УДК 004.891

ІНТЕЛЕКТУАЛЬНА ІНФОРМАЦІЙНА СИСТЕМА «ПІДБОРУ МОНОБЛОКУ»

Супрун М.В. *e-mail*: *mrsolomka2447@gmail.com*
Холодняк Ю.В., к.т.н., доц. e-mail: *yuliya.kholodnyak@tsatu.edu.ua*
Таврійський державний агротехнологічний університет імені Дмитра Моторного

Актуальність та постановка проблеми. В останні роки в усьому світі спостерігається все зростаючий інтерес до систем, які в автоматизованому виді видають товар клієнту за набором ознак та характеристик. Власноруч проводити обробку такої кількості інформації неможливо. Тому виникає необхідність в створенні комп'ютерних засобів автоматизації обробки, пошуку, збереження, систематизації даних, при цьому здатних аналізувати дані, пристосовуватися до змін інформації, робити власні висновки на основі аналізу.

Метою статті є розробка інтелектуальної інформаційної системи, яка дасть можливість враховувати побажання користувача по вибору моноблоку.

Основні матеріали дослідження. Інтелектуальна система (ІС) – це технічна або програмна система, яка забезпечує розв'язування неформалізованих задач користувача в деякій предметній галузі та організовує його взаємодію з комп'ютером у звичних поняттях, термінах, образах, що вважалися творчими, які традиційно належать конкретним предметним областям, інформація зберігаються в пам'яті інтелектуальної системи. Структура інтелектуальної системи включає три основні блоки - базу знань, механізм виведення рішень і інтелектуальний інтерфейс. Властивість ІС виконувати творчі завдання називається штучним інтелектом.

Комунікаційні здібності ІС характеризують спосіб взаємодії кінцевого користувача з системою – тобто можливість формулювання довільного запиту в діалозі системи. Складні, погано формалізовані завдання – завдання, що погано формалізуються та вимагають побудови оригінального, нестандартного алгоритму рішення, залежно від конкретної ситуації, для якої можуть бути характерні невизначеність і динамічність.

Оскільки в основі інтелектуальних систем, що створюються як додаток теорії несилової взаємодії, лежить обробка різних дій з виробленням адекватною цим діям реакції, то такі системи отримали назву рефлекторних (по аналогії з поведінкою об'єктів живої природи). За критерієм ступеня інтелектуалізації, який у першому наближенні може характеризуватися обсягом інформації, що обробляється, інтелектуальні системи можна поділити на:

- системи перебору варіантів рішень згідно встановленої пріоритетності для наперед змодельованих ситуацій;
- ІС, які приймають рішення за детермінованими вирішальними правилами без навчання;
- інтелектуальні системи, що реалізують алгоритми компараторного розпізнавання за еталонами;
- експертні системи, що з метою прийняття ефективних рішень маніпулюють спеціальними знаннями
- накопиченими фахівцями експертами у конкретній предметній сфері знань;

- інтелектуальні системи, що здатні самонавчатися;
- знання-орієнтовані ІС, що здатні утворювати та використовувати базу знань.

Інтелектуальні системи, що здатні самонавчатися, можна поділити на такі основні класи:

1. ІС, що розв'язують задачу розпізнавання образів за апріорно класифікованою навчальною матрицею (навчання з "учителем").

2. ІС, що реалізують алгоритми факторного кластер-аналізу.

3. ІС, що реалізують алгоритми кластер-аналізу при незмінному словнику ознак і за апріорно некласифікованими навчальними матрицями, тобто за умови неповної апріорної інформації про функціональний стан системи (навчання без "учителя").

4. ІС, які реалізують алгоритми автоматичної класифікації за апріорно некласифікованими навчальними матрицями та здатні оптимізувати параметри словника ознак розпізнавання.

5. Відмово стійкі ІС, що здатні самостійно діагностувати свій функціональний стан і відновлювати свою функціональну спроможність при виникненні відмов.

6. Адаптивні ІС, що здійснюють класифікаційне самонастроювання та самоорганізацію.

