МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра

інформатики та програмної інженерії

(повна назва кафедри, циклової комісії)

**КУРСОВА РОБОТА**

з Основи програмування. Частина 2. Методології програмування

(назва дисципліни)

на тему: «Гра у 15»

Студента 1-го курсу, групи ІП-44

Сича Серафима Олександровича

Спеціальності 121 «Інженерія програмного забезпечення»

Керівник \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Кількість балів: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Національна оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |
| --- | --- | --- | --- |
| Члени комісії |  |  |  |
|  | (підпис) |  | (посада, вчене звання, науковий ступінь, прізвище та ініціали) |
|  |  |  |  |
|  | (підпис) |  | (посада, вчене звання, науковий ступінь, прізвище та ініціали) |

Київ – 202\_ рік

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

(назва вищого навчального закладу)

Кафедра інформатики та програмної інженерії

Дисципліна Основи програмування. Курсова робота

Напрям «ІПЗ»

Курс 1 Група ІП-44 Семестр 2

ЗАВДАННЯ

на курсову роботу студента

(прізвище, ім’я, по батькові)

1. Тема роботи

2. Строк здачі студентом закінченої роботи

3. Вихідні дані до роботи

4. Зміст пояснювальної записки (перелік питань, які підлягають розробці)

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

6. Дата видачі завдання

КАЛЕНДАРНИЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Назва етапів курсової роботи | Термін виконання етапів роботи | Підписи керівника, студента |
| 1. | Отримання теми курсової роботи |  |  |
| 2. | Підготовка ТЗ |  |  |
| 3. | Пошук та вивчення літератури з питань курсової роботи |  |  |
| 4. | Розробка сценарію роботи програми |  |  |
| 6. | Узгодження сценарію роботи програми з керівником |  |  |
| 5. | Розробка (вибір) алгоритму розв’язання задач |  |  |
| 6. | Узгодження алгоритму з керівником |  |  |
| 7. | Узгодження з керівником інтерфейсу користувача |  |  |
| 8. | Розробка програмного забезпечення |  |  |
| 9. | Налагодження розрахункової частини програми |  |  |
| 10. | Розробка та налагодження інтерфейсної частини програми |  |  |
| 11. | Узгодження з керівником набору тестів для контрольного прикладу |  |  |
| 12. | Тестування програми |  |  |
| 13. | Підготовка пояснювальної записки |  |  |
| 14. | Здача курсової роботи на перевірку |  |  |
| 15. | Захист курсової роботи |  |  |

Студент

(підпис)

Керівник Головченко М.М.

(підпис) (прізвище, ім’я, по батькові)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_ р.

АНОТАЦІЯ

Пояснювальна записка до курсової роботи: 55 сторінок, 24 рисунки, 28 таблиць, 4 посилання.

Мета роботи: з ТЗ

Вивчено методи … розроблено та реалізовано алгоритми методів …

Виконана програмна реалізація алгоритму.

СИСТЕМА НЕЛІНІЙНИХ РІВНЯНЬ, МЕТОД НЬЮТОНА, МЕТОД СІЧНИХ, МЕТОД ЯКОБІ, СИСТЕМА ЛІНІЙНИХ РІВНЯНЬ, МАТРИЦЯ ЯКОБІ.

Зміст

[Вступ 7](#_Toc199023633)

[1 Постановка задачі 8](#_Toc199023634)

[2 Теоретичні відомості 9](#_Toc199023635)

[3 Опис алгоритмів 10](#_Toc199023636)

[3.1 Переміщення плиток по полю 10](#_Toc199023637)

[3.2 Автоматичне розмішування 10](#_Toc199023638)

[3.3 Опис структур 11](#_Toc199023639)

[3.4 Автоматичний розв’язок 11](#_Toc199023640)

[3.5 Утворення стану 12](#_Toc199023641)

[3.6 Перевірка стану на повторення 12](#_Toc199023642)

[4 Опис програмного забезпечення 13](#_Toc199023643)

[4.1 Діаграма класів програмного забезпечення. 13](#_Toc199023644)

