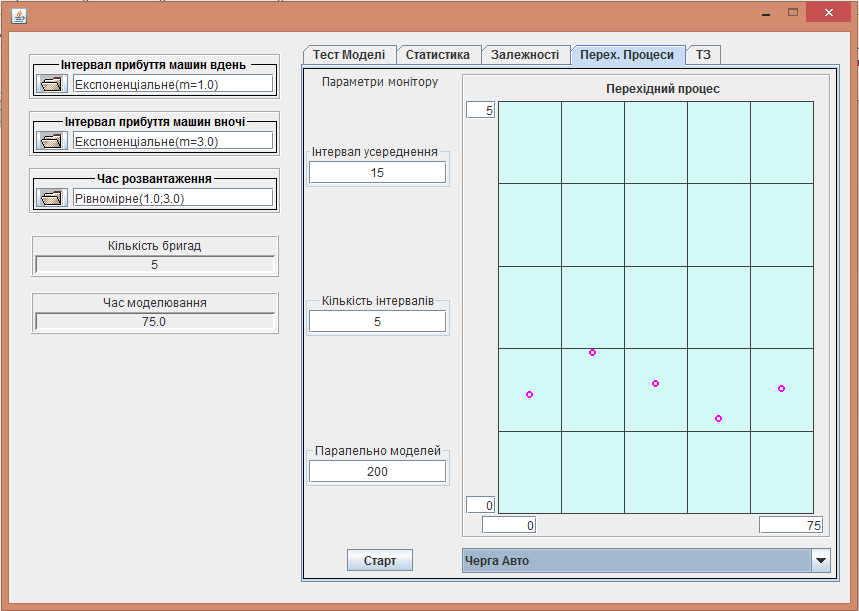


Рисунок 3.12 – Перехідний процес для черги авто

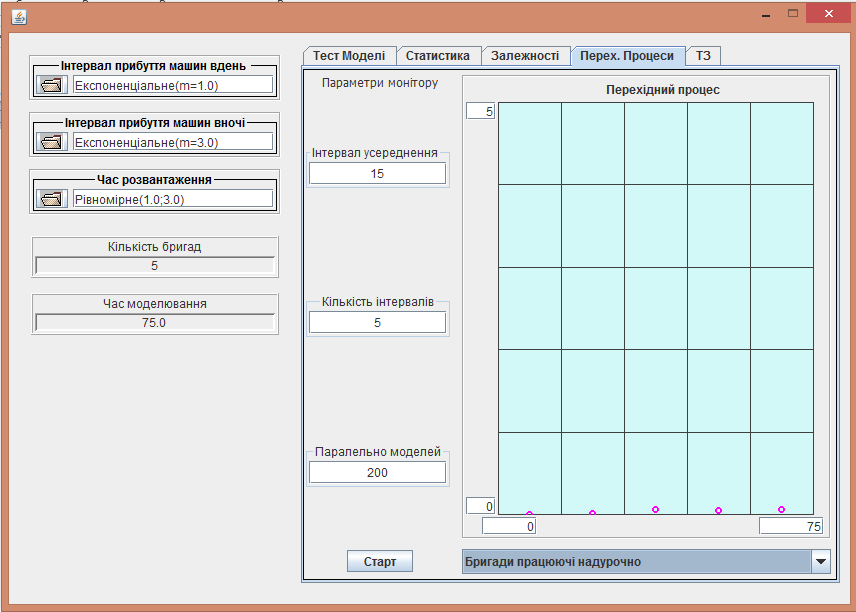


Рисунок 3.13 – Перехідний процес для бригад працюючих надурочно

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

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

## Система контролю версіями Git

Git – розподілена система керування версіями файлів та спільної роботи. Git є однією з найефективніших, надійних і високопродуктивних систем керування версіями, що надає гнучкі засоби нелінійної розробки, що базуються на відгалуженні і злитті гілок. Основні її переваги розподілу і використання бінарних дифів, що спрощує роботу з системою декількох користувачів одночасно, і спрощує зберігання бінарних файлів, ресурсів.

