МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАИНЫ

ЧЕРНІГІВСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ

КАФЕДРА ІНФОРМАЦІЙНИХ ТА КОМП’ЮТЕРНЫХ СИСТЕМ

**МОДЕЛЮВАННЯ ПОСІВНОЇ КОМПАНІЇ**

Розрахунково-графічна робота з дисципліни “Моделювання”

|  |  |  |
| --- | --- | --- |
| Виконавці  студенти гр. КІ-122 |  | Г.М. Тур  С. . Лобода |
| Перевірив  доцент |  | П.Г. Бивойно |

**2015**

**ТЕХНІЧНЕ ЗАВДАННЯ**

на виконання розрахунково– графічної роботи з дисципліни «Моделювання»

**Тема:** Моделювання посівної компанії,   
варіант BeanFeast

**Виконавці:** Тур Г.М., Лобода С. ., гр. КІ – 122.

**Опис системи**

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

**Завдання на проектування**

Створити Java застосування для імітаційного моделювання роботи ділянки тестування комп'ютерів, яке дозволить:

– налаштування параметрів моделі, а саме: кількість вантажівок, кількість сівалок, розмір кузова вантажівки, закони розподілення для випадкових величин часу завантаження вантажівки, часу розвантаження вантажівки, часу, який вантажівка витрачає на дорогу, часу посіву;

– проведення тестових запусків моделі при різних налаштуваннях з динамічною індикацією кількості вантажівок у дорозі, довжини черги вантажівок на складі, довжини черги сівалок за зерном, довжини черги вантажівок на розвантаження та можливості виведення протоколу роботи моделі під час тестових запусків;

– проведення експериментів для отримання статистичних характеристик для довжини черг, часу простою вантажівок, сівалок та навантажувача на складі та загального часу роботи посівної компанії.

– отримання залежностей для середніх значень вказаних вище параметрів від кількості вантажівок;

– отримання перехідних процесів для середніх значень довжини кожної з черг.

**Обсяг текстової документації**

Робота обсягом 15–20 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 2015 г.

**Зміст**

[1 АНАЛІЗ СИСТЕМИ, ЩО ПІДЛЯГАЄ МОДЕЛЮВАННЮ 4](#_Toc419136494)

[1.1 Опис системи 4](#_Toc419136495)

[1.2 Виділення основних абстракцій системи 4](#_Toc419136496)

[1.3 Аналіз активних абстракцій системи 6](#_Toc419136497)

[1.3.1 Конвеєр 6](#_Toc419136498)

[1.3.2 Тестувальник 8](#_Toc419136499)

[1.3.3 Налагоджувач 8](#_Toc419136500)

[1.3.4 Пакувальник 9](#_Toc419136501)

[1.4 Аналіз можливостей фреймворку Simulation для реалізації абстракцій системи 10](#_Toc419136502)

[2 РЕАЛІЗАЦІЯ СИСТЕМИ 12](#_Toc419136503)

[2.1 Реалізація шару подання 12](#_Toc419136504)

[2.1.1 Режим перегляду технічного завдання 12](#_Toc419136505)

[2.1.2 Режим тестування моделі 13](#_Toc419136506)

[2.1.3 Режим накопичення та відображення статистичних даних 13](#_Toc419136507)

[2.1.4 Режим проведення однофакторних багаторівневих експериментів 14](#_Toc419136508)

[2.1.5 Режим дослідження перехідних процесів 15](#_Toc419136509)

[2.1.6 Публічний програмний інтерфейс шару подання 15](#_Toc419136510)

[2.2 Реалізація шару моделі 15](#_Toc419136511)

[2.2.1 Клас TestPCModel 15](#_Toc419136512)

[2.3 Реалізація компонентів моделі 18](#_Toc419136513)

[2.3.1 Клас Computer 18](#_Toc419136514)

[2.3.2 Клас Conveyor 19](#_Toc419136515)

[2.3.3 Клас Tester 19](#_Toc419136516)

[2.3.4 Клас Setuper 20](#_Toc419136517)

[2.3.5 Клас Packer 21](#_Toc419136518)

[ВИСНОВКИ 22](#_Toc419136519)

# АНАЛІЗ СИСТЕМИ, ЩО ПІДЛЯГАЄ МОДЕЛЮВАННЮ

## Опис системи

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

## Виділення основних абстракцій системи

Аналізуючи опис системи, перш за все можна виділити таку абстракцію, як “Сівалка”. Ця абстракція моделює реальну сівалку, – в обов’язки якої входить сіяти зерно на полі.

Також можна виділити абстаркцію “Вантажіка”. Ця абстракція моделюе реальну вантажівку яка бере зерно на складі і відвозить його на поле і там завантажуе сівалку.