7. ІС, що вирішують проблему шалювання, яка полягає у побудові для шкал з різними мірами виміру зведеної шкали, координати якої можуть бути обернено відображені на відповідні вихідні шкали.

8. Сенсорні ІС, що моделюють чуттєві функції людини та базуються на "образному" комп'ютері, наділеному властивостями "технічного зору", усномовного розпізнавання, розпізнавання пахощів тощо.

9. Гібридні ІС, які поєднують різні алгоритми та методи автоматичної класифікації.

До знанню-орієнтованих інтелектуальних систем відносять:

1. Системи, що ґрунтуються на інструктивних знаннях (rulebased reasoning).

2. Системи, що ґрунтуються на автоматичному доведенні теорем (automatic theorem-proving techniques).

3. Системи, що ґрунтуються на автоматичному породженні гіпотез (automatic hypothesizing).

4. Системи, що ґрунтуються на доведенні за аналогією (analogical reasoning).

5. Об'єктно-орієнтовані інтелектуальні системи (object-oriented intelligent systems).

6. Об'єктно-логічні інтелектуальні системи, що поєднують окремі переваги об'єктно-орієнтованих систем з системами автоматичного доведення теорем і використовують об'єктно-логічні мови, фреймові логіки (F-logics), логіки транзакції (transaction logics) та інше. Зрозуміло, що наведена класифікація не є досить повною, оскільки відбувається неперервне розширення номенклатури ІС як за призначенням, так і за принципами функціонування.

Одна з головних переваг інтелектуальної інформаційної системи, у тому що вона не потребує великих потужностей для її запуску та користування, також займає не велику кількість місця та може бути портативною, також може бути відкритою з будь-якого ПК, ноутбуку або моноблоку.

Дана інтелектуальна система має наступні переваги:

- простий інтерфейс;
- малі мінімальні характеристики для запуску.

- великий товарний асортимент;
- велика кількість параметрів для підбору;

Користувач має можливість пошуку товару за назвою через поле пошуку, або через параметри моделей, такі як:

- ціна;
- об’єм вбудованої пам’яті;
- об’єм оперативної пам’яті;
- розширення дисплею;
- наявність дискретної відеокарти;
- наявність веб камери.

PyQt — це бібліотека Python для створення програм із графічним інтерфейсом за допомогою інструментарію Qt. PyQt є вільним програмним забезпеченням (за ліцензією GPL) і розробляється з 1999 року. Остання версія PyQt6 – на основі Qt 6 – випущена у 2021 році, і бібліотека продовжує оновлюватись. На основі знань бібліотеки PyQt також можна використовувати PySide2, PySide6, PyQt6 і PyQt5.

Сьогодні використовуються дві основні версії: PyQt5 на основі Qt5 та PyQt6 на основі Qt6. Обидві майже повністю сумісні, за винятком імпорту та відсутності підтримки деяких сучасних модулів з Qt6. У PyQt6 вносяться зміни у роботу просторів імен та прапорів, але ними легко управляти.

Після підключення потрібних бібліотек в нашому випадку будемо використовувати PySide2.

Створюється вікно, потім додаються елементи з назвами блоків та чекбокси, розташовуються на площині вікна щоб не накладались один на одного та щоб не були в одній кучі, так само додаються і елементи з виводом інформації результатів.

