ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД УКООПСПІЛКИ
«ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ»

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ БІЗНЕСУ ТА СУЧАСНИХ ТЕХНОЛОГІЙ

ФОРМА НАВЧАННЯ ДЕННА
КАФЕДРА МАТЕМАТИЧНОГО МОДЕЛЮВАННЯ ТА СОЦІАЛЬНОЇ ІНФОРМАТИКИ

Допускається до захисту

Завідувач кафедри _____О.О. Ємець
(підпис)

«_____»_____2020 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА ДО ДИПЛОМНОЇ РОБОТИ

на тему

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТРЕНАЖЕРУ З ТЕМИ «АЛГОРИТМІЧНО НЕРОЗВ'ЯЗНІ ПРОБЛЕМИ» ДИСТАНЦІЙНОГО НАВЧАЛЬНОГО КУРСУ «ТЕОРІЯ ПРОГРАМУВАННЯ»

зі спеціальності 122 «Комп'ютерні науки»

Виконавець роботи Товстоножко Олександр Анатолійович
_____ «___»_____2020р.
(підпис)
Науковий керівник к.ф.-м.н., доц. Черненко Оксана Олексіївна
_____ «___»_____2020р.
(підпис)

ПОЛТАВА 2020р.

ЗМІСТ

ВСТУП

Одним з головних напрямків процесу інформатизації сучасного суспільства стає інформатизація освіти, що забезпечує широке впровадження в практику психолого-педагогічних розробок, спрямованих на інтенсивність процесу навчання, реалізацію ідей розвиваючого навчання, вдосконалення форм і методів організації навчального процесу. Застосування в освіті комп'ютерів і інформаційних технологій робить істотний вплив на зміст, методи і організацію навчального процесу з різних дисциплін.

У зв'язку з цим, в даний час відбувається інтенсивна розробка навчальних, ігрових і контролюючих програм, які допомагають суттєво зекономити час викладача, розвинути творчі здібності учня шляхом створення сприятливого середовища, тренуватися у вирішенні завдань певного класу і отримувати адекватні оцінки своїх знань.

Метою роботи є розробка тренажеру з теми «Алгоритмічно нерозв’язні проблеми» дистанційного навчального курсу «Теорія програмування».

Об’єктом розробки є процес дистанційного навчання математичним дисциплінам.

Предмет розробки – тренажер для огляду алгоритмічно нерозв’язних проблем на мові програмування Java.

Головне завдання – створити програмну реалізацію тренажеру з теми «Алгоритмічно нерозв’язні проблеми».

Перелік використаних методів:

- вивчення теоретичного матеріалу по різновидам тренажерів,
- вивчення теоретичного матеріалу з обраної теми,
- розробка тренажера,
- аналіз отриманих даних.

Робота складається з чотирьох розділів. У першому розділі розглянуто постановку задачі. У другому розділі описано значення та приклади алгоритмічно нерозв’язних проблем, навчальні програми для різних дисциплін. У третьому розділі розглянуто основні означення з теми «Алгоритмічно нерозв’язні проблеми», описано алгоритм роботи тренажеру, розробку блок-схеми алгоритму. У четвертому розділі – описано обґрунтування вибору програмних засобів, процес програмної реалізації, інструкцію по використанню тренажеру.

Обсяг пояснювальної записки: 64 стор., в т.ч. основна частина - 43 стор., джерела - 11 назв.

1. ПОСТАНОВКА ЗАДАЧІ

Основним завданням роботи є розробка тренажеру з теми «Алгоритмічно нерозв’язні проблеми» дистанційного навчального курсу «Теорія програмування».

Основні завдання роботи:

- описати постановку задачі;
- описати огляд робіт, аналогічних вибраній темі;
- навести основні поняття з теми для використання в тренажері;
- розробити алгоритм роботи тренажеру та скласти його блок-схему;
- описати мову програмування та технології, що були використані при розробці програми;
- описати процес реалізації основних етапів створення тренажеру;
- перевірити роботу тренажеру з наведенням прикладів;
- описати необхідну користувачу інструкцію.

Для можливості використовувати тренажер для вивчення дистанційного курсу «Теорія програмування» іноземними студентами слід розробити можливість зміни мови тренажеру. Інший спосіб – запуск тренажеру лише після вибору мови.

На стартовому вікні повинна виводитись тема тренажеру та інформація про розробника. Якщо передбачено декілька блоків із питаннями, то можна надати можливість перейти відразу до вибраного.

На кожному кроці алгоритму має виводитися завдання та перевірятися відповідь. При помилці повинно відображатися повідомлення про це, як варіант – вказати правильну відповідь або пояснення розв’язку.

Після проходження тренажеру слід вивести результат.

2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1. Значення та приклади алгоритмічно нерозв’язних проблем

За час свого існування людство придумало безліч алгоритмів для вирішення різноманітних практичних і наукових проблем. Поставимо запитання: а чи існують такі проблеми, для яких неможливо придумати алгоритми їх вирішення?

Твердження про існування алгоритмічно нерозв'язних проблем є дуже переконливим, адже - ми констатуємо, що ми не знаємо відповідного алгоритму, не тільки зараз а й ми не можемо принципово ніколи його знайти. Виявляється, що існують такі класи задач, для розв’язання яких немає і не може бути єдиного універсального прийому. Проблеми розв’язання такого роду задач називають алгоритмічно нерозв'язними проблемами. Однак алгоритмічна нерозв'язність проблеми розв'язання задач того чи іншого класу зовсім не означає неможливість розв’язання будь-якої конкретної задачі з цього класу. Мова йде про неможливість вирішення всіх завдань даного класу одним і тим самим прийомом. *Алгоритмічна нерозв'язність* - найважливіша властивість деяких класів коректно поставлених завдань, що допускають застосування алгоритмів, яка полягає в тому, що задача кожного з цих класів у принципі не має загального, універсального алгоритму розв’язання, що об'єднує цей клас [2].

Першою фундаментальною теоретичною роботою, пов'язаною з доказом алгоритмічної нерозв'язності, була робота Курта Геделя – його відома теорема про неповноту символічних логік.

Зусиллями різних дослідників список алгоритмічно нерозв'язних проблем був значно розширений. Сьогодні прийнято при доказі алгоритмічної нерозв'язності деякої задачі зводити її до класичної задачі – «задачі зупину» [3, 4].

Алгоритмічна нерозв'язність не є проблемою теорії алгоритмів або невдачею, вона є науковим фактом. Популярність досліджень алгоритмічної нерозв'язності не поступається будь-яким іншим дослідженням у теорії алгоритмів і потребує іноді значних зусиль. Наприклад, доведення алгоритмічної нерозв'язності 10-ї проблеми Гільберта, одержане Ю. В. Матіясевичем, відносять до видатних наукових досягнень XX століття.

Знання можливої алгоритмічної нерозв’язності має в галузі комп’ютерних технологій таке саме значення, як для фізика знання про неможливість створення вічного двигуна. Припустимо, що до Білла Гейтса приходить винахідник з пропозицією надати йому грант (грошові інвестиції) для написання програми, яка приймає на вхід будь-яку програму і видає відповідь, потрапить ця програма в нескінченний цикл чи ні за будь-яких вхідних даних. Б. Гейтс йому, звісно, відповідь, оскільки знає, що ця задача є алгоритмічно нерозв'язною [5].

Проблеми, що стосуються абстрактних машин [6]:

- Проблема зупинки;
- Проблема самозастосування;
- Busy beaver;
- Будь-яка проблема, сформульована в теоремі Райса;
- Визначити, чи досягне коли-небудь задана вихідна конфігурація в грі «Життя» заданої кінцевої конфігурації.

Проблеми, що стосуються матриць [6]:

- Проблема вмираючої матриці: для даної кінцевої множини квадратних матриць $n \times n$ визначити, чи існує добуток всіх або деяких з цих матриць (можливо, з повтореннями) в якому-небудь порядку, що дає нульову матрицю. Проблема нерозв'язна навіть для n=3 (можливість розв'язання для n = 2 є відкритим питанням)

- Проблема одиничної матриці: для даної кінцевої множини квадратних матриць $n \times n$ визначити, чи існує добуток всіх або деяких з цих матриць (можливо, з повтореннями) в якому-небудь порядку, що дає одиничну матрицю. проблема нерозв'язна для цілочисельних матриць починаючи з n=4 та розв'язна для n = 2 (можливість розв'язання для n = 3 є відкритим питанням). Проблема еквівалентна питанню, чи є матрична півгрупа групою.
- Проблема вільності матричної напівгрупи алгоритмічно нерозв'язна для цілочисельних матриць починаючи з n = 3 і відкрита для n = 2.

Інші проблеми [6]:

- Задача розв'язності
- Виводимість формули в арифметиці Пеано
- Проблема збіжності Поста
- Обчислення колмогорівської складності довільного рядка
- Ідеальний архіватор, що створює для будь-якого вхідного файлу програму найменшого можливого розміру, що друкує цей файл
- Ідеальний оптимізувальний за розміром компілятор
- Десята проблема Гільберта
- Визначити, чи можна замостити площину даними набором плиток Ванга
- Визначити, чи можна замостити площину даними набором поліміно.
- Проблема уніфікації другого і вищого порядків
- Проблема виводу типів в моделі Хіндлі — Мілнера з rank-N поліморфізмом

2.2. Навчальні програми для різних дисциплін

Інформаційна технологія навчання є новою методичною системою, що дозволяє розглядати учня не як об'єкт, а як суб'єкт навчання, а комп'ютер - як засіб навчання. Той, якого навчають переходить в нову категорію тому, що за формою комп'ютерне навчання є індивідуальним, самостійним, але здійснюється за загальною методикою, реалізованої в навчальній програмі.

Навчальні програми бувають декількох типів: інформаційні, довідкові, контролюючі, комбіновані. Одні навчальні програми здатні контролювати знання учнів, інші містять в собі елементи «навчального тренажера», треті допомагають оволодінню новим навчальним матеріалом, четверті призначені для того, щоб стимулювати інтерес до досліджуваного предмета.

Існує кілька основних принципів побудови навчальних програм:

Принцип доступності при комп'ютерному навчанні переходить від принципу загальної доступності, для певної вікової групи учнів або для деякого усередненого учня даного віку, в принцип індивідуальної доступності і розглядається як можливість досягнення мети навчання. Навчальний матеріал, реалізований в комп'ютерному навчанні, передбачає наявність розгалужень, різних шляхів і швидкостей проходження навчального курсу, надання допомоги у вигляді пояснень, підказок, додаткових вказівок і завдань, постійно контролює і підтримує на необхідному рівні мотивацію учня.

Принцип наочності в комп'ютерному навчанні дозволяє побачити те, що не завжди можливо в реальному житті навіть за допомогою самих чутливих і точних приладів.

Принцип систематичності і послідовності пов'язаний як з організацією навчального матеріалу, так і з системою дій учня по його

засвоєнню. Комп'ютерне навчання характеризується послідовністю специфічних дій, частина яких притаманна навчання в будь-яких формах, а частина - тільки комп'ютерного. Такими діями, наприклад, є сприйняття інформації з екрана дисплея, робота в знакових моделях, введення відповіді з клавіатури. Для забезпечення принципу послідовності учню на початку сеансу комп'ютерного навчання корисно дати орієнтовну основу дії, сформулювати мету навчання. Поняття послідовності отримало свій сенс в інформаційних технологіях навчання, під послідовністю розуміється черговість видачі навчальних фрагментів навчальною програмою.

Перед показом кожного вікна програми необхідно подбати про вид кожного рядка, кожного слова і навіть кожного символу тексту. Головним тут є ясність повідомлення, що видається. Повідомлення в цілому можна охарактеризувати чотирма показниками читабельності: типом стилю, довжиною рядка, вирівнюванням тексту, точками переривання. Стиль включає виразність зображення, підкреслення, виділення мерехтінням і іншими засобами важливих моментів змісту, розміри тексту, всілякі способи чергування, курсив і колір. Очевидно, що розмір тексту повинен бути невеликим, принаймні таким, щоб можна було розрізняти подібні малі літери і знаки.

При читанні тексту засвоюється сенсорна, синтаксична, семантична і прагматична інформації. Той, хто читає використовує різні процеси подання та обробки знань для засвоєння інформації, що міститься в тексті.