Наступною абстракцією є “Склад”, який моделюе реальний склад. Він призначений для завантаження ванатжівок зерном.

Абстракції “черга сівалок”, “черга вантажівок до розвантаження”, “черга вантажівок у дорозі” та “черга вантажівок на складі” моделюють черги сівалок та вантажівок, які чекають на завантаження та розвантаження відповідно.

Таблиця 4.1 – Абстракції системи, що входять до складу моделі

|  |  |
| --- | --- |
| Абстракція | Перелік завдань |
| Сівалки | Сіють зерно на полі. |
| Черга сівалок | Місце, де сівалки чекають на завантаження зерна. Відображувати статистичні характеристики довжини черг у графічному та текстовому вигляді. |
| Вантажівка | Перевозить зерно зі складу на поле і завантажуе сівалки. |
| Черга вантажівок до розвантаження | Місце, де вантажівки чекають на появу сівалки для розвантаження. Відображувати статистичні характеристики довжини черг у графічному та текстовому вигляді. |
| Ємність вантажівки | Кількість сівалок які може завантажити вантіжвка за один раз |
| Черга вантажівок у дорозі | Місце, де вантажівки знаходяться поки вони у дорозі. Відображувати статистичні характеристики довжини черг у графічному та текстовому вигляді. |
| Склад | Завантажуе вантажівки. |
| Черга вантажівок на складі | Місце, де вантажівки знаходяться поки вони завантажуються. Відображувати статистичні характеристики довжини черг у графічному та текстовому вигляді. |

## Аналіз активних абстракцій системи

Активними абстракціями системи є склад, вантажівка та сівалка. Розглянемо їх поведінку та представимо у вигляді діаграм діяльності.

### Склад

Головне завдання цієї абстракції – завантаження через випадкові проміжки часу вантажівок, які стоять у черзі до завантаження. Склад спочатку чекає на появу вантажівки в черзі, потім починає заповнювати кузов вантажівки порціями зерна через випадкові проміжки часу, коли вантажівка заповнена склад повертається до початкового стану. Свої дії склад має виконувати впродовж усього часу моделювання для будь якого режиму роботи моделі.

Для роботи конвеєру необхідні дані, перелік яких наведено у таблиці 1.2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблиця 1.2 – Атрибути абстракції Конвеєр | | | | |
| Назва поля | Клас | Призначення поля | Джерело | |
| finishTime | double | Час моделювання | GUI | |
| rnd | rnd.Randomable | Генератор випадкових чисел | GUI | |
| queueForLoading | qusystem.QueueForTransaction | Посилання на чергу вантажівок, що чекають завантаження | Model | |

Правила дії складу схематично можна представити у вигляді діаграми діяльності, рисунок 1.2.

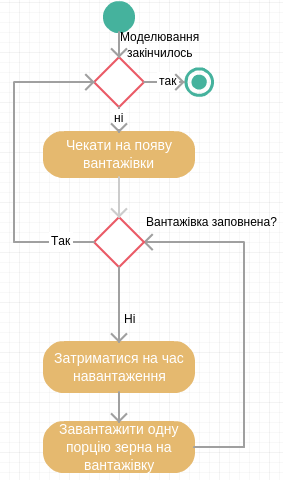


Рисунок 1.2 – Діаграма діяльності абстракції склад

### Вантажівка

Вантажівка працює з сівалками і чергою на навантаження. Спочатку сівалка їде на склад за зерном, витрачаючи на дорогу випадковий час. На складі вантажівка стає в чергу і чекає поки склад почне її завантажувати. Коли вантажівка заповнюється, вона їде на поле витрачаючи на це випадковий час. Вантажівка, що приїхала на поле, стає в чергу і чекає коли зявиться незавантажена сівалка. Пусту сівалку машина завантажує за деякий час і повертається в чергу, до тих пір коли її кузов не стане порожнім. Коли кузов став порожнім вантажівка їде назад на склад. Вантажівки працює впродовж усього часу моделювання.

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

|  |  |  |  |
| --- | --- | --- | --- |
| Таблиця 1.3 – Атрибути абстракції Тестувальник | | | |
| Назва поля | Клас | Призначення поля | Джерело |
| queueForLoading | qusystem.QueueForTransaction | Черга вантажівок на складі | Model |
| queueForTracktor | qusystem.QueueForTransaction | Черга вантажівок, які чекають сівалку | Model |
| queueForTrucksOnRoad | qusystem.QueueForTransaction | Вантажівки, що знаходяться у дорозі | Model |
| queuForBeans | qusystem.QueueForTransaction | Черга сівалок, які чекають на завантаження | Model |
| finishTime | double | Час моделювання | Gui |
| rnd | rnd.Randomable | Посилання на генератор випадкових чисел | Gui |
| Capacity | int | Эмність вантажівки | Gui |

Правила дії вантіжвки можна побачити на діаграмі діяльності вантажівки на рисунку 1.2.