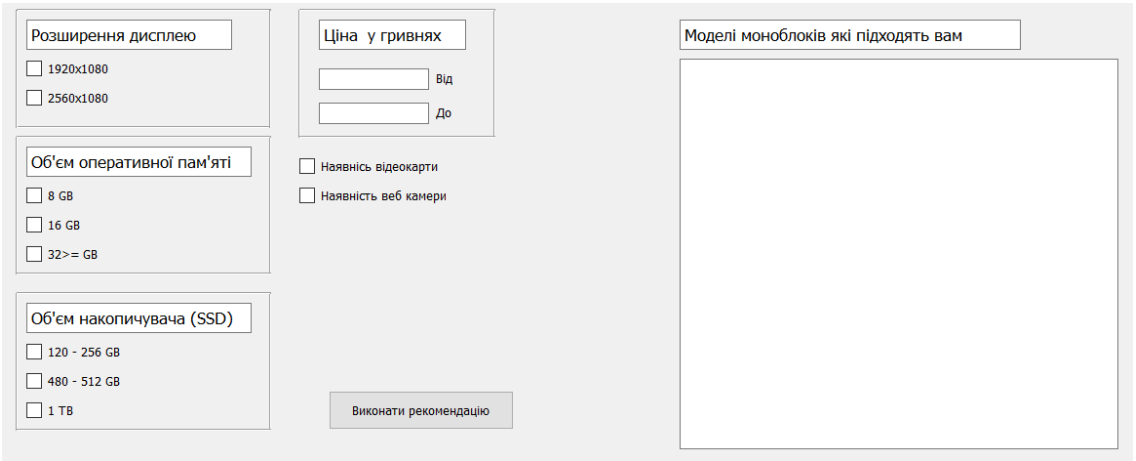

Рисунок 1. Тіло інтерфейсу

Як ми бачило на рис. 1 у вигляді блоків оформлено вибір чекбоксів з характеристиками моноблоків, також у середині блока підписано характеристики та що саме у ньому знаходяться, по інший бік вікна розташований вивід інформації.

На рис. 2 ми бачимо блок програми з розширенням дисплею та наявністю у середені нього чекбоксу з вибором розширення дисплею, також поле яке вказує назву блоку та лінії які візуально відділяють блок від інших елементів.

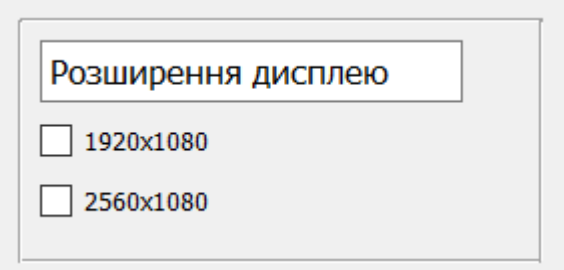

Рисунок 2. Блок “Розширення дисплею”

На рисунку 3 показано блок з обємом оперативної памьяті, у ньому містится чекбокси з вибором кількості оперативної памьяті, а саме 8, 16, 32 та більше гігабайти оперативної памьяті, також розміщене поле з назвою блоку та лінії які виділяють поле тим самим візуально видіяють поле та показують межі блоку.

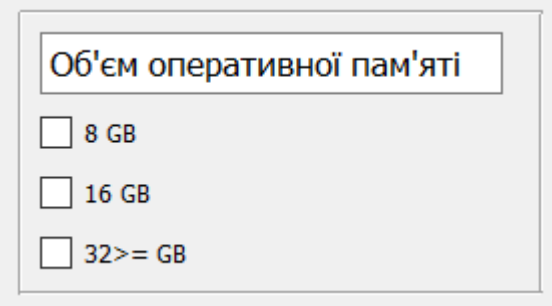

Рисунок 3. Блок “Об’єм оперативної пам’яті”

На рисунку 4 показано блок “Об'єм накопичувача (SSD)”, у ньому містится три чекбокси з вибору об'єму накопичувача який буде встановлений в моноблок, 120-256 Гб, 480-512 Гб, 1Тб, також у блоці є поле яке вказуе назву блоку та що в нбому містиця, лінії яки візуально виділяють блок від инших елементів.

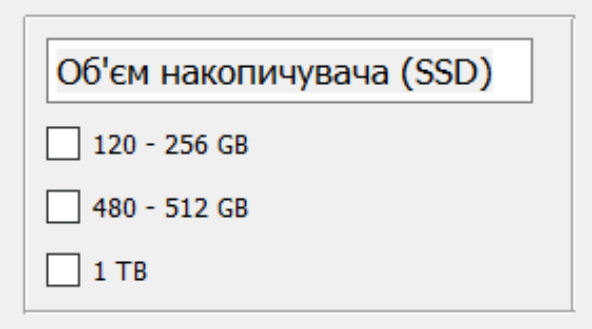

Рисунок 4. Блок “Об’эм накопичувача (SSD)”

На рисунку 5 показано блок “Ціна у гривнях”, у ньому містится, два поля для вводу інформації у вигляді максимальної ціни та мінімальної ціни на яку розраховуе користувач, розміщені слова “Від”, “До”, відповідно до поля у яке повинно вноситися число, також назва блоку та линії які візуально розділяють блок від інших елементів.