У навчальних програмах існує три основних способи організації кадру.

При першому способі кадр розділяється на дві, не обов'язково рівні, частини. Одна частина може бути використана для введення даних з клавіатури, а інша - для виведення команд, підказок.

В іншому випадку - вікна екрану можуть мати різний розмір і зміст в різні моменти навчання. Другий спосіб організації простору кадру означає, що немає зарезервованих ділянок кадру, які містять деякий бланк для

видачі інформації. Необхідна інформація видається в міру необхідності, а відведена для цього частину кадру може використовуватися для інших цілей.

Третій спосіб - використання вкладених вікон. Таке вікно з'являється в певному місці екрану на вимогу, воно зникає або коли на екрані зникає дана вимога, або з'являється нове. Також необхідно мати на кадрі ділянку для введення відповіді учня на питання системи. Бажано, щоб і питання задавалися в певному місці кадру. Постійно в одному і тому ж місці повинна з'являтися (або постійно бути присутнім) орієнтовна інформація. Необхідно так само мати список керуючих впливів, таких як переривання навчання, очищення екрану.

Навчальні програми використовуються для вирішення широкого діапазону педагогічних задач [7].

3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Алгоритмічно нерозв’язні проблеми

Існування алгоритмічно нерозв'язних проблем випливає вже з теореми Геделя про неповноту формальних систем. Справа у тому, що існує тісний зв'язок між алгоритмами і вирахуваннями. Власне кажучи, і алгоритми, і вирахування – це сукупності чітких, однозначно заданих скінченних інструкцій, що описують якісь дії із символічними об'єктами. Однак у випадку алгоритму ці інструкції мають характер розпоряджень, що задають однозначний порядок виконання операцій над символічними об'єктами, тоді як у випадку вирахувань – інструкції не визначать черговості їх виконання [3].

Означення 1. Алгоритмічною проблемою називається проблема побудови алгоритму, що володіє тими чи іншими властивостями.

Ми отримаємо вирішення алгоритмічної проблеми, якщо або знайдемо шуканий алгоритм (тобто подано його опис), або доведем, що такого алгоритму не існує.

Однією з найбільш загальних теорем теорії алгоритмів, що пояснює природу багатьох проблем у теорії алгоритмів, є теорема Райса.

Уведемо деякі визначення.

Означення 2. Характеристичною функцією множини A будемо називати функцію

$$\chi_A(x) = \begin{cases} 1, \textit{якщо } x \in A, \\ 0, \textit{якщо } x \notin A. \end{cases}$$

Множина A називається рекурсивною, якщо її характеристична функція рекурсивна.

Означення 3. Нехай Q – деяка властивість одномісних частково-рекурсивних функцій. Властивість Q називається нетривіальною, якщо існують функції, що володіють цією властивістю, так і функції, що не володіють.

Усі властивості, що розглядаються в теорії алгоритмів, зазвичай є нетривіальними. Наприклад, властивості функцій : усюди визначеність, взаємна однозначність, тотожна рівна нулю та інші.

Ураховуючи те, що частково-рекурсивні функції можна задати програмою їх обчислення, виникає питання: чи можливо за програмою визначити, чи має відповідна функція певну нетривіальну властивість?

Відповідно до тез Черча і Тюрінга задача є алгоритмічно розв’язуваною тоді і тільки тоді, коли існує деяка машина Тюрінга M, що розв’язує цю задачу. За поставленим завданням необхідно визначити, чи характерна функції $f_м(x)$, що реалізується машиною M, властивість Q. Функцію $f_м(x)$ будемо розуміти як функцію, що відповідає програмі з номером Геделя *x*.

Теорема Райса. Якою б не була нетривіальна властивість Q одномісних частково-рекурсивних функцій, задача розпізнання цієї властивості алгоритмічно нерозв’язна, тобто не існує машини Тюрінга, що розв’язує цю задачу.

Наслідки з теореми Райса. Неможливо за допомогою універсального алгоритму перевірити всюди визначеність функції $f_м(x)$; не існує алгоритму перевірки того, що програма обчислить нульову функцію; питання про те, обчислюють чи ні дві програми одну й ту саму функцію, масово нерозв’язна; не існує алгоритму, що визначає за текстом програми, чи буде ця програма обчислювати деяку конкретну обчислювальну функцію.

Теорема Райса є однією з найбільш загальних теорем теорії алгоритмів, що пояснює природу багатьох проблем нерозв’язності у практиці програмування.

Першою фундаментальною теоретичною працею, пов'язаною з доведенням алгоритмічної нерозв'язності, була робота Курта Геделя – його відома теорема про неповноту символічних логік. Це була строго сформульована математична проблема, для якої не існує розв’язного її

алгоритму. Зусиллями різних дослідників список алгоритмічно нерозв'язних проблем був значно розширений. Сьогодні прийнято під час доведення алгоритмічної нерозв'язності деякої проблеми зводити її до класичної задачі – «задачі зупинки».

Теорема. Не існує алгоритму (машини Тюрінга), що дозволяє за описом довільного алгоритму і його вихідних даних (і алгоритм, і дані задані символами на рядку машини Тюрінга) визначити, зупиняється цей алгоритм на цих даних чи працює нескінченно.

Таким чином, фундаментально алгоритмічна нерозв'язність пов'язана з нескінченністю виконуваних алгоритмом дій, тобто неможливістю передбачити, що для будь-яких вихідних даних розв’язок буде отриманий за кінцеву кількість кроків [4].

Тим не менше, можна спробувати сформулювати причини, що призводять до алгоритмічної нерозв'язності. Ці причини достатньо умовні, оскільки всі вони зводяться до проблеми зупину, однак такий підхід дозволяє більш глибоко зрозуміти природу алгоритмічної нерозв'язності :

а) Відсутність загального методу розв'язання задачі

Проблема 1. *Розподіл дев’яток у запису числа π*

Визначимо функцію f (n) = i, де n - кількість дев'яток поспіль у десятковому записі числа π, а $_i$- номер найлівішої дев'ятки з nдев'яток поспіль: π = 3,141592 ... f (1) = 5.

Завдання полягає в обчисленні функції f (n) для довільно заданого n.

Оскільки число π є ірраціональним і трансцендентним, то ми не знаємо жодної інформації про розподіл дев'яток (так само як і будь-яких інших цифр) у десятковому записі числа.

Обчислення f (n) пов'язане з обчисленням наступних цифр у розкладанні, доти, поки ми не виявимо n дев'яток поспіль, однак у нас немає загального методу обчислення f(n), тому для деяких n обчислення можуть тривати нескінченно - ми навіть не знаємо в принципі (за природою числа π), чи існує розв’язок для всіх n .

Проблема 2. *Десята проблема Гільберта*

Вона полягає у знаходженні універсального методу цілочислового розв’язання довільного алгебраїчного діофантового рівняння. Доведення алгоритмічної нерозв'язності цієї задачі зайняло близько двадцяти років і було завершене Ю. В. Матіясевичем у 1970 році.

Формально мова йде про цілочислове розв’язання рівнянь вигляду

$P(x_1, x_2, \ldots, x_n) = 0$, де P - многочлен із цілими коефіцієнтами й цілими показниками степенів. Доведено, що такого алгоритму не існує, тобто відсутній загальний метод визначення цілих коренів цього рівняння [4].

Проблема 3. *Обчислення досконалих чисел*

Досконалі числа – це числа, які дорівнюють сумі своїх дільників, наприклад:

28 = 1 +2 +4 +7 +14.

Визначимо функцію S(n) = n-ое за рахунком досконале число і сформулюємо задачу обчислення S(n) по довільно заданому n.

Немає загального методу обчислення досконалих чисел, навіть не відома множина досконалих чисел. Тому алгоритм повинен перебирати всі числа поспіль, перевіряючи їх на досконалість.

Відсутність загального методу рішення не дозволяє відповісти на питання про умову зупинення алгоритму. Якщо перевірили M чисел при пошуку n-го досконалого числа – чи означає це, що його взагалі не існує?

б) Інформаційна невизначеність завдання

Проблема 4. *Позиціонування машини Посту на останньому позначеному ящику*

Нехай на стрічці машини Посту задані набори помічених ящиків (кортежів) довільної довжини з довільними відстанями між кортежами і каретка знаходиться у самого лівого поміченого ящика. Задача полягає в установці каретки на самий правий позначений ящик останнього кортежу.

Спроба побудови алгоритму, що вирішує це завдання, призводить до необхідності відповіді на питання – чи немає більше на стрічці кортежів або вони є десь правіше, коли після виявлення кінця кортежу ми зрушили вправо по порожніх ящиках на M позицій і не виявили початок наступного кортежу?

Інформаційна невизначеність задачі полягає у відсутності інформації або про кількість кортежів на стрічці, або про максимальні відстані між кортежами. За наявності такої інформації (при дозволі інформаційної невизначеності) задача стає алгоритмічно розв’язаною.

в) Логічна нерозв'язність

Проблема 5. *Проблема еквівалентності алгоритмів*

За двома довільно заданими алгоритмами (наприклад, для двох машин Тьюринга) визначити, чи будуть вони видавати однакові вихідні результати на будь-яких вхідних даних.

Проблема 6. *Проблема тотальності*

За довільно заданим алгоритмом визначити, чи буде він зупинятися на всіх можливих наборах вхідних даних. Інше формулювання цього завдання – чи є частковий алгоритм P усюди визначеним [3]?

Прикладом алгоритмічної нерозв’язності може бути нерозв’язність “проблеми зупинки”. “Проблема зупинки” – це проблема пошуку універсальної програми, що дозволяє за записом довільної програми (наприклад, функціональної таблиці машини Тюрінга), а також за записом довільних вхідних даних установити, чи зупиниться обчислювальний пристрій, що діє відповідно до даної програми і обробляє вхідні дані, або ж він буде працювати нескінченно довго.

Теорема про "зупинку". Програма називається застосовною до вхідних даних, якщо вона рано чи пізно зупиниться і видасть деякий результат. У протилежному разі говорять, що програма незастосовна до вхідних даних. Теорема про “зупинку” стверджує, що проблема

застосовності довільної програми до довільних вхідних даних алгоритмічно нерозв'язна.

Ця теорема доводиться досить просто. Перший крок полягає у тому, що вводиться поняття самозастосовності програми. Програма називається самозастосовною, якщо вона ефективно переробляє текст, що відповідає її власному запису, у деякий результат за скінченне число кроків. У протилежному разі якщо програма не зупиняється і продовжує працювати нескінченно довго, то вона називається не самозастосовною.

Спочатку доводиться таке твердження: не існує програми, застосовної до всіх несамозастосовних програм і тільки до них. Доведення полягає у вказівці на суперечливість поняття про таку програму. Задамося питанням: чи є дана програм самозастосовною? Якщо вона самозастосовна, то вона несамозастосовна (оскільки застосовна лише до несамозастосовних програм). Якщо ж вона несамозастосовна, то вона самозастосовна (оскільки застосовна до всіх несамозастосовних програм).

Виходячи з цього результату, можна також довести неіснування програми, здатної універсальним способом розпізнавати не самозастосовність довільних програм. Дійсно, якщо така програма існує, то можна побудувати й програму, застосовну до всіх не самозастосовних програм і тільки до них.

Розпізнавання несамозастосовності:

Позначимо буквою B програму, здатну розпізнавати несамозастосовність. Тоді наступна програма буде програмою, застосовною до всіх несамозастосовних програм і тільки до них.

1. Виконати послідовність команд B, перейти до кроку 2.
2. Якщо отримана відповідь “так”, то перейти до кроку 3, у протилежному разі перейти до кроку 4.
3. Закінчити процес.
4. Перейти до кроку 4.

Ця програма зупиняється, якщо розглянута як вхідні дані програма є несамозастосовною, і не зупиняється (зациклюється на кроці 4) у протилежному разі.

Використовуючи даний результат можна також показати, що не існує й програми, що розпізнає універсальним способом самозастосовність (оскільки у протилежному разі можна побудувати алгоритм, що розпізнає несамозастосовність).