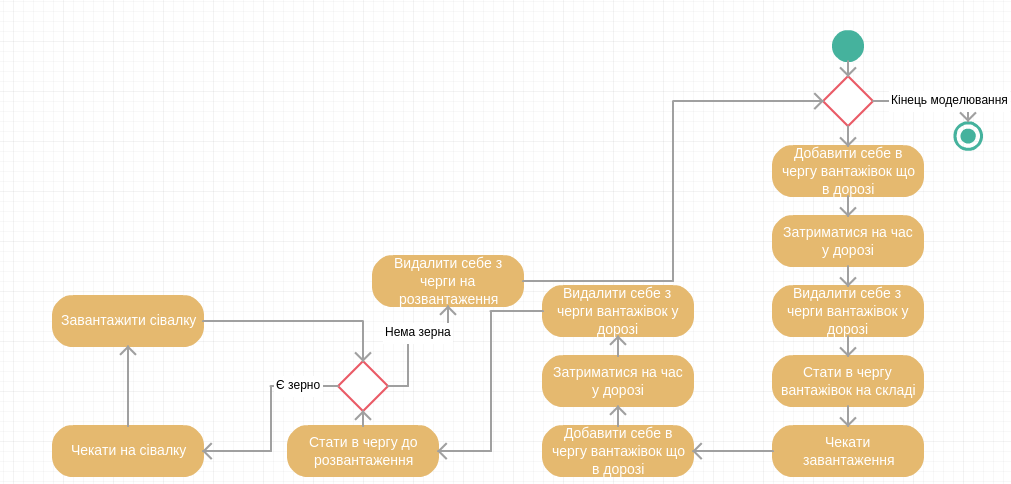


Рисунок 1.2 – Діаграма діяльності вантажівки

### Сівалка

Абстракція «сівалка» виконує посів зерна яке привозить вантажівка. Якщо є вільна сівалка, вантажівка навантажує її, після чого сівалка їде сіяти зерно затримуючись на деякий проміжок часу.

Сівалка працює впродовж усього часу моделювання.

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблиця 1.4 – Атрибути абстракції Налагоджувач | | | | |
| Назва поля | | Клас | Призначення поля | Джерело |
| queueForBeans | | qusystem.QueueForTransaction | Черга сівалок, що чекають на вантажівку | Model |
| finishTime | | double | Час моделювання | Gui |
| rnd | | rnd.Randomable | Посилання на генератор випадкових чисел | Gui |

Правила дії сівалки можна побачити на діаграмі діяльності конвеєра на рисунку 1.3.

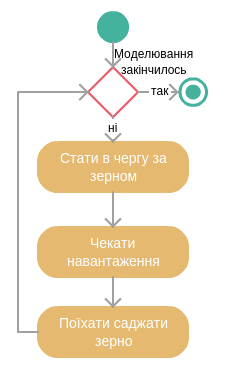


Рисунок 1.3 – Діаграма діяльності абстракції сівалка

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

Для створення бригади вантажівок та бригади сівалок можна використати клас process.MultiActor.

Абстракції «черга вантажівок на завантаження», «черга вантажівок у дорозі», «черга сівалок на завантаження» та «черга вантажівок на розвантаження» моделюють черги вантажівок та сівалок, які чекають на завантаження або розвантаження. Їх можна реалізувати як об’єкти класу queues.QueueForTransactions.

# РЕАЛІЗАЦІЯ СИСТЕМИ

## 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 панелі вибрано GridLayout. Праворуч розташований компонент TabbedPane, на закладках якого розташовані панелі, що з’являються після вибору відповідного режиму роботи.

### Режим перегляду технічного завдання

Перша закладка 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.

Відображення файлу реалізується кодом:

textPane = null;

String str="/rgr\_BF/tz.htm";

URL url = getClass().getResource(str);