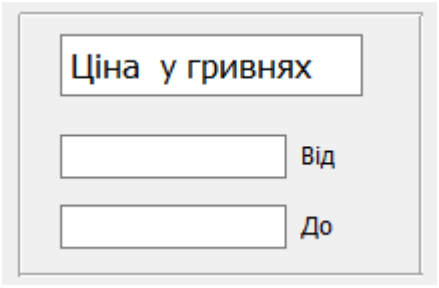

Рисунок 5. Блок “Ціна у гривнях”

На рисунку 6 наведено елемнти які не входять до блоку, це чекбокси які відповідають за характеристики наявності відеокардеокарти та вебкамера.

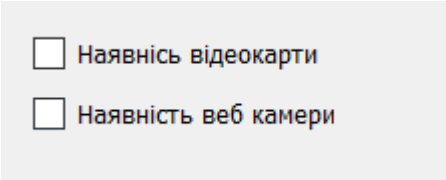

Рисунок 6. Елементи поза блоками

На рисунку 7 наведено блок з полем виведення інформації та полем яке виводить інформацію про поле яке знаходится під ним.

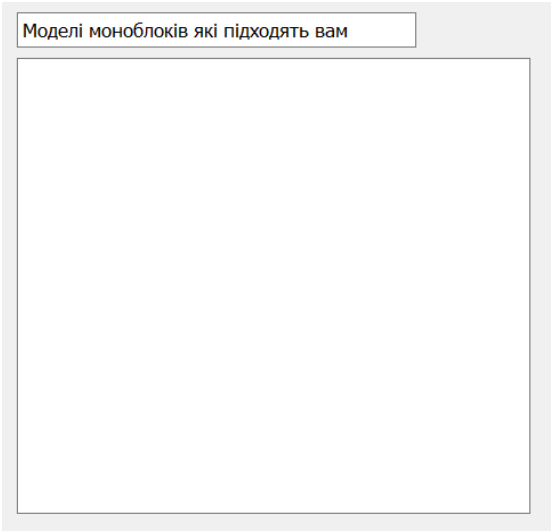

Рисунок 7. Елементи поза блоками

Для розробки клієнтської частини була задіяна мова програмування Python при використанні вбудованих бібліотек та встановленої бібліотеки PySide2 для створення графічних інтерфейсів.

Для отримання інформації від користувача було прийнято рішення використовувати чекбокси та рядок введення для визначення цінового діапазону.

Приклад коду з додавання елементу чекбокс:

```
        self.checkBox = QtWidgets.QCheckBox(self.centralwidget)
        self.checkBox.setGeometry(QtCore.QRect(310, 180, 151, 20))
        self.checkBox.setObjectName("checkBox")
self.checkBox.setText("Наявність відеокарти")
        self.checkBox.stateChanged.connect(self.clickBox)
```

По цьому прикладу ми додали елемент, потім задали місце розташування та розміри елемента у пікселях, задали назву, останній рядок перевіряє статус чекбокса.

Приклад функції перевірки статусу чекбоксу та зберігання його:

```
def clickBox(self, x):
    if x == QtCore.Qt.Checked:
        self.x_1 = 1
    else:
self.x_1 = 0
```

Функція “clickBox” перевіряє стан чекбоксу та якщо чекбокс ввімкнутий то значення змінної “x_1” присвоюється одиниця, у іншому випадку нуль.

Опис кожного блока та блока з виводом результату рекомендації для користувача.