[4.2 Опис методів частин програмного забезпечення 14](#_Toc199023645)

[4.2.1 Стандартні методи 14](#_Toc199023646)

[4.2.2 Користувацькі методи 15](#_Toc199023647)

[5 Тестування програмного забезпечення 18](#_Toc199023648)

[5.1 План тестування 18](#_Toc199023649)

[5.2 Приклади тестування 19](#_Toc199023650)

[6 Інструкція користувача 23](#_Toc199023651)

[6.1 Робота з програмою 23](#_Toc199023652)

[6.2 Системні вимоги 26](#_Toc199023653)

[Висновки 27](#_Toc199023654)

[Перелік посилань 28](#_Toc199023655)

[Додаток А Технічне завдання 29](#_Toc199023656)

[Додаток Б Тексти програмного коду 31](#_Toc199023657)

Вступ

«Гра у 15» - класична головоломка, що розвиває гнучкість мислення та можливість обчислення кроків наперед. Курсова робота представляє собою програмну візуалізацію даної гри. Базуючись на ООП, вона допомагає засвоїти базові його принципи а також навчитися працювати з графічним інтерфейсом користувача. Розробка даного застосунку сприяє засвоєнню принципів ООП та включає практику роботи зі структурами даних.

У даній роботі реалізовано алгоритми пошуку стану\* в деревоподібній структурі (префіксне дерево), евристичної оцінки близькості до розв’язку, а також розглянуто алгоритм пріоритетного вибору стану з множини існуючих.

Основними завданнями курсової роботи є розробка програмної версії «Гри у 15», з відповідними перевірками на коректність ходів та можливістю як ручного, так і автоматичного розмішування плиток. Також метою є реалізація можливості автоматичного розв’язку головоломки з довільного стану (моменту розв’язку головоломки користувачем).

\*Стан – характеристика поля, що представляє собою послідовність плиток у вигляді логічного списку, або ж матриці елементів поля.

# Постановка задачі

Розробити програмну реалізацію головоломки «Гра у 15» та виконати наступні пункти:

1. Реалізувати графічний інтерфейс користувача
2. Забезпечити коректне переміщення плиток по полю.
3. Надати можливість автоматичного і ручного розмішування.
4. Надати можливість автоматичного і ручного розв’язку.
5. Забезпечити показ автоматичного розв’язку користувачу.
6. Відображати поточну кількість ходів користувача.
7. Моніторити стан гри та завершити, коли головоломка вирішена.

Таким чином програмний продукт має представляти собою реалізацію як повного функціоналу фізичної версії гри так і специфічного для програмного забезпечення.

# Теоретичні відомості

Гра у 15 – класична головоломка, що складається з поля 4х4. На ньому розміщуються 15 плиток з номерами від 1 до 15 та залишається пуста клітинка. Ходи відбуваються шляхом переміщення плиток, сусідніх до пустої на її місце. Початковий стан являє собою випадкове розміщення плиток на полі, проте в даній роботі розмішування відбувається аналогічними «ходами» для збереження можливості розв’язку пазлу, оскільки половина випадкових конфігурацій не мають розв’язку. Задачею гри являється утворення такої конфігурації стану, де всі плитки будуть розміщені в порядку зростання, а пусте місце буде знаходитися в правому нижньому куті поля.

Для ефективного автоматичного розв’язку застосовується А\* - алгоритм знаходження оптимального розв’язку пазлу. Для його реалізації використано наступні алгоритми та структури:

1. Пріоритетна черга для обрання найкращих станів
2. Алгоритм оцінки евристики стану
3. Префіксне дерево для збереження відвіданих станів та їх вартості

Евристика алгоритму побудована на базі манхетенської відстані та лінійного конфлікту

# Опис алгоритмів

Основний алгоритм гри складається з декількох етапів:

1. Переміщення плиток по полю
2. Автоматичне розмішування

Вони не являються повноцінними алгоритмами та реалізовані як реакція на сигнали і тому не мають визначеної точки входу але їх сукупність утворює базову логіку гри. Оскільки дані алгоритми, або їх частини, не обов’язково будуть прив’язані до сталої послідовності то будуть розглядатися без алгоритмічної послідовності кроків.