### Режим тестування моделі

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

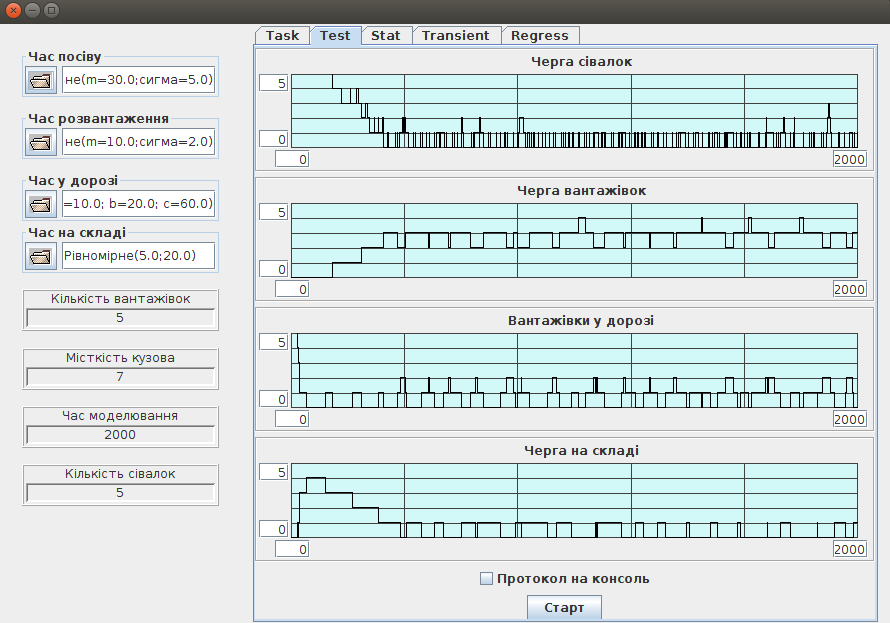


Рисунок 2.1 – Інтерфейс користувача моделі у режимі тестування

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

protected void onMouseStartClicked() {

btnStart.setEnabled(false);

diagramFeastQueue.clear();

diagramRoad.clear();

diagramSklad.clear();

diagramTruckQueue.clear();

Dispatcher dispatcher = new Dispatcher();

dispatcher.addDispatcherFinishListener(

()->btnStart.setEnabled(true));

IModelFactory factory = (d)-> new BeanFeastModel(d, this);

BeanFeastModel model =(BeanFeastModel) factory.createModel(dispatcher);

model.initForTest();

dispatcher.start();

}

### Режим накопичення та відображення статистичних даних

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

Для реалізації 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 BeanFeastModel(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.

Компонет ExperimentManager вимагає також, щоб модель реалізувала інтерфейс IStatisticsable.

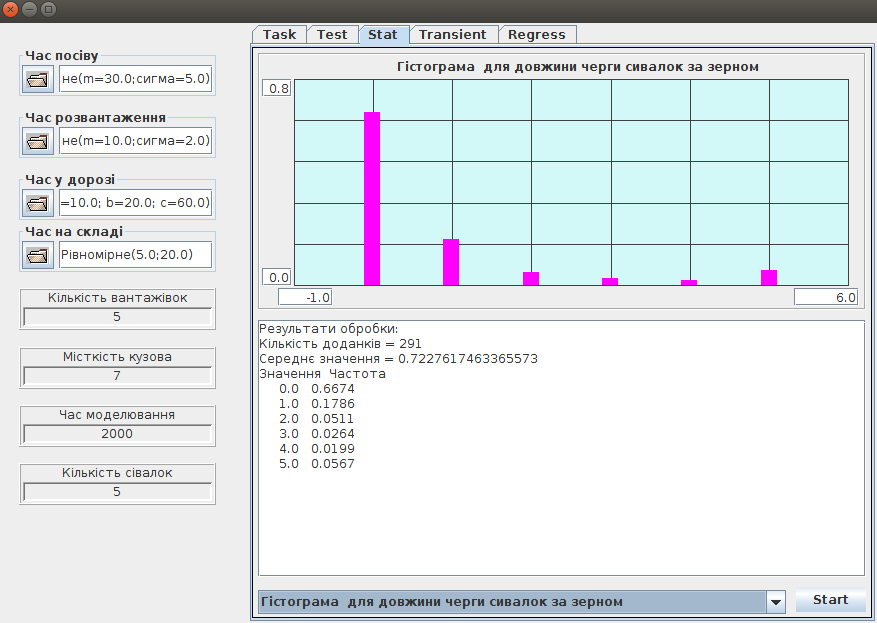


Рисунок 2.2 – Інтерфейс користувача моделі у режимі відображення статистичних даних про роботу моделі

### Режим проведення однофакторних багаторівневих експериментів

У режимі “Regres”, 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 компонент типу ExperimentManager.

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

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

Для співпраці з цим компонентом модель має реалізувати інтерфейс IExperimentable.