Приклад додавання тексту з описом блоків:

```
Self.textBrowser = QtWidgets.QTextBrowser(self.centralwidget)
self.textBrowser.setGeometry(QtCore.QRect(30, 40, 211, 31))
self.textBrowser.setObjectName("textBrowser")
self.textBrowser.setHtml("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\" \"http://www.w3.org/TR/REC-html40/strict.dtd\">\n"" <html> <head> <meta name=\"qrichtext\" content=\"1\" /><style type=\"text/css\">\n" "p, li { white-space: pre-wrap; }\n" "</style></head><body style=\" font-family:\'MS Shell Dlg 2\'; font-size:7.8pt; font-weight:400;font-style:normal;\">\n" "<p style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-size:10pt;\">Розширення дисплею </span></p></body></html>")
```

Тільки для цього елемента застосовується Html, ця мова розмітки дає змогу втілити майже все на що здатна Html та Css, нам вона дала можливість додати опис до блоку саме таким методом.

Приклад додавання звичайного тексту та розміщення його на вікні:

```
self.labelFrom = QtWidgets.QLabel(self.centralwidget)
self.labelFrom.setGeometry(QtCore.QRect(450, 90, 80, 20))
self.labelFrom.setText("Від")
```

У тій самій функції у якій знаходяться наші елементи які ми додали до інтерфейсу знаходиться й додавання слів “Від” та “До” у блоку з вибором ціни, у прикладі ми бачимо додавання елементу “лейбл” та задання йому розташування та розміри, потім вказуємо те що ми хочемо бачити на місці розташування елементу, аналогічним чином створюється й другий “лейбл” з словом “До”.

Приклад створення кнопки та розташування її у інтерфейсі:

```
self.pushButton = QtWidgets.QPushButton(self.centralwidget)
self.pushButton.setGeometry(QtCore.QRect(340, 420, 190, 40))
self.pushButton.setObjectName("pushButton")
self.pushButton.clicked.connect(self.magic)
self.pushButton.setText("Виконати рекомендацію")
```

На прикладі ми можемо бачити дуже схожі команди на ті що приведені у прикладах вище, тут також задіяна віддача у вигляді ініціації виклику функції “magic”, вона починає обробку узятих даних та у конечному результаті виводить на поле справа рекомендовані моноблоки.

Системне тестування тестує інтегровану систему для перевірки відповідності всім вимогам. Перевірка повноти та правильності документації користувача є важливою частиною системного тестування. Всі тестові комбінації

повинні розроблятися тільки з використанням документації користувача. Однією з головних цілей тестування є перевірка відповідності працездатності інформаційної системи в цілому або її окремих модулів очікуванням користувача.

Тестування включає в себе наступні етапи:

- функціонального тестування – виявлення невідповідностей між реальною поведінкою реалізованих функцій і очікуваною поведінкою відповідно до специфікації і вимог.

Залежно від мети, функціональне тестування може проводитися:

- на основі функціональних вимог, зазначених у специфікації вимог. При цьому для тестування створюються тестові випадки (test cases), створення яких враховує пріоритетність функцій ПЗ, які необхідно покрити тестами. Таким чином ми можемо переконатися в тому, що всі функції продукту, що розробляється, працюють коректно при різних типах вхідних даних, їх комбінацій, кількості і т.д.

- на основі бізнес-процесів, які має забезпечити додаток. У цьому випадку, нас цікавить не так працездатність окремих функцій ПЗ, як коректність виконуваних операцій, з точки зору сценаріїв використання системи. Таким чином, тестування в даному випадку буде гуртуватися на варіантах використання системи (use cases).

- тестування продуктивності, це комплекс типів тестування, метою якого є визначення працездатності, стабільності, споживання ресурсів та інших атрибутів якості додатка в умовах різних сценаріїв використання та навантажень. Тестування продуктивності дозволяє знаходити можливі вразливості та недоліки в системі з метою запобігання їх негативному впливу на роботу програми в умовах використання. Необхідні параметри роботи системи в певному середовищі можна тестувати за допомогою:

- визначення робочої кількості користувачів додатку.
- вимірювання часу виконання різних операцій системи.
- визначення продуктивності додатка при різних рівнях навантаження.
- визначення допустимих меж продуктивності програми при різних рівнях навантаження.