Головна відмінність Git від будь-яких інших СКВ (наприклад, Subversion і їй подібних) це те, як Git дивиться на дані. В принципі, більшість інших систем зберігає інформацію як список змін для файлів. Ці системи (CVS, Subversion, Perforce, Bazaar та інші) відносяться до збережених даними як до набору файлів і змін,зроблених для кожного з цих файлів в часі. Git не зберігає свої дані у такому вигляді. Замість цього Git вражає збережені дані набором зліпків невеликої файлової системи. Кожен раз, коли фіксується поточна версія проекту, Git, по суті, зберігає зліпок того, як виглядають всі файли проекту на поточний момент. Заради ефективності, якщо файл не змінювався, Git не зберігає файл знову, а робить посилання на раніше збережений файл.

Для забезпечення цілісності історії та стійкості до змін заднім числом використовуються криптографічні методи, також можлива прив'язка цифрових підписів розробників до тегів і комітів.

Приклади команд:

* git init ініціалізація репозитарію;
* git status стан (що було редаговане, що додано);
* git add додати в індекс всі зміни;
* git commit вчинення комітів;
* git revert скасування змін, зроблених у минулому окремим комітів;
* git branch створення, перерахування та видалення гілок;
* git clone створення копії (віддаленого) репозитарія;
* git push вносимо зміни в віддалений репозитарій (віддалену гілку).

На рисунку 3.1 представлений скриншот разміщеного на GitHub проекту.

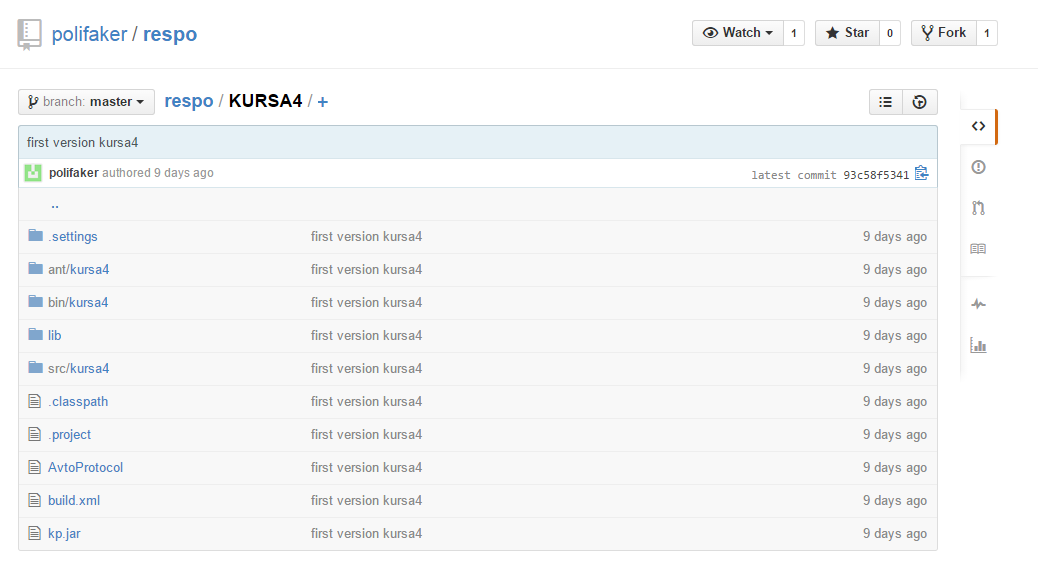


Рисунок 3.13 – Розміщений проект на GitHub

## Система збирання проектів

Apache Ant (англ. ant – мураха і водночас акронім – «Another Neat Tool») – java-утиліта для автоматизації процесу збирання програмного продукту.