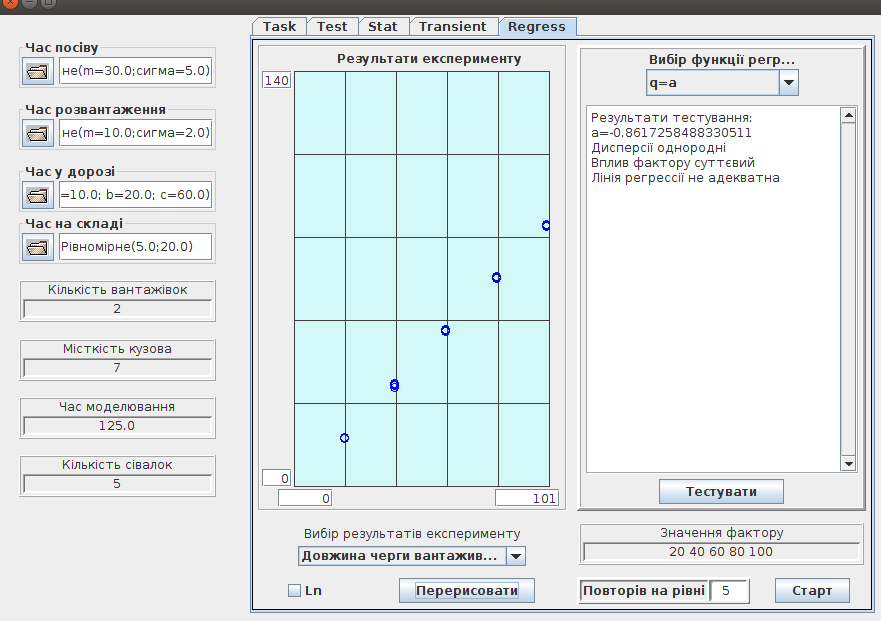


Рисунок 2.3 – Інтерфейс користувача моделі у режимі проведення багаторівневих однофакторних експериментів

### Режим дослідження перехідних процесів

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

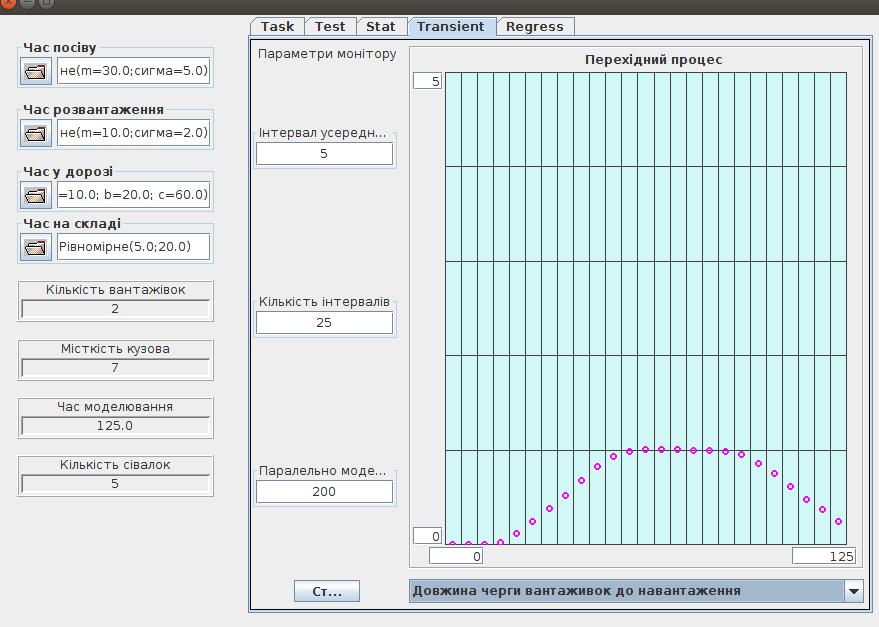


Рисунок 2.4 – Інтерфейс користувача моделі у режимі дослідження перехідних процесів

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

transProcessManager. setFactory((d)-> new BeanFeastModel(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.

## Реалізація шару моделі

Цей шар складається тільки з класу моделі.

### Клас BeanFeastModel

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.

public class BeanFeastModel implements IExperimentable, IStatisticsable, ITransProcesable {

private Dispatcher dispatcher;

private MainWindow gui;

private Truck truck;

private Tracktor tracktor;

private Sklad sklad;

private QueueForTransactions<Truck> queueForLoading;

private QueueForTransactions<Truck> queueForTracktor;

private QueueForTransactions<Tracktor> queueForBeans;

private QueueForTransactions<Truck> queueTrucksOnRoad;

private MultiActor multiTruck;

private MultiActor multiTracktor;

private DiscretHisto histoForQueueForLoading;

private DiscretHisto histForQueueForTracktor;

private DiscretHisto histoForQueueForBeans;

private DiscretHisto histoForQueueTrucksOnRoad;

private Histo histoForTruckWaitingTime;

private Histo histoForTracktorkWaitingTime;

private Histo histoForSkladWaitingTime;

public Histo getHistoForTruckWaitingTime() {

if (histoForTruckWaitingTime == null){

histoForTruckWaitingTime = new Histo();