Тестування програмного забезпечення — процес перевірки відповідності заявлених до продукту вимог і реально реалізованої функціональності, який здійснюють шляхом спостереження за його роботою в штучно створених ситуаціях і на обмеженому наборі тестів, обраних певним чином.

Можуть оцінювати:

- відповідність вимогам, якими керувалися проектувальники та розробники,
- правильність відповіді для всіх можливих вхідних даних,
- виконання функцій за прийнятний час,
- практичність,
- сумісність із програмним забезпеченням та операційними системами,
- відповідність задачам замовника.

- компонентне тестування перевіряє функціональність і шукає дефекти в частинах програми, які доступні і можуть бути протестовані окремо (модулі програми, об’єкти, функції і т.д.). Зазвичай компонентне (модульне) тестування проводиться викликаючи код, який необхідно перевірити чи за підтримки середовищ розробки, таких як фреймовки (каркаси) для модульного тестування або інструменти для дебагу. Всі знайдені дефекти, як правило виправляються в коді без формального їх опису в системі багів (Bug Tracking System). Один з найбільш ефективних підходів до компонентного (модульного) тестування – це

підготовка автоматизованих тестів до початку основного кодування ПЗ. Це називається розробка від тестування (test-driven development) або підхід тестування спочатку (test first approach). При цьому підході створюються і інтегруються невеликі шматки коду, навпроти яких запускаються тести, написані до початку кодування. Розробка ведеться до тих пір поки всі тести не будуть успішними.

- приймальне тестування проводиться з метою: визначення чи задовольняє система приймальні критерії там винесення рішення замовником або іншою уповноваженою особою приймається програма чи ні. Приймальне тестування виконується відповідно до Плану приймальних Робіт. Рішення про проведення приймального тестування приймається, коли: продукт досяг необхідного рівня якості та замовник ознайомлений з Планом приймальних Робіт (Product Acceptance Plan) або іншим документом, де описаний набір дій, пов'язаних з проведенням приймального тестування, дата проведення, відповідальні і т.д.

В процесі тестування системи були виявлені наступні результати:

- сторінки системи відповідають макету на всіх пристроях;
- контент відображається коректно;
- обробники подій працюють коректно;
- оформлення системи працює без помилок;
- система працює з БД без помилок;
- процес підбору працює коректно.

Після проведення усіх тестувань ІІС пройшла всі етапи верефікації та відповідає усім умовам успішного проходження верефікації.

Висновок. В результаті проведеної роботи було створено інтелектуальну інформаційну системи «Підбір моноблоку» відповідно до бажань клієнта та було виконано розробку технічного завдання, в якому наведена мета, параметри системи її призначення, вимоги до створення цієї системи. Був описаний користувацький інтерфейс системи, а саме те яким чином створювались об'єкти у вікні, за допомогою яких команд та якого коду вони виконують свою роботу, також описана взаємодія та взяття інформації від користувача та вивід цієї інформації у вікні, та дублюванні її у вигляді текстового документу який створюється у директорії в якої і розташовується інтелектуальна інформаційна система. Було проведене тестування яке складалося з декількох етапів: функціонального тестування, тестування продуктивності, тестування програмного забезпечення, компонентне тестування, приймальне тестування. Після проведення усіх тестувань не було виявлено багів, верифікація пройшла на всіх етапах тестування.

Список використаних джерел:

1. Drabick R. Growth of maturity in the testing process. International Software Testing Institute 1999. http://www.softtest.org/ articles/rdrabick3.htm.

2. Gelperin D., Hetzel B. The Growth of Software Testing. Commum. ACM. 1988. V. 31, N 6, P. 687 – 695..

3. IOM. To Err is Human: Building a Safer Health System: Institute of Medicine (IOM); 1999.

4. Wang, S., et al. A Cost-Benefit Analysis of Electronic Medical Records in Primary Care. The American Journal of Medicine. 2003. Vol. 114. P. 397–403.