І нарешті, можна показати, що алгоритмічно нерозв'язною є проблема розпізнавання застосовності довільної програми до довільних вхідних даних. Припустимо зворотне. Нехай Е – програма, що за заданою довільною програмою і заданим на вході словом розпізнає застосовність даної програми до даного слова. Неважко побудувати програму, що дозволяє установити, чи є задане слово кодом даної програми. Позначимо таку програму буквою Р.

1. Тоді можна побудувати програму Н.
2. Застосувати послідовність команд Р. Перейти до кроку 2.
3. Якщо послідовність команд Р дала відповідь “так”, перейти до кроку 3, інакше – до кроку 4.
4. Виконати послідовність команд Е. Кінець.
5. Перейти до кроку 4.

Висновок:

Програма Н є програмою, що розпізнає самозастосовність довільних програм. Така програма неможлива, а отже, неможлива і програма Е.

Чому існують нерозв'язні проблеми. Проблема – це питання про приналежність ланцюжка мові. Множина мов над будь-яким алфавітом, де більше одного символу, незліченна. Програми, будучи скінченними ланцюжками у скінченному алфавіті, утворять зліченну множину. Іншими словами, проблем нескінченно більше, ніж програм [8].

3.2. Алгоритм роботи тренажеру

На стартовому вікні виводиться назва тренажеру, інформація щодо розробника. Пропонується перейти до перевірки основних означень з теми або до алгоритмічно нерозв'язних проблем.

Основні означення з теми

Крок 1. Виберіть відповідь. Алгоритмічною проблемою називається:

- проблема побудови алгоритму, що володіє тими чи іншими властивостями;
- проблема побудови алгоритму, що не володіє тими чи іншими властивостями.

Якщо вказано неправильну відповідь, то відображається пояснення: «Алгоритмічною проблемою називається проблема побудови алгоритму, що володіє тими чи іншими властивостями». Перехід до наступного кроку.

Крок 2. Виберіть відповідь (можливий вибір декількох варіантів). Ми отримаємо вирішення алгоритмічної проблеми, якщо:

- знайдемо шуканий алгоритм (тобто подано його опис);
- доведем, що такого алгоритму не існує;
- вирішення цієї алгоритмічної проблеми відсутнє.

Якщо вказано неправильну відповідь, то відображається пояснення: «Ми отримаємо вирішення алгоритмічної проблеми, якщо або знайдемо шуканий алгоритм (тобто подано його опис), або доведем, що такого алгоритму не існує». Перехід до наступного кроку.

Крок 3. Виберіть відповідь. Характеристичною функцією множини А будемо називати функцію:

- $\chi_A(x) = \begin{cases} 1, якщо\, x \in A, \\ 0, якщо\, x \notin A. \end{cases}$;
- $\chi_A(x) = \begin{cases} 1, якщо\, x \in A, \\ -1, якщо\, x \notin A. \end{cases}$;

- $\chi_A(x)=\begin{cases}1, якщо\ x \notin A,\\ 0, якщо\ x \in A.\end{cases}$.

Якщо вказано неправильну відповідь, то відображається пояснення: «Характеристичною функцією множини А будемо називати функцію $\chi_A(x)=\begin{cases}1, якщо\ x \in A,\\ 0, якщо\ x \notin A.\end{cases}$». Перехід до наступного кроку.

Крок 4. Виберіть відповідь. Нехай Q – деяка властивість одномісних частково-рекурсивних функцій. Властивість Q називається нетривіальною, якщо:

- існують функції, що володіють цією властивістю;
- існують функції, що не володіють цією властивістю;
- всі відповіді правильні.

Якщо вказано неправильну відповідь, то відображається пояснення: «Властивість Q називається нетривіальною, якщо існують функції, що володіють цією властивістю, так і функції, що не володіють». Перехід до наступного кроку.

Крок 5. Виберіть відповідь. Теорема Райса: Якою б не була нетривіальна властивість Q одномісних частково-рекурсивних функцій, задача розпізнання цієї властивості:

- алгоритмічно нерозв’язна, тобто не існує машини Тюрінга, що розв’язує цю задачу;
- алгоритмічно розв’язна, тобто існує машина Тюрінга, що розв’язує цю задачу;
- всі відповіді правильні.

Якщо вказано неправильну відповідь, то відображається пояснення: «Теорема Райса: Якою б не була нетривіальна властивість Q одномісних частково-рекурсивних функцій, задача розпізнання цієї властивості алгоритмічно нерозв’язна, тобто не існує машини Тюрінга, що розв’язує цю задачу». Перехід до наступного кроку.

Крок 6. Виберіть відповідь (можливий вибір декількох варіантів). Наслідки з теореми Райса:

- неможливо за допомогою універсального алгоритму перевірити всюди визначеність функції $f_м(x)$;
- не існує алгоритму перевірки того, що програма обчислить нульову функцію;
- питання про те, обчислюють чи ні дві програми одну й ту саму функцію, масово нерозв’язна;
- не існує алгоритму, що визначає за текстом програми, чи буде ця програма обчислювати деяку конкретну обчислювальну функцію.

Якщо вказано неправильну відповідь, то відображається пояснення: «Наслідки з теореми Райса: Неможливо за допомогою універсального алгоритму перевірити всюди визначеність функції $f_м(x)$; не існує алгоритму перевірки того, що програма обчислить нульову функцію; питання про те, обчислюють чи ні дві програми одну й ту саму функцію, масово нерозв’язна; не існує алгоритму, що визначає за текстом програми, чи буде ця програма обчислювати деяку конкретну обчислювальну функцію». Перехід до наступного кроку.

Крок 7. Повідомлення про завершення:

Ви пройшли перевірку основних означень з теми! Пропонуємо перейти до огляду алгоритмічно нерозв’язних проблем.

Якщо вибрано перехід, то виводиться умова першої проблеми і завдання. Якщо ні – завершення роботи тренажеру.

Алгоритмічно нерозв’язні проблеми

Крок 1. Проблема 1: Розподіл дев’яток у запису числа π.

Визначимо функцію f (n) = i, де n - кількість дев'яток поспіль у десятковому записі числа π, a_i - номер найлівішої дев'ятки з n дев'яток поспіль: π = 3,141592 ... f (1) = 5.

Завдання полягає в обчисленні функції f (n) для довільно заданого n.

Наведіть доведення:

1. Оскільки число π є ірраціональним і трансцендентним, то ми не знаємо жодної інформації про розподіл дев'яток (так само як і будь-яких інших цифр) у десятковому записі числа.
2. Обчислення f (n) пов'язане з обчисленням наступних цифр у розкладанні, доти, поки ми не виявимо n дев'яток поспіль.
3. Однак у нас немає загального методу обчислення f(n), тому для деяких n обчислення можуть тривати нескінченно.
4. Ми навіть не знаємо в принципі (за природою числа π), чи існує розв’язок для всіх n .

Якщо не вказано правильну послідовність, то відображається відповідне пояснення. Перехід до наступного кроку.

Крок 2. Проблема 2: Десята проблема Гільберта.

Вона полягає у знаходженні універсального методу цілочислового розв’язання довільного алгебраїчного діофантового рівняння.

Наведіть доведення:

1. Формально мова йде про цілочислове розв’язання рівнянь вигляду P (x1, x2,…, xn) = 0, де P - многочлен із цілими коефіцієнтами й цілими показниками степенів.
2. Доведено, що такого алгоритму не існує, тобто відсутній загальний метод визначення цілих коренів цього рівняння.

Якщо не вказано правильну послідовність, то відображається відповідне пояснення. Перехід до наступного кроку.

Крок 3. Проблема 3: Обчислення досконалих чисел.

Досконалі числа – це числа, які дорівнюють сумі своїх дільників, наприклад:

28 = 1 +2 +4 +7 +14.

Визначимо функцію S(n) = n-ое за рахунком досконале число і сформулюємо задачу обчислення S(n) по довільно заданому n.

Наведіть доведення:

1. Немає загального методу обчислення досконалих чисел, навіть не відома множина досконалих чисел.
2. Тому алгоритм повинен перебирати всі числа поспіль, перевіряючи їх на досконалість.
3. Відсутність загального методу рішення не дозволяє відповісти на питання про умову зупинення алгоритму.
4. Якщо перевірили M чисел при пошуку n-го досконалого числа – чи означає це, що його взагалі не існує?

Якщо не вказано правильну послідовність, то відображається відповідне пояснення. Перехід до наступного кроку.

Крок 4. Проблема 4: Позиціонування машини Посту на останньому позначеному ящику.

Нехай на стрічці машини Посту задані набори помічених ящиків (кортежів) довільної довжини з довільними відстанями між кортежами і каретка знаходиться у самого лівого поміченого ящика. Задача полягає в установці каретки на самий правий позначений ящик останнього кортежу.

Наведіть доведення:

1. Спроба побудови алгоритму, що вирішує це завдання, призводить до необхідності відповіді на питання. – чи немає більше на стрічці кортежів або вони є десь правіше, коли після виявлення кінця кортежу ми зрушили вправо по порожніх ящиках на M позицій і не виявили початок наступного кортежу?
2. Інформаційна невизначеність задачі полягає у відсутності інформації або про кількість кортежів на стрічці, або про максимальні відстані між кортежами.
3. За наявності інформації про кількість кортежів на стрічці або про максимальні відстані між кортежами (при дозволі

інформаційної невизначеності) задача стає алгоритмічно розв’язаною.

Якщо не вказано правильну послідовність, то відображається відповідне пояснення. Перехід до наступного кроку.

Крок 5. Проблема 5: Проблема еквівалентності алгоритмів.

За двома довільно заданими алгоритмами (наприклад, для двох машин Тьюринга) визначити:

- чи будуть вони видавати різні вихідні результати на будь-яких вхідних даних;
- чи будуть вони видавати однакові вихідні результати на будь-яких вхідних даних;
- чи будуть вони видавати однакові вихідні результати на певних вхідних даних;
- чи будуть вони взагалі видавати вихідні результати.

Якщо вказано неправильну відповідь, то відображається пояснення: «За двома довільно заданими алгоритмами (наприклад, для двох машин Тьюринга) визначити, чи будуть вони видавати однакові вихідні результати на будь-яких вхідних даних». Перехід до наступного кроку.

Крок 6. Проблема 6: Проблема тотальності.

За довільно заданим алгоритмом визначити, чи буде він зупинятися на всіх можливих наборах вхідних даних.

Інше формулювання цього завдання:

- Чи є частковий алгоритм P визначеним?
- Чи є частковий алгоритм P усюди визначеним?
- Чи є частковий алгоритм P частково визначеним?

Якщо вказано неправильну відповідь, то відображається пояснення: «Інше формулювання цього завдання – чи є частковий алгоритм P усюди визначеним?». Перехід до наступного кроку.

Крок 7. Проблема 7: Теорема про "зупинку".

Програма називається застосовною до вхідних даних, якщо вона рано чи пізно зупиниться і видасть деякий результат. У протилежному разі говорять, що програма незастосовна до вхідних даних. Теорема про “зупинку” стверджує, що проблема застосовності довільної програми до довільних вхідних даних алгоритмічно нерозв'язна.

Ця теорема доводиться досить просто. Перший крок полягає у тому, що вводиться поняття самозастосовності програми. Спочатку доводиться таке твердження: не існує програми, застосовної до всіх несамозастосовних програм і тільки до них.

Перехід до наступного кроку.

Крок 8. Розпізнавання несамозастосовності:

Позначимо буквою B програму, здатну розпізнавати несамозастосовність. Тоді наступна програма буде програмою, застосовною до всіх несамозастосовних програм і тільки до них.

Ця програма зупиняється, якщо розглянута як вхідні дані програма є несамозастосовною, і не зупиняється (зациклюється на кроці 4) у протилежному разі.

Встановіть правильну послідовність:

1. Виконати послідовність команд B, перейти до кроку 2.
2. Якщо отримана відповідь “так”, то перейти до кроку 3, у протилежному разі перейти до кроку 4.
3. Закінчити процес.
4. Перейти до кроку 4.

Якщо не вказано правильну послідовність, то відображається відповідне пояснення. Перехід до наступного кроку.