Ant – платформонезалежний аналог UNIX-утиліти make, але з використанням мови Java, він вимагає платформи Java, і краще пристосований для Java-проектів. Найпомітніша безпосередня різниця між Ant та Make те, що Ant використовує XML для опису процесу збирання і його залежностей, тоді як Make має свій власний формат Makefile. За замовчуванням XML-файл називається build.xml.

Ant був створений в рамках проекту Jakarta, сьогодні – самостійний проект першого рівня Apache Software Foundation.

Перша версія була розроблена інженером Sun Microsystems Джеймсом Девідсоном (James Davidson), який потребував утиліти подібної make, розробляючи першу референтну реалізацію J2EE. Приклади завдань:

* Javac компіляція Java-коду
* Copy копіювання файлів
* Delete видалення файлів і директорій
* Move переміщення файлів і директорій
* Replace зміщення фрагментів тексту у файлах
* JUnit автоматичний запуск Unit-тестів
* Exec виконання зовнішньої команди
* Zipс творення архіву у форматі zip
* Javadoc формування javadoc

Зміст build.xml файлу, який використовувався для збирання проекту представлений в лістингу 3.1.

Лістинг 3.1 – Зміст build.xml файлу

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <project default="run" name="kp">  <property name="lib.dir" value="./lib"/>  <path id="classpath">  <fileset dir="${lib.dir}" includes="\*\*/\*.jar"/>  </path>    <path id="build.classpath">  <fileset dir=".">  <include name="lib/\*.jar"/>  </fileset>  </path>  <pathconvert property="manifest.classpath" pathsep=" ">  <path refid="build.classpath"/>  <mapper>  <chainedmapper>  <flattenmapper/>  <globmapper from="\*.jar" to="lib/\*.jar"/>  </chainedmapper>  </mapper>  </pathconvert>  <target name="buildjar" depends="compile">  <delete file="kp.jar"></delete>  <mkdir dir="ant/kursa4/lib"/>  <copy includeemptydirs="false" todir="ant/kursa4/lib">  <fileset dir="./lib">  </fileset>  </copy>    <jar jarfile="kp.jar">  <fileset dir="ant" />  <manifest>  <attribute name="Main-Class" value="kursa4.Main"/>  <attribute name="Class-Path" value="${manifest.classpath}"/>  </manifest>  </jar>  </target>    <target name="run" depends="buildjar">  <java jar="kp.jar" fork="true" >  <classpath>  <path refid="classpath"/>  <pathelement location="kp.jar"/>  </classpath>  </java>  </target>    <target name="compile">  <delete dir="ant"></delete>  <mkdir dir="ant"/>  <javac srcdir="src" destdir="ant" includeantruntime="false">  <classpath>  <path refid="classpath"></path>  </classpath>  </javac>  </target>    <target name="doc">  <javadoc sourcepath="src" destdir="doc"/>  </target>    </project> |

**ВИСНОВКИ**

В У процесі виконання курсової 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 фреймворку.

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.

Були отримані навики командної розробки. За допомогою системи контролю версій Git робота велась паралельно між усіма учасниками команди.

Для автоматизації процесу збирання проекту використовувався інструмент Ant, який дозволив полегшити процес створення jar архіву та виконання тестів.

При впровадженні всіх можливих поліпшень проект цілком може стати конкурентоспроможним продуктом. Проте на даному етапі проект є навчальним прикладом.

**СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ**

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 c.: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.

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**

|  |  |  |
| --- | --- | --- |
| **Позначення** | **Назва** | **Примітка** |
|  |  |  |
| КC 113124.10 | Документація |  |
| КС 113107.10 | Текст програми |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  | Комплекси |  |
|  | Середовище розробки |  |
|  | Eclipse Luna компанії |  |
|  | Oracle |  |
|  |  |  |
|  |  |  |
|  | ОС Windows 8.1 |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Додаток А**

«Текст програми»