Також присутній алгоритм автоматичного вирішення.

## Переміщення плиток по полю

Відбувається на натиснення кнопки-плитки або за автоматичного сортування.

Якщо дана плитка є сусідньою до пустої клітинки, то перемістити плитку на її місце.

Інакше ігноруємо натиснення.

## Автоматичне розмішування

1. Повторити визначену кількість раз:
   1. Генеруємо випадкове число з діапазону [0, 14]
   2. Переміщуємо плитку за даним індексом в масиві за умови, що вона не повторює попередній крок.

Автоматичний розв’язок представлений алгоритмом A\* а тому не має власної математичної моделі, бо являється евристичним алгоритмом. З цієї причини даний алгоритм не можна повністю відокремити від його програмної реалізації та описати не згадавши структури обробки відвіданих станів та інших під алгоритмів.

## Опис структур

1. Пріоритетна черга:

Пріоритетна черга формується на базі min-heap і тому у вершині має завжди найкращий, відповідно до зазначених методів порівняння.

1. Структура збереження існуючих станів:

Структура являє собою, модифіковане для збереження станів та їх евристики, префіксне дерево.

Кожен вузол зберігає елемент послідовності плиток стану та перелік наступних вузлів. Шлях від вершини до останнього вузла (листка) являється станом гри. Відповідно для кожного вузла буде справедливе твердження, що вся сукупність її дочірніх станів не буде містити таких елементів, які вже наявні в одному з її батьківських станів (їх ланцюга)

Вузол на 16 рівні відповідно не буде мати нащадків, а його значення поточної кількості кроків, що до цього було рівне 0, змінюється на кількість кроків для досягнення даного стану.

## Автоматичний розв’язок

1. ПОЧАТОК
2. Сформувати початковий стан на основі вихідної матриці.
3. Додати початковий стан до пріоритетної черги.
4. Цикл обробки станів, поки верхній не являється кінцевим:
   1. Для верхнього елементу черги:
      1. Створити всі похідні стани:
         1. Для кожного напрямку руху пустої клітинки (плитки 0):
            1. ЯКЩО даний напрямок руху плитки 0 не буде означати вихід за поле ТА не буде являтися оберненим до останнього руху, ТО Утворити стан (підрозділ 3.5)
            2. Перевірити стан на повторення (підрозділ 3.6)
            3. ЯКЩО стан не являється повтором, ТО додати у структуру збереження існуючих станів та до черги.
   2. Видалити верхній елемент черги.
5. КІНЕЦЬ

## Утворення стану

1. ПОЧАТОК
2. Скопіювати матрицю батьківського стану.
3. Обміняти місцями пусту плитку (елемент стану зі значенням “0” – еквівалент пустої клітинки) та плитку з відповідної сторони від неї.
4. Обчислити евристику (сума відстаней по горизонталі та вертикалі від кожної плитки окрім пустої до її цільового місцезнаходження)
5. КІНЕЦЬ

## Перевірка стану на повторення

1. ПОЧАТОК
2. Цикл для кожного елементу матриці стану:
   1. ЯКЩО в списку нащадків вузла немає вузла з даним значенням, ТО додати такий до списку.
   2. Зробити даний стан активним
3. ЯКЩО поточна кількість кроків рівна нулю (такого стану ще не було) АБО більша за кількість кроків нового стану, ТО оновити на кількість даного стану. Такий стан вважаємо новим.
4. КІНЕЦЬ