Крок 9. Припустимо зворотне. Нехай E – програма, що за заданою довільною програмою і заданим на вході словом розпізнає застосовність даної програми до даного слова. Неважко побудувати програму, що дозволяє установити, чи є задане слово кодом даної програми. Позначимо таку програму буквою P.

Встановіть правильну послідовність:

1. Тоді можна побудувати програму H.
2. Застосувати послідовність команд P. Перейти до кроку 2.
3. Якщо послідовність команд P дала відповідь “так”, перейти до кроку 3, інакше – до кроку 4.
4. Виконати послідовність команд E. Кінець.
5. Перейти до кроку 4.

Якщо не вказано правильну послідовність, то відображається відповідне пояснення. Перехід до наступного кроку.

Крок 10. Висновок:

Програма H є програмою, що розпізнає самозастосовність довільних програм. Така програма неможлива, а отже, неможлива і програма E.

Перехід до наступного кроку.

Крок 11. Повідомлення про завершення:

Ви пройшли огляд алгоритмічно нерозв’язних проблем! Завершити роботу тренажеру?

Якщо вибрано вихід, то тренажер закривається. Якщо ні – перехід на стартове вікно.

3.3. Розробка блок-схеми алгоритму

На рисунках 3.1 – 3.4 зображена блок-схема алгоритму роботи тренажеру.

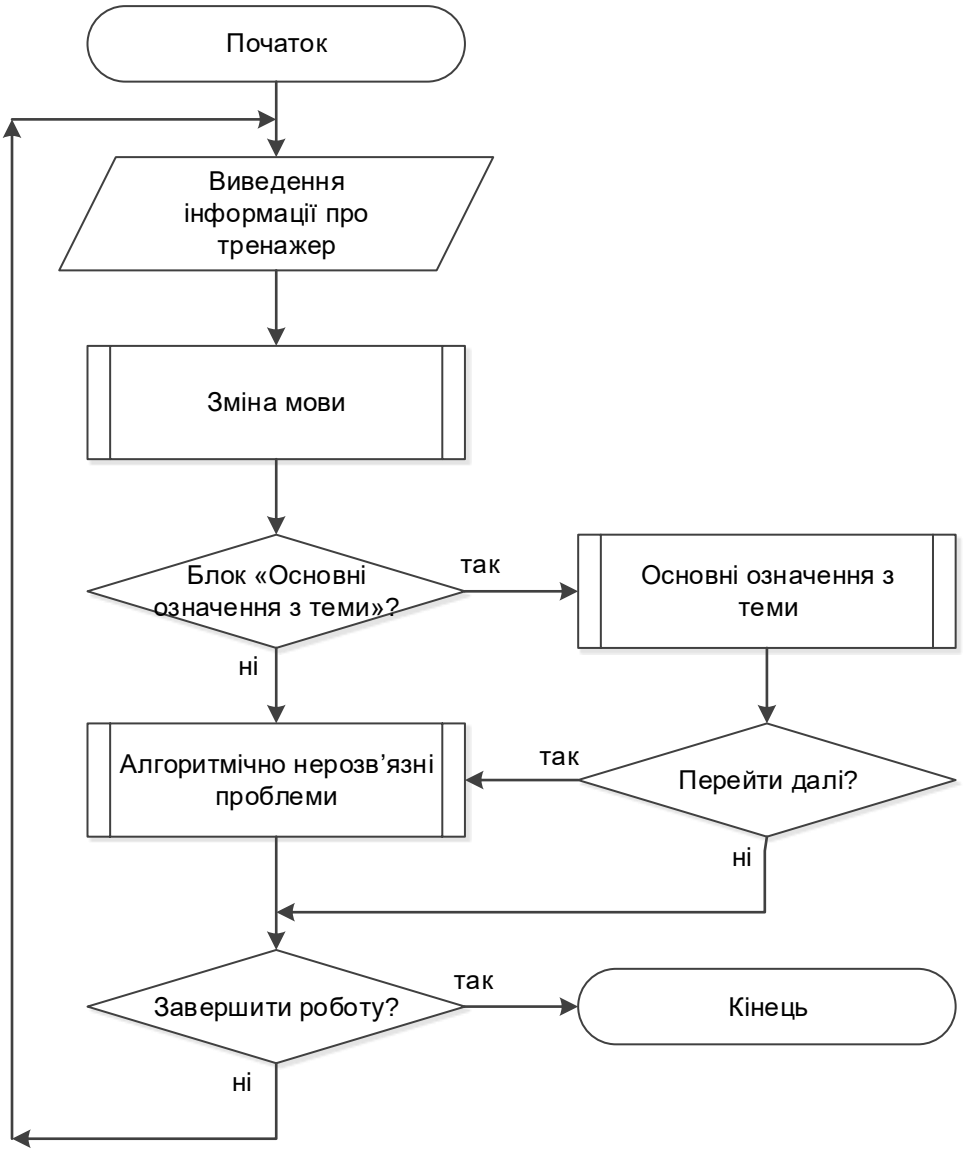

Рисунок 3.1 – Блок-схема алгоритму роботи тренажеру

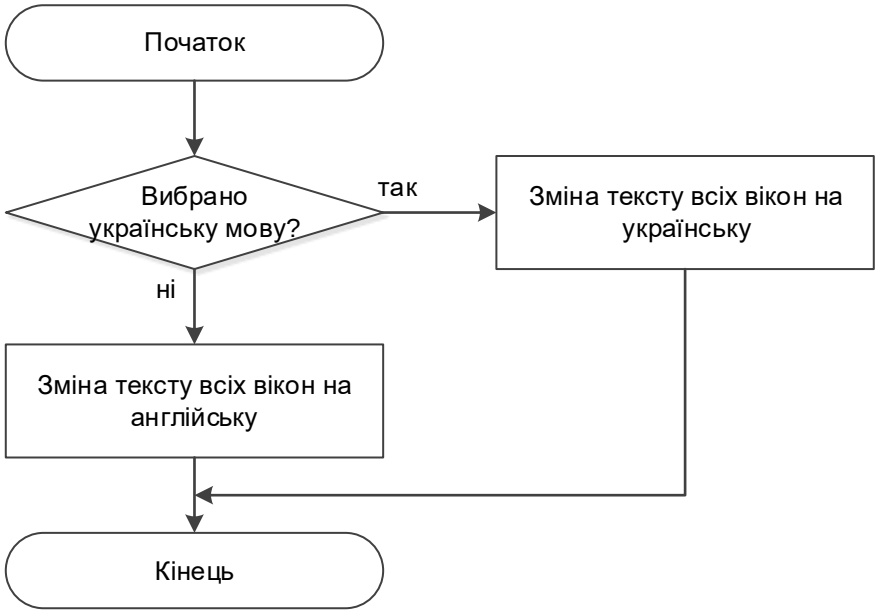

Рисунок 3.2 – Блок-схема процесу «Зміна мови»

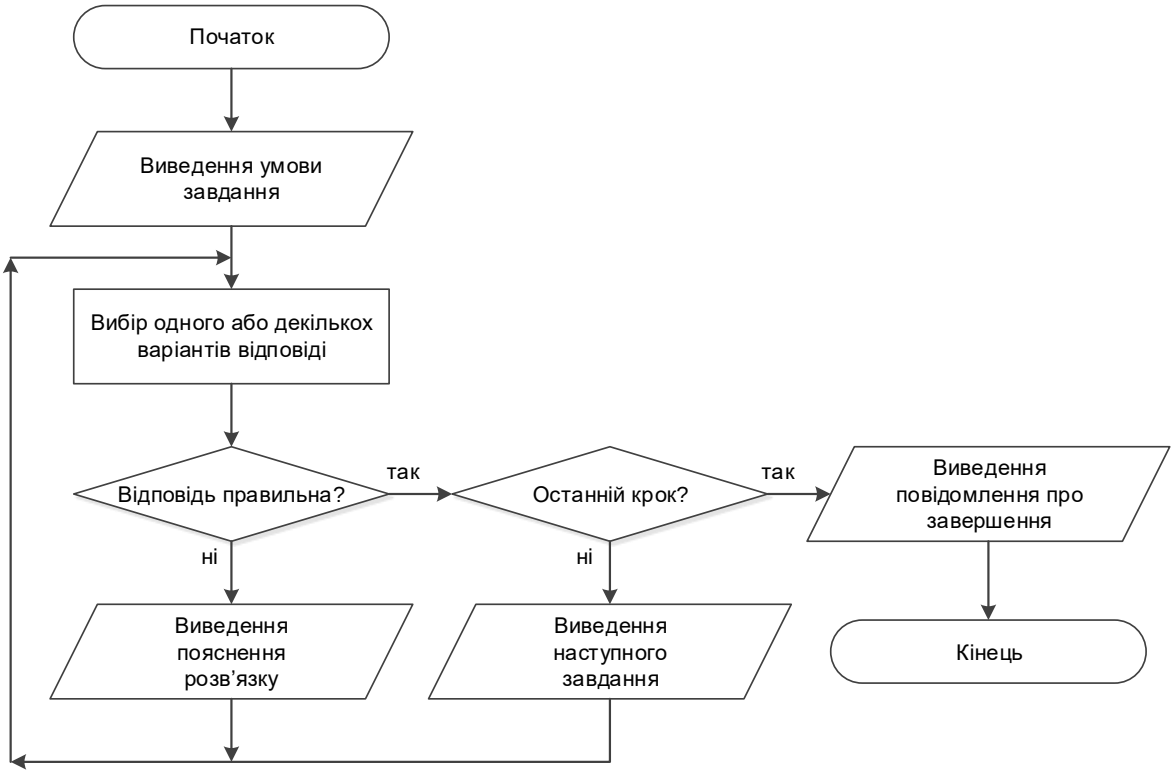

Рисунок 3.3 – Блок-схема процесу «Основні означення з теми»

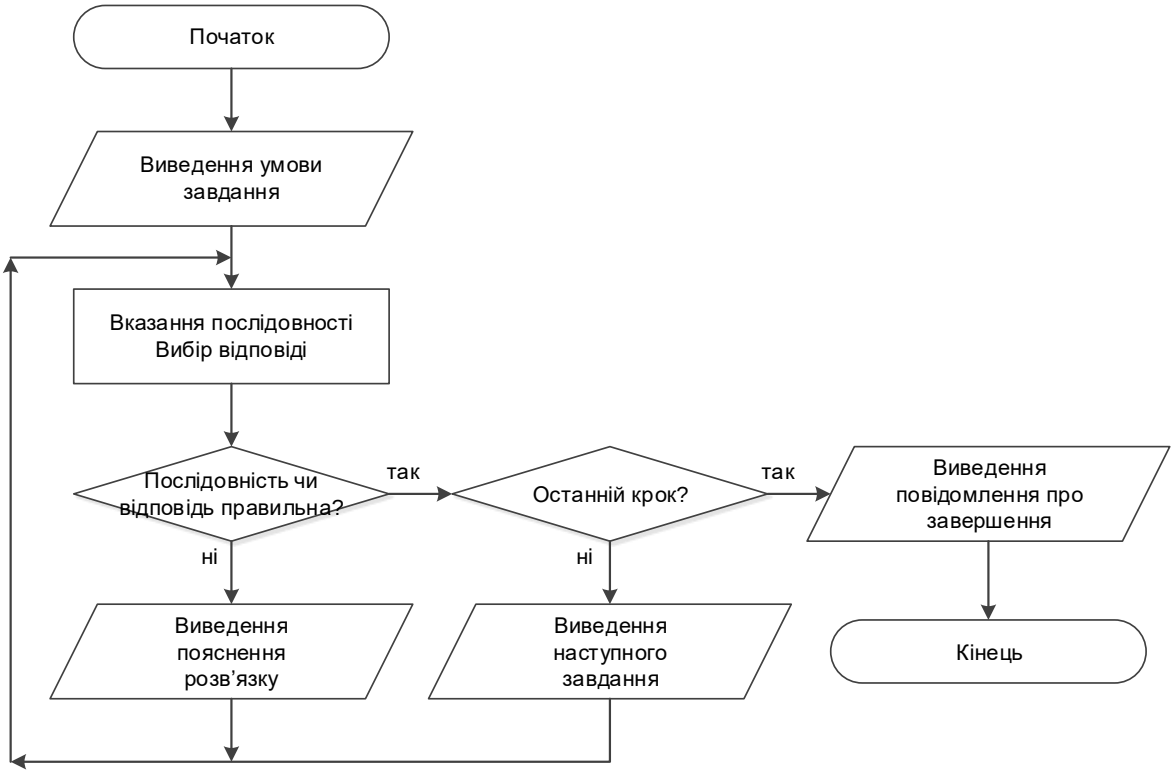

Рисунок 3.4 – Блок-схема процесу «Алгоритмічно нерозв'язні проблеми»

4. ПРАКТИЧНА ЧАСТИНА

4.1. Обґрунтування вибору програмних засобів

C # vs Java - яка з об'єктно-орієнтованих мов програмування найкраще підійде для роботи? Вони обидва мають великі бібліотеки, які можна використовувати для створення додатків для ПК, Інтернет, мобільних пристроїв та інших платформ. У обох є великі спільноти захоплених шанувальників і багато онлайн-підтримки.