}

return histoForTruckWaitingTime;

}

public Histo getHistoForTracktorWaitingTime() {

if (histoForTracktorkWaitingTime == null){

histoForTracktorkWaitingTime = new Histo();

}

return histoForTracktorkWaitingTime;

}

public Histo getHistoForSkladWaitingTime() {

if (histoForSkladWaitingTime == null){

histoForSkladWaitingTime = new Histo();

}

return histoForSkladWaitingTime;

}

public DiscretHisto getHistoForQueueForLoading() {

if (histoForQueueForLoading == null) {

histoForQueueForLoading = new DiscretHisto();

}

return histoForQueueForLoading;

}

public DiscretHisto getHistForQueueForTracktor() {

if (histForQueueForTracktor == null) {

histForQueueForTracktor = new DiscretHisto();

}

return histForQueueForTracktor;

}

public DiscretHisto getHistoForQueueForBeans() {

if (histoForQueueForBeans == null) {

histoForQueueForBeans = new DiscretHisto();

}

return histoForQueueForBeans;

}

public DiscretHisto getHistoForQueueTrucksOnRoad() {

if (histoForQueueTrucksOnRoad == null) {

histoForQueueTrucksOnRoad = new DiscretHisto();

}

return histoForQueueTrucksOnRoad;

}

public Map<String, IHisto> getStatistics() {

Map<String, IHisto> map = new HashMap<>();

map.put("Гістограма для довжини черги вантаживок до навантаження",

getHistoForQueueForLoading());

map.put("Гістограма для довжини черги грузовикив до розвантаження",

getHistForQueueForTracktor());

map.put("Гістограма для довжини черги сивалок за зерном",

getHistoForQueueForBeans());

map.put("Килькисть вантаживок у дорози",

getHistoForQueueTrucksOnRoad());

map.put("Час очикування вантаживок", getHistoForTruckWaitingTime());

map.put("Час очикування сивалок", getHistoForTracktorWaitingTime());

map.put("Час очикування складу", getHistoForSkladWaitingTime());

return map;

}

public MultiActor getMultiTruck() {

if (multiTruck == null) {

multiTruck = new MultiActor();

multiTruck.setNameForProtocol("MultiActor для бригади сивалок");

multiTruck.setOriginal(getTruck());

multiTruck.setNumberOfClones(gui.getChooseDataTruckCount().getInt());

}

return multiTruck;

}

public MultiActor getMultiTracktor() {

if (multiTracktor == null) {

multiTracktor = new MultiActor();

multiTracktor.setNameForProtocol("MultiActor для бригади сивалок");

multiTracktor.setOriginal(getTracktor());

multiTracktor.setNumberOfClones(gui.getChooseDataTracktorCount().getInt());

}

return multiTracktor;

}

public Truck getTruck() {

if (truck == null){

truck = new Truck("Грузовик", gui, this);

}

return truck;

}

public Tracktor getTracktor() {

if (tracktor == null){

tracktor = new Tracktor("Сивалка", gui, this);

}

return tracktor;

}

public BeanFeastModel(Dispatcher d, MainWindow mainWindow) {

if (d == null || mainWindow == null) {

System.out.println("Не визначено диспетчера або GUI для Model");

System.out.println("Подальша робота неможлива");

System.exit(0);

}

dispatcher = d;

gui = mainWindow;

componentsToStartList();

}

private void componentsToStartList() {

dispatcher.addStartingActor(getSklad());

dispatcher.addStartingActor(getMultiTracktor());

dispatcher.addStartingActor(getMultiTruck());

}

private Sklad getSklad() {

if (sklad == null){

sklad = new Sklad("Склад", gui, this);

}

return sklad;

}

public void initForTest() {

Painter p = gui.getDiagramFeastQueue().getPainter();

getQueueForBeans().setPainter(p);

getQueueTrucksOnRoad().setPainter(

gui.getDiagramRoad().getPainter());

getQueuForTracktor().setPainter(

gui.getDiagramTruckQueue().getPainter());

getQueuForLoading().setPainter(gui.getDiagramSklad().getPainter());

if (gui.getChckbxConsoleLog().isSelected()){

dispatcher.setProtocolFileName("Console");

}

}

public QueueForTransactions<Truck> getQueuForLoading() {

if (queueForLoading == null){

queueForLoading = new QueueForTransactions<Truck>();

queueForLoading.setNameForProtocol("Черга грузовиков на складе");

queueForLoading.setDiscretHisto(getHistoForQueueForLoading());

queueForLoading.setDispatcher(dispatcher);

}

return queueForLoading;