# Опис програмного забезпечення

## Діаграма класів програмного забезпечення.



Рисунок 4.2 – Діаграма класів

## Опис методів частин програмного забезпечення

#### Стандартні методи

У таблиці 4.1 наведено перелік стандартних методів

Таблиця 4.1 – Стандартні методи

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва класу | Назва методу | Призначення методу | Опис вхідних параметрів | Опис вихідних параметрів |
| 1 | std::vector | push\_back | Додає елемент у кінець вектора | Елемент | - |
| 3 | std::vector | begin | Ітератор на початок | - | Ітератор |
| 4 | std::vector | end | Ітератор на кінець | - | Ітератор |
| 5 | std::priority\_queue | push | Додає елемент у чергу | Елемент | - |
| 6 | std::priority\_queue | pop | Видаляє верхній елемент | - | - |
| 7 | std::priority\_queue | top | Повертає верхній елемент | - | Посилання на елемент |
| 8 | std::priority\_queue | empty | Чи черга порожня | - | bool |
| 9 | std::stack | push | Додає елемент у стек | Елемент | - |
| 10 | std::stack | pop | Видаляє верхній елемент | - | - |
| 11 | std::stack | top | Повертає верхній елемент | - | Посилання на елемент |

Продовження таблиці 4.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва класу | Назва методу | Призначення методу | Опис вхідних параметрів | Опис вихідних параметрів |
| 12 | std::stack | empty | Чи стек порожній | - | bool |
| 18 | QWidget | move | Переміщує віджет | Нова позиція (QPoint) | - |
| 19 | QWidget | pos | Повертає поточну позицію віджета | - | QPoint |

#### Користувацькі методи

У таблиці 4.2 наведено перелік користувацьких методів

Таблиця 4.2 – Користувацькі методи

| № п/п | Назва класу | Назва методу | Призначення методу | Опис вхідних параметрів | Опис вихідних параметрів |
| --- | --- | --- | --- | --- | --- |
| 1 | Puzzle | showResult | Повідомляє про розв’язану гру, відображає результат | title – заголовок вікна, message – текст вікна | - |
| 2 | AutomaticSolve | solveThis | Алгоритм пошуку розв’язку | sourceArray – вхідна матриця стану | - |
| 3 | ExistingStatesTree | isAdded | Повертає логічний результат чи додано в перелік проглянутих станів | currentState – стан для якого виконується перевірка | bool - результат |

Продовження таблиці 4.2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва класу | Назва методу | Призначення методу | Опис вхідних параметрів | Опис вихідних параметрів |
| 4 | ExistingStatesTree | deleteSubtree | Рекурсивне вивільнення пам’яті префіксного дерева | node – вузол для якого виконуємо вивільнення пам’яті | - |
| 5 | Field | isNear | Перевіряє, чи знаходяться плитки поряд одна з одною | tile1, tile2 – покажчики на плитки, що перевіряємо на сусідство | bool – результат перевірки |
| 6 | Field | moveTile | Переміщення плитки, якщо вона поряд з пустою | tile – покажчик на плитку | - |
| 7 | Field | isSorted | Перевіряє, чи відсортовано поле | - | bool – результат перевірки |
| 8 | Field | automaticShuffle | Автоматично перемішує плитки випадковим чином | - | - |
| 9 | Field | manualShuffle | Активує режим ручного перемішування, ховає непотрібні елементи керування | - | - |

Продовження таблиці 4.2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва класу | Назва методу | Призначення методу | Опис вхідних параметрів | Опис вихідних праметрів |
| 10 | State | calculateHeuristic | Обчислює евристику для стану | - | - |
| 11 | State | isSame | Порівнює поточний стан з іншим на ідентичність | other – інший стан для порівняння | bool – true, якщо однакові |
| 12 | State | canMove | Перевіряє, чи можна перемістити нульову плитку в заданому напрямку | zero\_move – напрямок ходу | bool – true, якщо можливо |

# Тестування програмного забезпечення

## План тестування

Метою даного розділу є перевірка головних функцій програми та обробки виключних ситуацій. Тестування охоплює ключові способи взаємодії користувача з інтерфейсом. Результати тестів оформлено у вигляді таблиць.

1. Перевірка можливості переміщення фішки, яка межує з пустою.
2. Перевірка заборони руху фішки, яка не межує з пустою.
3. Перевірка запуску розв’язування при вже вирішеній конфігурації за умови ручного розмішування.
4. Перевірка блокування інших дій під час автоматичного вирішення.
5. Перевірка оновлення лічильника кроків при ручному переміщенні плиток в стані розв’язування.
6. Перевірка відображення вікна результату.