Маючи так багато спільного, вибір правильного мови вимагає більш тонкого, ретельного підходу. Давайте докладніше розглянемо подібності та відмінності між C # і Java.

Що таке C #?

C # - це мова програмування загального призначення, яка вперше з'явилась в 2000 році в рамках ініціативи Microsoft .NET. Вона була розроблена для загальної мовної інфраструктури (CLI) - відкритої специфікації, розробленої Microsoft і стандартизованої ISO і ECMA. Додатки C # скомпільовані в байт-код, який може запускатися при реалізації CLI.

Що таке JAVA?

Java, спочатку випущений Sun Microsystems в 1995 році, є мовою програмування загального призначення, яка була розроблена з конкретною метою, що дозволяє розробникам "write once, run anywhere", тобто написати код один раз і запускати в будь-якому місці. Java-додатки скомпільовані в байт-код, який може запускатися при реалізації віртуальної машини Java (JVM). Подібно CLI, JVM допомагає подолати розрив між вихідним кодом і 1 і 0, які розуміє комп'ютер.

C # VS. JAVA: основні подібності.

Поява як Java, так і C #, тісно пов'язана з переходом від низькорівневих мов програмування, таких як мови програмування C ++, до

мов вищого рівня, які компілюються в байт-код. Байт-код можна запустити на віртуальній машині. З цим пов'язаний ряд переваг, в першу чергу, можливість написання коду, який буде зрозумілий людині і буде працювати на будь-якій апаратній архітектурі, на якій встановлена віртуальна машина. Якщо відкинути синтаксичні примхи в сторону, то не дивно, що ці дві подібні між собою мови так популярні для розробників додатків. Ось кілька основних подібностей між C # і Java:

- Безпека типів. Помилка типу виникає, коли тип даних одного об'єкта помилково призначається іншому об'єкту, створюючи ненавмисні побічні ефекти. І C #, і Java працюють на те, щоб гарантувати виявлення таких типів незаконних привидів під час компіляції. Якщо приведення не може бути застосоване до нового типу, тоді під час виконання такі винятки будуть видалені.

- Прибирання сміття: На мовах нижчого рівня управління пам'яттю може бути стомлюючим, адже потрібно пам'ятати про те, що необхідно правильно видалити нові об'єкти, щоб звільнити ресурси. На C # і Java є вбудоване прибирання сміття, яке допомагає запобігти витоку пам'яті шляхом видалення об'єктів, які більше не використовуються додатком. Виток пам'яті все ще можуть виникати, але завдяки основам управління пам'яттю - це вже не ваша проблема.

- Одиночне спадкоємство. Обидві мови підтримують одиночне спадкоємство - це означає, що існує тільки один шлях з будь-якого базового класу в будь-який з його похідних класів. Це обмежує ненавмисні побічні ефекти, які можуть виникати при наявності декількох шляхів між декількома базовими класами і похідними класами. Diamond pattern - книжковий приклад цієї проблеми.

- Інтерфейси. Інтерфейс являє собою абстрактний клас, де всі методи абстрактні. Абстрактним методом є той метод, який оголошений, але не містить подробиць його реалізації. Код, що визначає будь-які методи або властивості, певні інтерфейсом, має надаватися класом, який

його реалізує. Це допомагає уникнути двозначності паттерна diamond, оскільки завжди ясно, який базовий клас реалізує даний похідний клас під час виконання. Результатом є чиста ієрархія лінійних класів одиночного наслідування в поєднанні з деякою універсальністю множинного спадкоємства. Фактично використання абстрактних класів є одним із способів множинного успадкування мов, які можуть подолати проблему паттерна diamond.

C # VS. JAVA: основні відмінності.

Важливо пам'ятати, що C # бере свій початок в бажанні Microsoft мати власний «Java-подібну» мову для платформи .NET. Оскільки C # не створювалася в вакуумі, нові функції були додані і налаштовані для вирішення проблем, з якими стикалися розробники Microsoft, коли вони спочатку намагалися створити свою платформу на Visual J++. У той же час співтовариство Java з відкритим вихідним кодом продовжувала зростати і між цими двома мовами розвивалася гонка технічних озброєнь. Ось деякі з основних відмінностей між C # і Java.

- Windows vs open-source. Хоча існують реалізації з відкритим вихідним кодом, C # в основному використовується в розробці для платформ Microsoft - .NET Framework CLR і є найбільш широко використовуваної реалізацією CLI. На іншому кінці спектру Java має величезну екосистему з відкритим вихідним кодом і у нього відкрилося друге дихання частково завдяки тому, що Google використовує JVM для Android.

- Підтримка узагальнень (Generics): Generics покращує перевірку типів за допомогою компілятора, в основному видаляючи приведення з вихідного коду. В Java кошти узагальнень реалізуються з використанням стирань. Параметри загального типу «стираються», а при компіляції в байт-код додаються приведення. C # також використовує узагальнення, інтегруючи його в CLI і надаючи інформацію про тип під час виконання, що дає невелике збільшення продуктивності.

- Підтримка делегатів (покажчиків): У C # є делегати, які по суті є як методи, які можуть бути викликані не повідомляючи цільового об'єкта. Для досягнення такої ж функціональності в Java вам необхідно використовувати інтерфейс з одним методом або іншим способом обходу, який може зажадати нетривіального кількості додаткового коду, в залежності від програми.

- Перевіряються виключення: Java розрізняє два типи винятків - Перевіряються і ті, що не перевіряються. C # вибрав більш мінімалістський підхід, маючи тільки один тип винятку. Хоча здатність ловити виключення може бути корисна, вона також може мати негативний вплив на масштабованість і контроль версій.

- Поліморфізм: C # і Java використовують дуже різні підходи до поліморфізму. Java допускає поліморфізм за замовчуванням, C # же повинен викликати ключове слово «virtual» в базовому класі і ключове слово «override» в похідному класі.

- Перерахування (Enums): в C # перерахування представляють собою прості списки іменованих констант, де базовий тип повинен бути цілим. Java являє перерахування більш глибоко, розглядаючи його як іменований екземпляр типу, що спрощує додавання користувацької поведінки окремим перерахуванням.

Мову, яку в кінцевому підсумку вирішите використовувати, буде багато в чому залежати від платформи, яку вибрали для свого проекту. Сьогодні C # використовується в основному для реалізації CLI на .NET Framework, Mono і Portable.NET. Якщо програмне забезпечення або веб-додаток створюються для Windows, C # буде працювати найкраще з набором технологій .NET.

Проте, якщо ви хочете розробляти для Unix, Linux або інших платформ за межами платформи Microsoft, екосистема з відкритим вихідним кодом - Java - кращий вибір. Спільнота постійно створює нові бібліотеки та інструменти. З'явилися нові потужні мови, такі як Scala,

Clojure і Groovy, і вони все теж засновані на JVM. До того ж це непогано, що більшість реалізацій JVM є загальнодоступними і безкоштовними. Java - основна мова розробки, який використовує Google для Android - найбільшої мобільної операційної системи в світі в даний час.

Перераховані вище переваги незначні, і жодна з мов не зникне найближчим часом. Обидві мови існують досить довго і, насправді, ви не зможете нічого такого побудувати на одній мові, чого б не змогли побудувати на іншому [9, 10].

4.2. Опис процесу програмної реалізації

Першим чином потрібно розмістити елементи на панель та привести її до потрібного виду. Так було зроблено стартове вікно і кроки для відображення (рис. 4.1, 4.2).

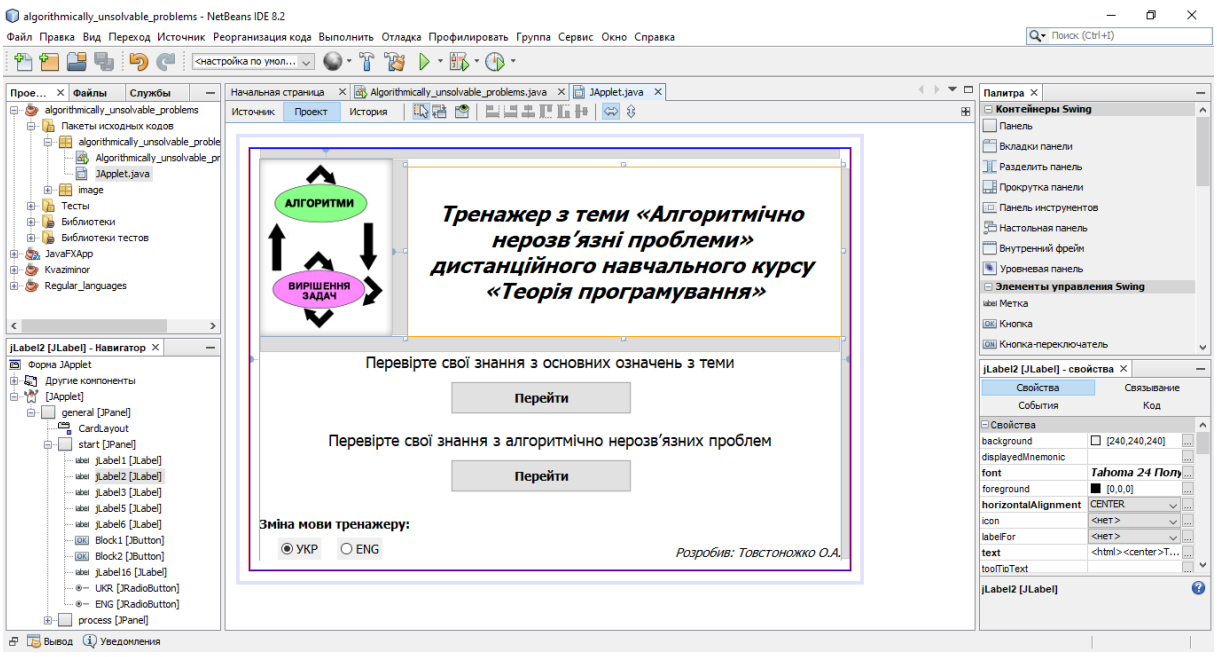

Рисунок 4.1 – Розміщення елементів на стартовому вікні

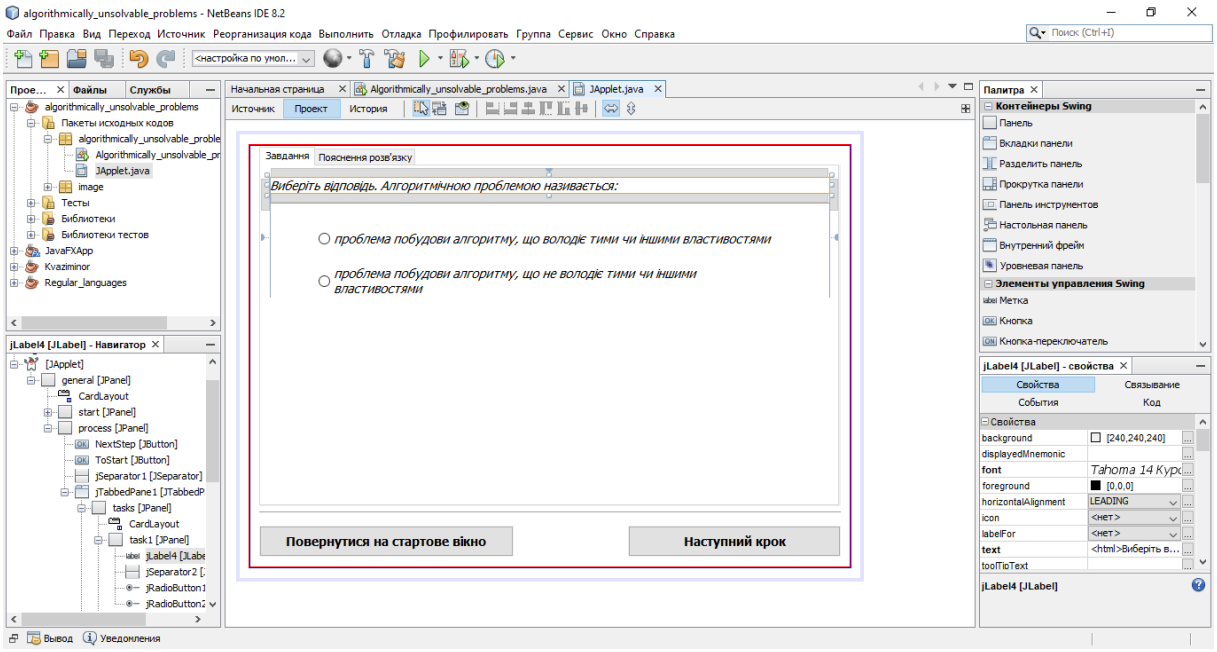

Рисунок 4.2 – Розміщення елементів на панелі з кроками

Тепер щоб тренажер працював потрібно розробити його функціонал. Для відстеження поточного кроку оголошено змінну step типу int, а для виведення панелі відповідно алгоритму – масив task[] типу String з назвами панелей.