}

public QueueForTransactions<Truck> getQueuForTracktor() {

if (queueForTracktor == null){

queueForTracktor = new QueueForTransactions<Truck>();

queueForTracktor.setNameForProtocol("Черга грузовиков до сивалок");

queueForTracktor.setDiscretHisto(getHistForQueueForTracktor());

queueForTracktor.setDispatcher(dispatcher);

}

return queueForTracktor;

}

public QueueForTransactions<Tracktor> getQueueForBeans() {

if (queueForBeans == null){

queueForBeans = new QueueForTransactions<Tracktor>();

queueForBeans.setNameForProtocol("Черга сивалок");

queueForBeans.setDiscretHisto(getHistoForQueueForBeans());

queueForBeans.setDispatcher(dispatcher);

}

return queueForBeans;

}

public QueueForTransactions<Truck> getQueueTrucksOnRoad() {

if (queueTrucksOnRoad == null){

queueTrucksOnRoad = new QueueForTransactions<Truck>();

queueTrucksOnRoad.setNameForProtocol("Грузовики в дороге");

queueTrucksOnRoad.setDiscretHisto(getHistoForQueueTrucksOnRoad());

queueTrucksOnRoad.setDispatcher(dispatcher);

}

return queueTrucksOnRoad;

}

@Override

public void initForStatistics() {

}

@Override

public Map<String, Double> getResultOfExperiment() {

Map<String, Double> map = new HashMap<>();

map.put("Довжинa черги вантаживок до навантаження",

getHistoForQueueForLoading().getAverage());

map.put("Довжина черги грузовикив до розвантаження",

getHistForQueueForTracktor().getAverage());

map.put("Довжина черги сивалок за зерном",

getHistoForQueueForBeans().getAverage());

map.put("Килькисть вантаживок у дорози",

getHistoForQueueTrucksOnRoad().getAverage());

map.put("Час очикування вантаживок", getHistoForTruckWaitingTime().getAverage());

map.put("Час очикування сивалок", getHistoForTracktorWaitingTime().getAverage());

map.put("Час очикування складу", getHistoForSkladWaitingTime().getAverage());

return map;

}

@Override

public void initForExperiment(double arg0) {

multiTruck.setNumberOfClones((int) arg0);

}

@Override

public Map<String, Double> getTransResult() {

Map<String, Double> map = new HashMap<>();

map.put("Довжинa черги вантаживок до навантаження",

getQueuForLoading().getAccumAverage());

map.put("Дос вжина черги грузовикив до розвантаження",

getQueuForTracktor().getAccumAverage());

map.put("Довжина черги сивалок за зерном",

getQueueForBeans().getAccumAverage());

map.put("Килькисть вантаживок у дорози",

getQueueTrucksOnRoad().getAccumAverage());

return map;

}

@Override

public void initForTrans(double finishTime) {

getTracktor().setFinishTime(finishTime);

getTruck().setFinishTime(finishTime);

getSklad().setFinishTime(finishTime);

gui.getChooseDataSimTime().setDouble(finishTime);

}

@Override

public void resetTransAccum() {

getQueueForBeans().resetAccum();

getQueueTrucksOnRoad().resetAccum();

getQueuForLoading().resetAccum();

getQueuForTracktor().resetAccum();

}

}

## Реалізація компонентів моделі

### Клас Truck

public class Truck extends Actor {

BeanFeastModel beanFeastModel;

private double finishTime;

private int capacity;

private int loaded = 0;

private ChooseRandom roadRnd;

private ChooseRandom unloadRnd;

private QueueForTransactions<Truck> queueForLoading;

private QueueForTransactions<Truck> queueForTracktor;

private QueueForTransactions<Truck> queueTrucksOnRoad;

private QueueForTransactions<Tracktor> queuForBeans;

private BooleanSupplier isBodyFull;

private BooleanSupplier isTracktorsReady;

public Truck(String name, MainWindow gui, BeanFeastModel beanFeastModel) {

// this.gui = gui;

this.beanFeastModel = beanFeastModel;

this.capacity = gui.getChooseDataTruckCapacity().getInt();

this.finishTime = gui.getChooseDataSimTime().getDouble();

roadRnd = gui.getChooseRandomRoadTime();

// loadRnd = gui.getChooseRandomLoad();

unloadRnd = gui.getChooseRandomTracktor();

setNameForProtocol(name);

queueForLoading = beanFeastModel.getQueuForLoading();

queueForTracktor = beanFeastModel.getQueuForTracktor();

queueTrucksOnRoad = beanFeastModel.getQueueTrucksOnRoad();

queuForBeans = beanFeastModel.getQueueForBeans();

setHistoForActorWaitingTime(beanFeastModel.getHistoForTruckWaitingTime());