Далі було створено функції:

- ShowPanel(JPanel panel, String card);
- CheckTask();
- ClearAll();
- Translate().

Відображення панелі відбувається завдяки картковому розміщенні, тобто одна панель помічається як основна, а інші в ній як картки. ShowPanel(JPanel panel, String card) виводить картку, вказану параметром String card, також вказується назва основної панелі через JPanel panel.

Функція CheckTask() перевіряє чи правильно вказана відповідь на поточному кроці. Якщо ні, то повертає значення типу String з назвою картки, яку необхідно відобразити в поясненні розв’язку.

Оскільки на будь-якому кроці є можливість повернутися на стартове вікно та почати спочатку, то ClearAll() очищує всі обрані раніше відповіді.

Остання функція Translate() служить для перекладу всіх карток з кроками і їх поясненнями розв’язку на вибрану мову.

При виборі мови на стартовому вікні спрацьовують події UKRActionPerformed або ENGActionPerformed. Аналогічно функції Translate() вони змінюють текст елементів всіх основних панелей на вибрану мову.

На стартовому вікні є можливість перейти до одного з двох блоків із завданнями. На кожну кнопку створена відповідна подія Block1ActionPerformed і Block2ActionPerformed. Вони діють схожим образом:

- очищують всі відповіді (ClearAll()),
- змінюють текст елементів згідно вибраної мови (Translate()),
- відображають картку відповідного кроку алгоритму (ShowPanel(tasks, "task1")).

Якщо натиснути кнопку «Повернутися на стартове вікно», то спрацьовує подія ToStartActionPerformed, що виводить стартове вікно.

При натисненні «Наступний крок» відбувається перевірка відповіді і як результат – перехід на наступний крок (функція CheckTask() повертає пусте значення) або виведення пояснення розв'язку (CheckTask() повертає назву картки).

Після перевірки основних означень з теми відображається повідомлення, де потрібно вибрати «Перейти» чи «Завершити роботу» – події NextBlockActionPerformed і Exit1ActionPerformed.

Так, NextBlockActionPerformed переходить до наступного кроку і відображає відповідну картку.

```
private void NextBlockActionPerformed(java.awt.event.ActionEvent evt) {
    NextStep.setEnabled(true);
    ShowPanel(tasks, task[step]);
    step++;
}
```

Подія Exit1ActionPerformed закриває тренажер.

Також було створено три додаткові події. Так як на одному кроці алгоритму варіанти відповіді представлені у вигляді формул, то їх збережено як картинки. Тому, при натисненні на них має вибиратися відповідний jRadioButton.

4.3. Інструкція по використанню тренажеру

Після запуску тренажеру відображається його стартове вікно, де вказано його тему та інформацію про розробника (рис. 4.3).

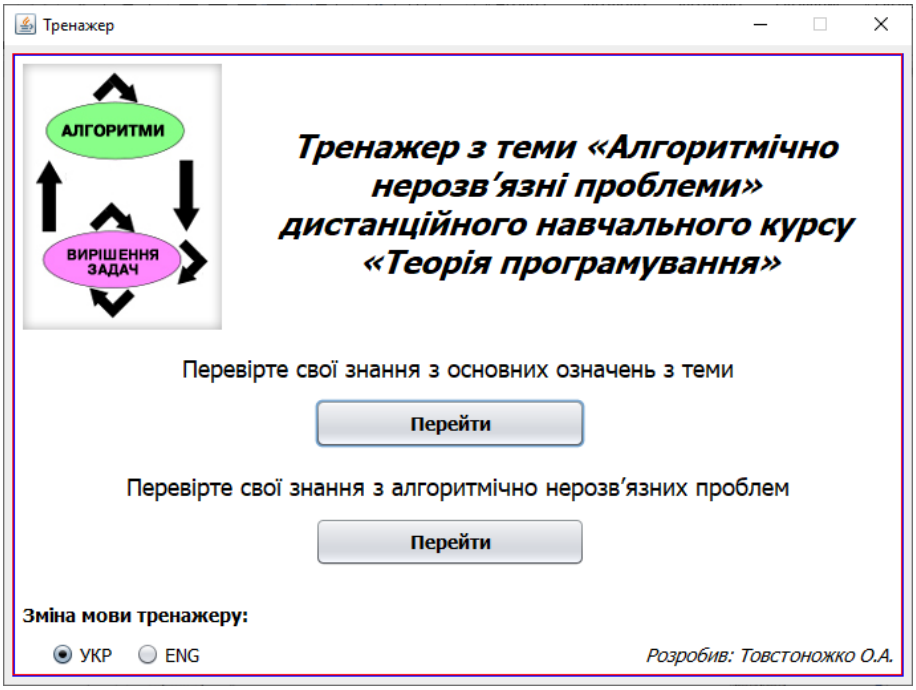

Рисунок 4.3 – Стартове вікно

Щоб змінити мову тренажеру потрібно її вибрати (УКР, ENG), по замовчуванню – українська. При зміні на англійську мову текст автоматично зміниться (рис. 4.4).

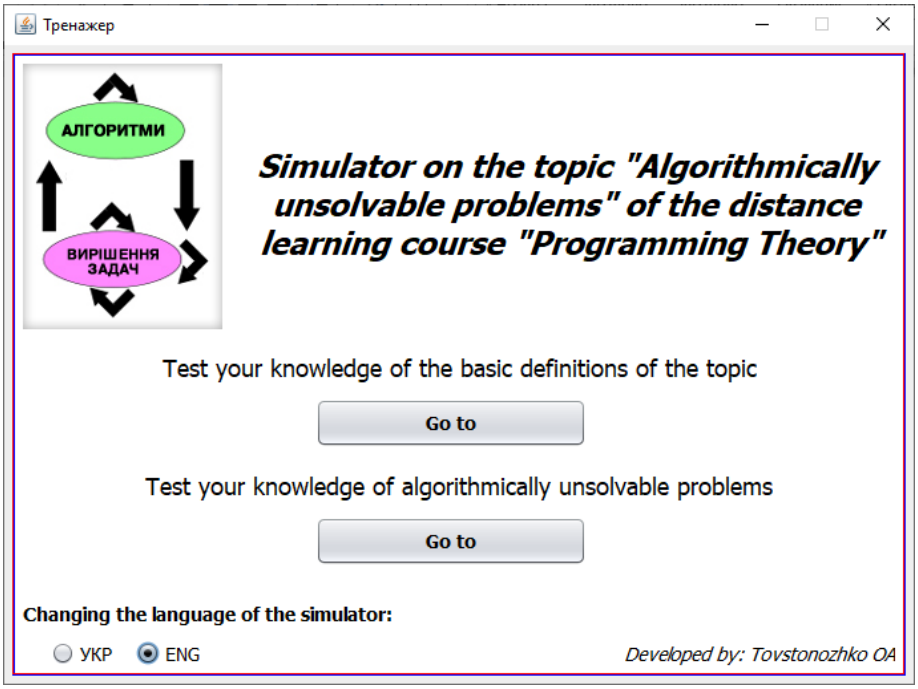

Рисунок 4.4 – Стартове вікно англійською мовою

Для переходу до завдань з перевірки основних означень з теми необхідно натиснути першу кнопку «Перейти». Виведеться перший крок (рис. 4.5).

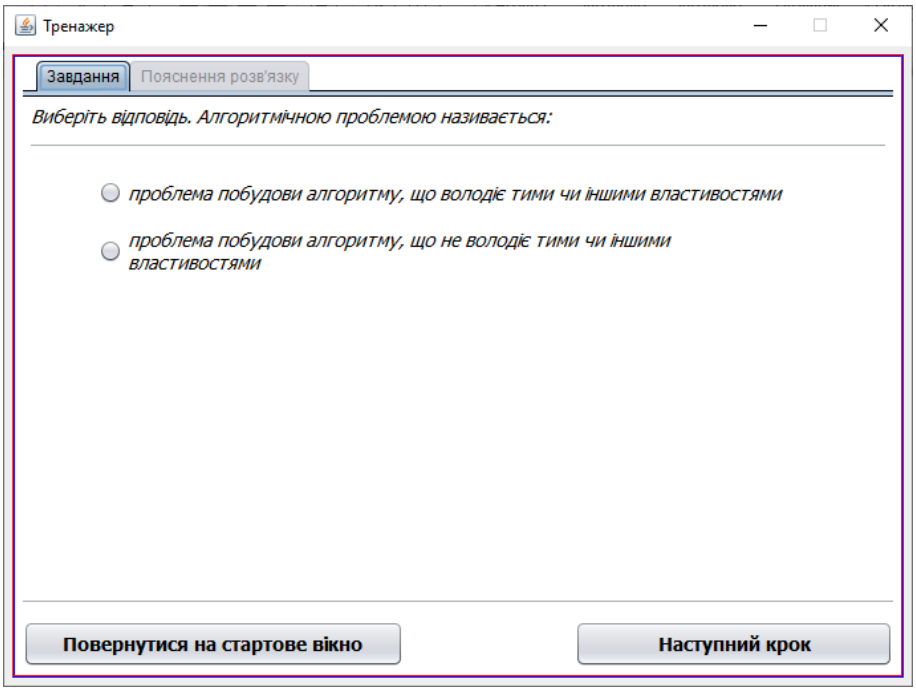

Рисунок 4.5 – Перший крок завдань з перевірки основних означень

Аналогічно для переходу до завдань з алгоритмічно нерозв’язних проблем потрібно натиснути другу кнопку «Перейти» (рис. 4.6).

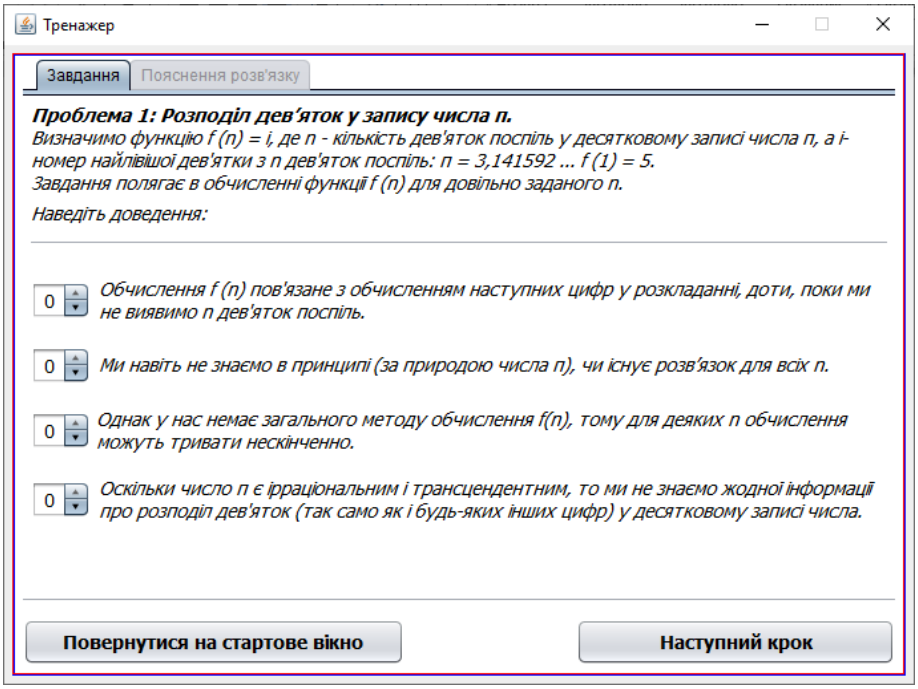

Рисунок 4.6 – Перший крок завдань з алгоритмічно нерозв’язних проблем

Розглянемо більш детально виконання завдань. Так, на першому кроці (рис. 4.5) виводиться умова і потрібно вибрати один з варіантів відповіді. Для її перевірки необхідно натиснути «Наступний крок».

Якщо відповідь правильна – відображається наступний крок (рис. 4.7), якщо ні – вказується пояснення розв’язку (рис. 4.8)

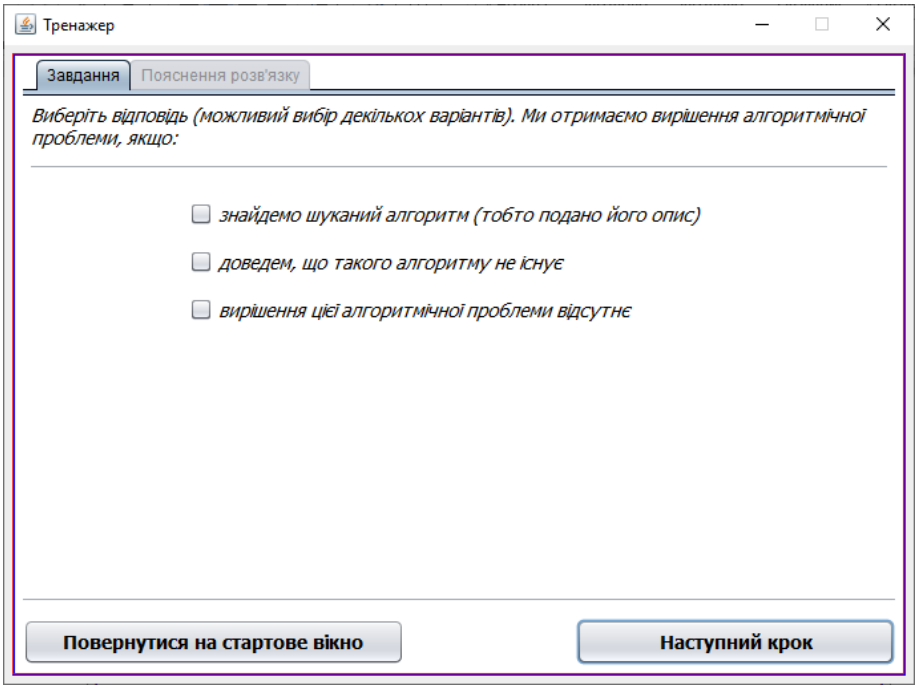

Рисунок 4.7 – Другий крок завдань з перевірки основних означень

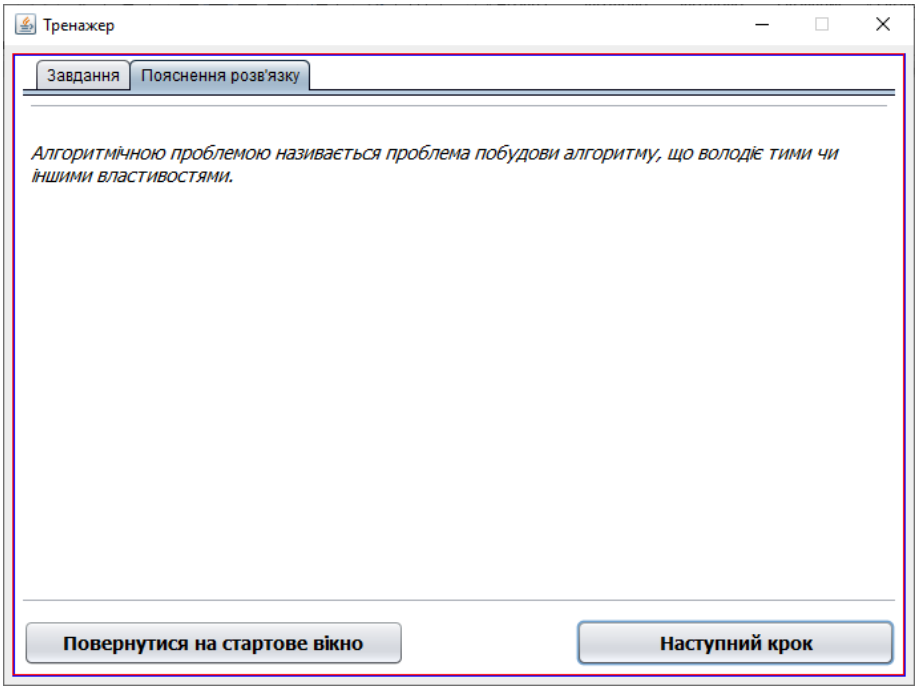

Рисунок 4.8 – Пояснення розв’язку першого завдання з перевірки основних означень

Також слід зазначити, що в будь-який момент можна повернутися на стартове вікно і розпочати все спочатку, змінити мову, перейти до іншого блоку.

На другому кроці (рис. 4.7) вже можливий вибір декількох варіантів відповіді, про це повідомляється в умові.

Після виконання завдань з перевірки основних означень виводиться відповідне повідомлення (рис. 4.9). Можна перейти до завдань з алгоритмічно нерозв’язних проблем або завершити роботу.

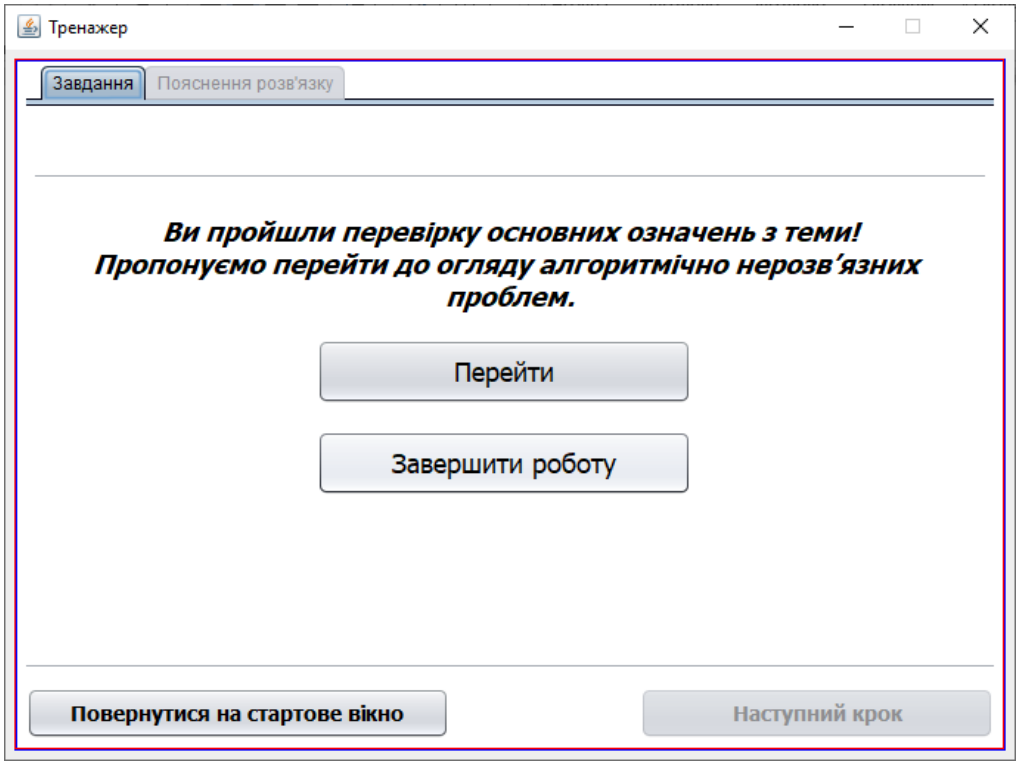

Рисунок 4.9 – Повідомлення про виконання завдань з перевірки основних означень

Далі йдуть завдання на доведення (рис. 4.6), тобто потрібно встановити правильну послідовність.

Якщо встановлено неправильно, то також відображається пояснення розв’язку (рис. 4.10).

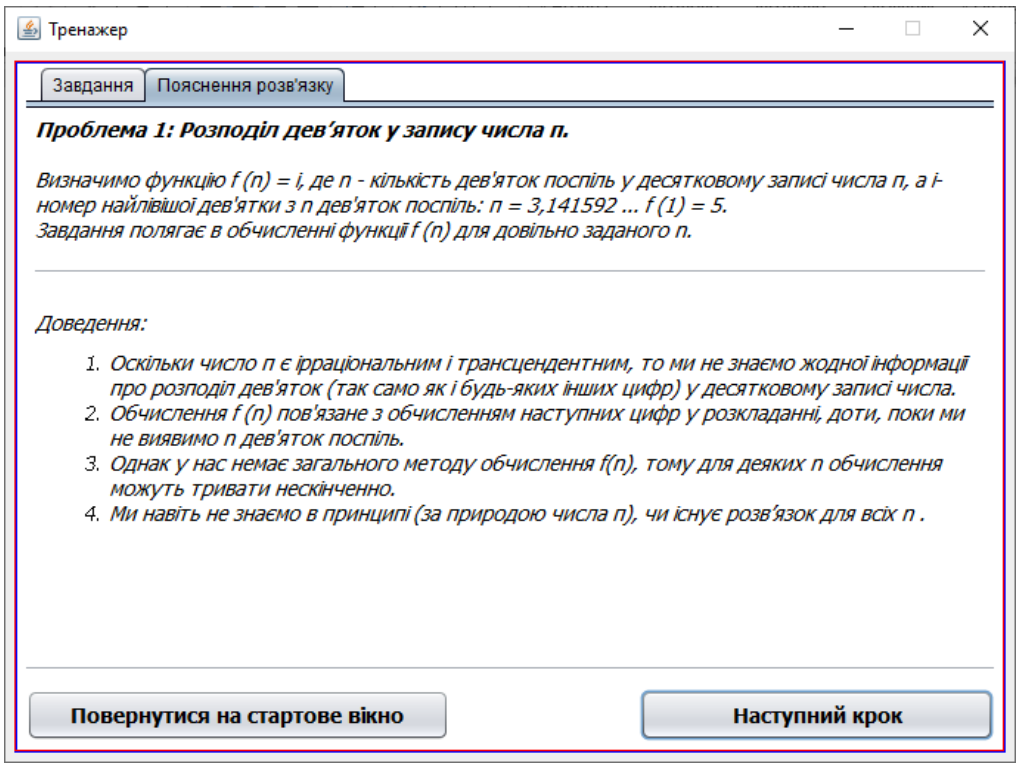

Рисунок 4.10 – Пояснення розв’язку першого завдання з алгоритмічно нерозв’язних проблем

Серед цих завдань можуть зустрічатися вибір відповіді (рис. 4.11) або просто наведення умови (рис. 4.12)