}

public boolean isFull(){

return loaded == capacity;

}

public void initCondition(){

isBodyFull = ()-> {

return isFull();

};

isTracktorsReady = ()->{

return queuForBeans.size() != 0;

};

}

@Override

protected void rule() {

initCondition();

while (getDispatcher().getCurrentTime() <= finishTime) {

queueTrucksOnRoad.addLast(this);

holdForTime(roadRnd.next());

queueTrucksOnRoad.remove(this);

queueForLoading.addLast(this);

try {

waitForCondition(isBodyFull, "погрузка зерна на склади");

} catch (Exception e) {

return;

}

getDispatcher().printToProtocol(getNameForProtocol() +

"поехал на розгрузку");

queueTrucksOnRoad.addLast(this);

holdForTime(roadRnd.next());

queueTrucksOnRoad.remove(this);

getDispatcher().printToProtocol(getNameForProtocol() +

"разгружается");

queueForTracktor.addLast(this);

while (loaded != 0){

try {

waitForCondition(isTracktorsReady, "разгрузка зерна");

} catch (Exception e) {

return;

}

Tracktor tractor = queuForBeans.removeFirst();

holdForTime(unloadRnd.next());

tractor.load();

--loaded;

System.out.println("севалка загружена");

}

queueForTracktor.remove(this);

loaded = 0;

}

}

public void setFinishTime(double finishTime) {

this.finishTime = finishTime;

}

public void addPortion() {

++loaded;

getDispatcher().printToProtocol(

getNameForProtocol() + "- у кузові стало " + loaded);

}

}

### Клас Tracktor

public class Tracktor extends Actor {

private double finishTime;

private QueueForTransactions<Tracktor> queueForBeans;

private boolean ready = false;

private BooleanSupplier isLoaded;

private ChooseRandom feastRnd;

public Tracktor(String name, MainWindow gui, BeanFeastModel beanFeastModel) {

setNameForProtocol(name);

queueForBeans = beanFeastModel.getQueueForBeans();

feastRnd = gui.getChooseRandomTracktor();

finishTime = gui.getChooseDataSimTime().getDouble();

setHistoForActorWaitingTime(beanFeastModel.getHistoForTracktorWaitingTime());

}

public void initCondition(){

isLoaded = ()-> ready;

}

@Override

protected void rule() {

initCondition();

while(getDispatcher().getCurrentTime() <= finishTime){

queueForBeans.addLast(this);

try{

waitForCondition(isLoaded, "ожидание погрузки зерна в севалку");

} catch (DispatcherFinishException e) {

return;

}

getDispatcher().printToProtocol(getNameForProtocol() +

"севалка поехала сеять зерно");

holdForTime(feastRnd.next());

getDispatcher().printToProtocol(getNameForProtocol() +

"севалка ждет погрузки зернa");

ready = false;

}

}

public void setFinishTime(double finishTime) {

this.finishTime = finishTime;

}

public void load(){

ready = true;

}

}

### Клас Sklad

public class Sklad extends Actor {

private double finishTime;

private QueueForTransactions<Truck> queueForLoading;

private BooleanSupplier isTrucksReady;

private ChooseRandom loadRnd;

public Sklad(String name, MainWindow gui, BeanFeastModel beanFeastModel) {

finishTime = gui.getChooseDataSimTime().getDouble();

queueForLoading = beanFeastModel.getQueuForLoading();

loadRnd = gui.getChooseRandomLoad();

setNameForProtocol(name);

setHistoForActorWaitingTime(beanFeastModel.getHistoForSkladWaitingTime());

}

@Override

protected void rule() {

initCondition();

while(getDispatcher().getCurrentTime() <= finishTime){

try {

waitForCondition(isTrucksReady, "мае бути вантаживка");

} catch (DispatcherFinishException e) {

return;

}

Truck truck = queueForLoading.removeFirst();

while (!truck.isFull()){

holdForTime(loadRnd.next());

truck.addPortion();

}

System.out.println("gruzovik pogrujen");

}

}

private void initCondition() {

isTrucksReady = ()-> queueForLoading.size() != 0;

}

public void setFinishTime(double finishTime) {

this.finishTime = finishTime;

}

}

## 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.

**СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ**

1. РГР з оделювання. 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: ЧНТУ, 2014. – 79 с.
2. Імітаційне моделювання паралельних процесів. Методичні вказівки до лабораторного практикуму та самостійної роботи з дисципліни «Моделювання» для студентів напряму підготовки 6.050201 – „Комп'ютерна інженерія”. Частина друга. /Укл.: Бивойно П.Г.  - Чернігів: ЧНТУ, 2014. - 50 с.