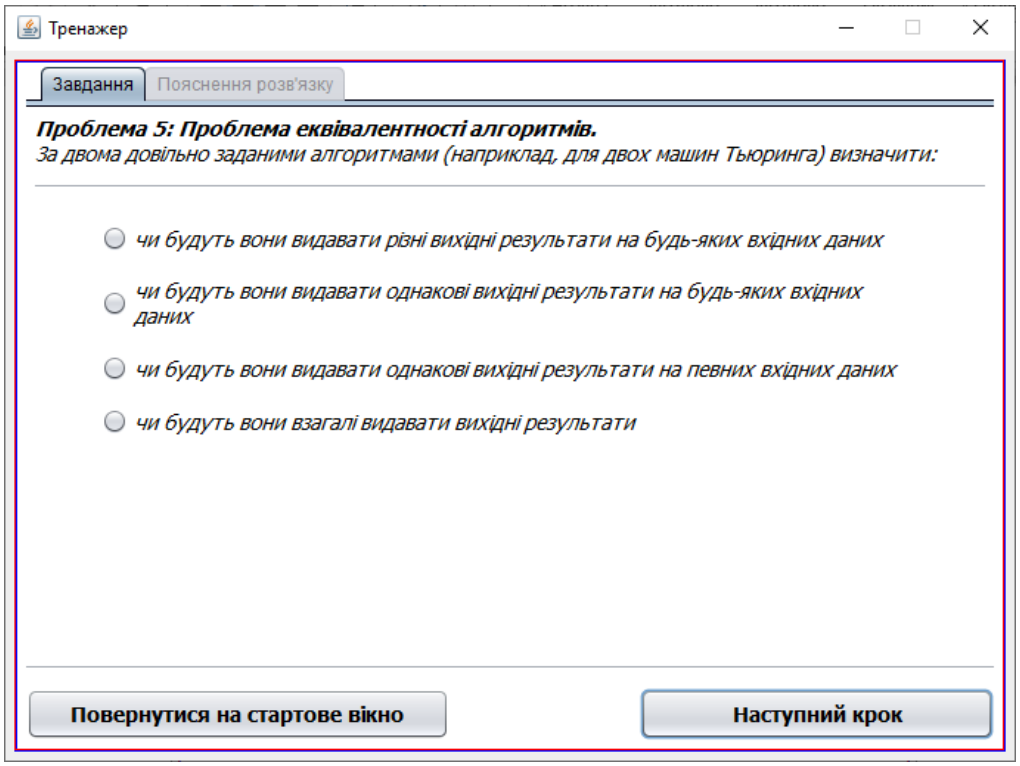

Рисунок 4.11 – П’ятий крок завдань з алгоритмічно нерозв’язних проблем

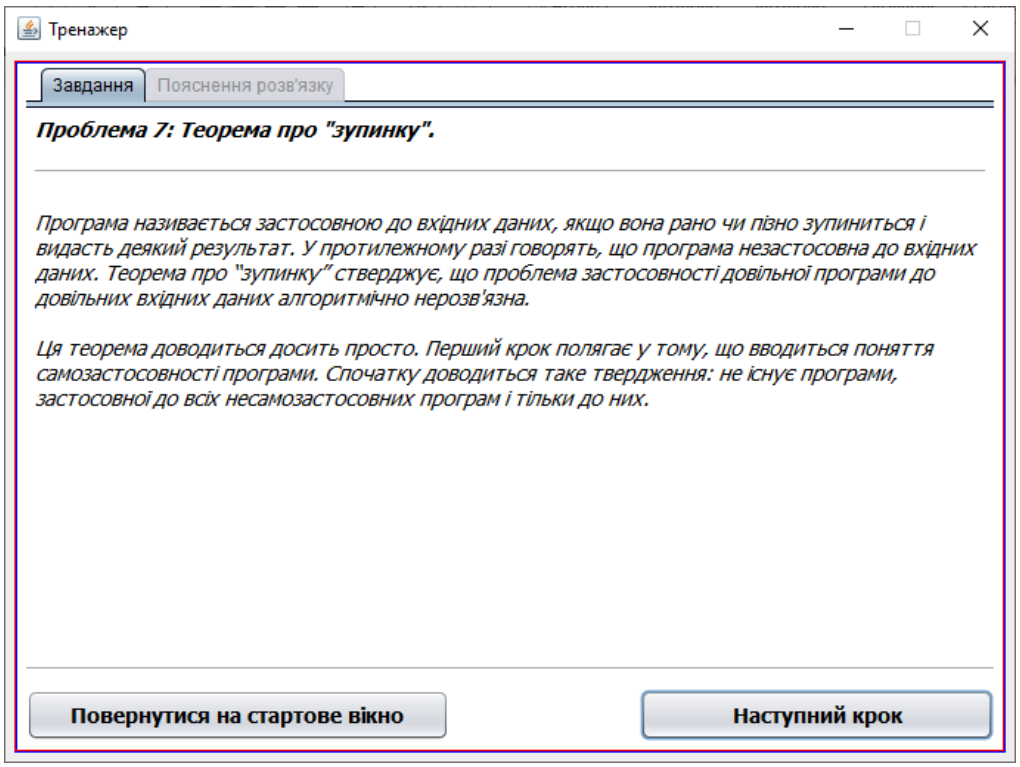

Рисунок 4.12 – Сьомий крок завдань з алгоритмічно нерозв’язних проблем

В кінці також відображається повідомлення про завершення (рис. 4.13).

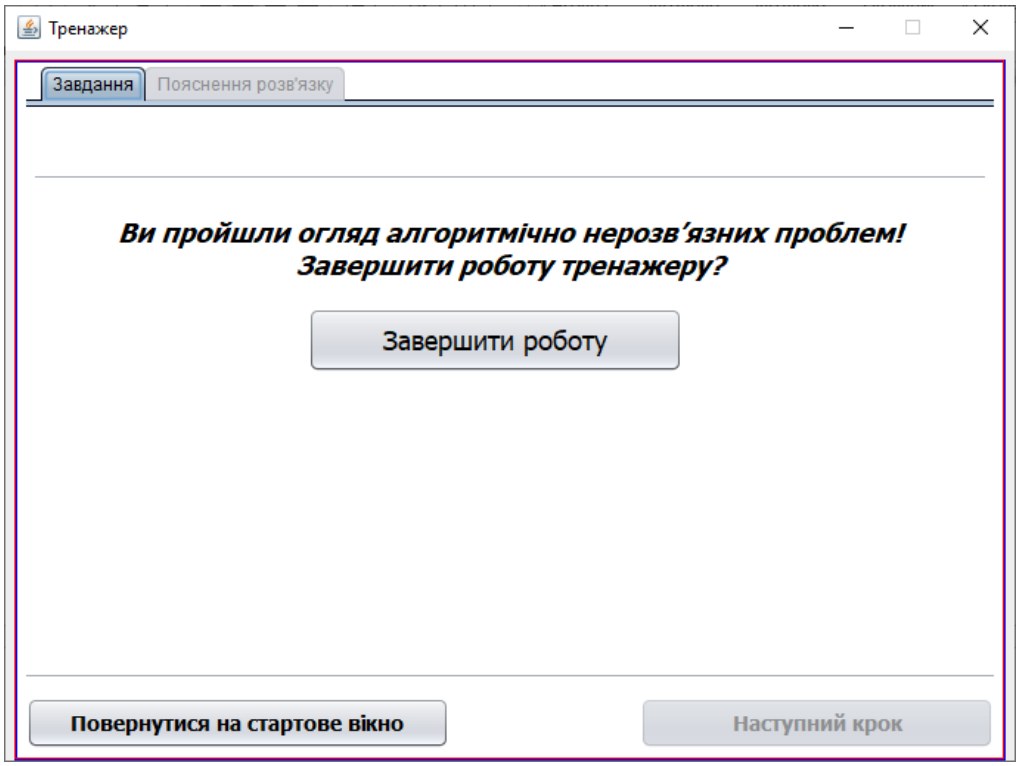

Рисунок 4.13 – Повідомлення про виконання завдань з алгоритмічно нерозв’язних проблем

ВИСНОВКИ

Використання комп'ютерних технологій в освіті є способом підвищення мотивації і індивідуалізації навчання, розвитку творчих здібностей і створення доброзичливого емоційного фону. При реалізації комп'ютерних тренажерів повинні використовуватися додаткові можливості, що надаються комп'ютерним середовищем: мультимедійні ефекти, інтерактивна взаємодія.

Комп'ютерні технології дозволили широко застосовувати в навчанні тренажери. Предметна область для учня не виступає явно, але він оволодіває нею в міру виконання наданих завдань.

Розглянемо виконані завдання проекту:

- описано постановку задачі;
- описано значення та приклади алгоритмічно нерозв’язних проблем, значення навчальних програм для різних дисциплін;
- наведено основні поняття з теми для використання в тренажері;
- розроблено алгоритм роботи тренажеру;
- описано мову програмування та технології, що були використані при розробці програми;
- описано процес реалізації основних етапів створення тренажеру;
- описано необхідну користувачу інструкцію.

При розробці алгоритму розглядалися наступні проблеми:

А) Відсутність загального методу розв'язання задачі

1. Проблема 1. Розподіл дев’яток у запису числа π
2. Проблема 2. Десята проблема Гільберта
3. Проблема 3. Обчислення досконалих чисел

Б) Інформаційна невизначеність завдання

1. Проблема 4. Позиціонування машини Посту на останньому позначеному ящику

В) Логічна нерозв'язність

1. Проблема 5. Проблема еквівалентності алгоритмів
2. Проблема 6. Проблема тотальності

Г) Розпізнавання несамозастосовності в теоремі про «зупинку»

На кожному кроці користувачу виводиться завдання і методи його вирішення. Для переходу до наступного кроку обов’язково потрібно виконати поставлену умову.

Якщо відповідь була надана неправильна, то відобразиться пояснення розв’язку.

Після виконання всіх завдань виведеться результат проходження тренажеру. Пропонується перейти на стартове вікно або вийти.

Для можливості використовувати тренажер для вивчення дистанційного курсу «Теорія програмування» іноземними студентами передбачається можливість зміни мови тренажеру.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ємець О. О. Методичні рекомендації до виконання дипломної роботи для студентів ступеня магістра спеціальності 122 «Комп’ютерні науки» / О.О.(Олег) Ємець. – Полтава : РВВ ПУЕТ, 2018. – 35 с.
2. Панішев А.В. Вступ до теорії складності дискретних задач : Монографія. / А.В. Панішев – Ж. : ЖДТУ, 2004. – 236с.
3. Горлова Т.М. Теорія алгоритмів. [Електронний ресурс]: конспект лекцій для студентів напряму підготовки 6.050101 «Комп'ютерні науки» денної та заочної форм навчання / Т.М. Горлова, К.Є. Бобрівник, Н.В. Лімaнська – К.: НУХТ, 2015. – 95 с.
4. Черненко О.О. Електронний навчально-методичний посібник для самостійного вивчення навчальної дисципліни «Теорія програмування» для студентів напряму 6.040302 «Інформатика»
5. Бондаренко М.Ф. Комп’ютерна дискретна математика: Підручник / М.Ф. Бондаренко, Н.В. Білоус, А.Г. Руткас. – Харків: «Компанія СМІТ», 2004. – 480 с.
6. Алгоритмічно нерозв'язна задача [Електронний ресурс] / Матеріал з Вікіпедії — вільної енциклопедії. Режим доступу: https://uk.wikipedia.org/wiki/Алгоритмічно-нерозв'язна-задача
7. Клыков В. В. Интерактивные компьютерные тренажеры по математическим дисциплинам : Дис. ... канд. техн. наук : 05.13.18 / В.В. Клыков. – Томск, 2005. – 158 с.
8. Проблема зупинки [Електронний ресурс] / Матеріал з Вікіпедії — вільної енциклопедії. Режим доступу: https://uk.wikipedia.org/wiki/Проблема_зупинки
9. Сравнение C Sharp и Java [Електронний ресурс] / Матеріал з Вікіпедії — вільної енциклопедії. Режим доступу: https://ru.wikipedia.org/wiki/Сравнение_C_Sharp_и_Java

10. C# против Java: какой язык программирования общего назначения выбрать? [Електронний ресурс] — Режим доступу: https://itvdn.com/ru/blog/article/csharp_vs_java
11. Бібліографічний запис. Бібліографічний опис. Загальні вимоги та правила складання: ДСТУ 7.1-2006. – [Чинний від 2007-07-01]. – К. : Держспоживстандарт України, 2007. – 47 с.

ДОДАТОК А. КОД ПРОГРАМИ