## Приклади тестування

Таблиця 5.1 – Перевірка можливості переміщення фішки, яка межує з пустою клітинкою.

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість переміщення фішки, яка межує з пустою |
| Початковий стан програми | Відкрите поле гри |
| Вхідні дані | Клік по суміжній до пустої плитці |
| Схема проведення тесту | Натискання на суміжну плитку |
| Очікуваний результат | Плитка переміщується, лічильник кроків збільшується на 1 |
| Стан програми після проведення випробувань | Плитка переміщена |

|  |  |
| --- | --- |
| Мета тесту | Перевірити заборону руху фішки, яка не межує з пустою |
| Початковий стан програми | Відкрите поле гри |
| Вхідні дані | Клік на віддаленій фішці |
| Схема проведення тесту | Натискання на фішку, що не межує з пустою |
| Очікуваний результат | Жодної реакції |
| Стан програми після проведення випробувань | Фішка не змінила положення |

Таблиця 5.2 - Перевірка заборони руху фішки, яка не межує з пустою клітинкою.

Таблиця 5.3 - Перевірка запуску розв’язування при вже вирішеній конфігурації за умови ручного розмішування.

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість запуску розв’язування при ручному розмішуванні за умови, що воле вже впорядковане |
| Початковий стан програми | Поле впорядковане у стані ручного розмішування |
| Вхідні дані | Останнє переміщення плитки призвело до упорядкованого поля |
| Схема проведення тесту | Ручне виставлення вирішеної конфігурації поля |
| Очікуваний результат | Кнопка початку розв’язування стає неактивною до зміни конфігурації |
| Стан програми після проведення випробувань | Далі продовжується ручне розмішування, кнопка початку розв’язування неактивна до зміни конфігурації |

Таблиця 5.4 - Перевірка блокування інших дій під час автоматичного вирішення.

|  |  |
| --- | --- |
| Мета тесту | Перевірити блокування інших дій під час автоматичного вирішення |
| Початковий стан програми | Поле у стані розв’язування |
| Вхідні дані | Натиснення на кнопку автоматичного розв’язкування |
| Схема проведення тесту | В процесі розв’язування натиснення на плитки |
| Очікуваний результат | Плитки неактивні |
| Стан програми після проведення випробувань | Розв’язування завершилося без виконання змін зі сторони користувача |

Таблиця 5.5 - Перевірка оновлення лічильника кроків при ручному переміщенні плиток в стані розв’язування.

|  |  |
| --- | --- |
| Мета тесту | Перевірити оновлення лічильника кроків при ручному русі |
| Початковий стан програми | Початок розв’язування, поле з початковим лічильником 0 |
| Вхідні дані | Кілька допустимих переміщень |
| Схема проведення тесту | Зробити N переміщень |
| Очікуваний результат | Лічильник показує “Moves: N” |
| Стан програми після проведення випробувань | Поле оновлене, кількість кроків збільшується на 1 при кожному переміщенні плитки |

Таблиця 5.6 - Перевірка відображення вікна результату.

|  |  |
| --- | --- |
| Мета тесту | Перевірити відображення результату після успішного завершення гри у ручному та автоматичному режимах |
| Початковий стан програми | Програма запущена, розташування фішок або вручну розмішане, або автоматично перемішане |
| Вхідні дані | Послідовні правильні переміщення фішок, що призводять до виграшної конфігурації; або натискання кнопки автоматичного розв’язання |
| Схема проведення тесту | 1. Завершити гру шляхом переміщень плиток вручну  2. Завершити гру за допомогою кнопки «Автоматичне розв’язання» |
| Очікуваний результат | З’являється вікно з привітанням, виводиться кількість ходів |
| Стан програми після проведення випробувань | Відображено QMessageBox із повідомленням про успіх, кнопки розмішування неактивні, показана кількість ходів, при натисненні кнопки «Retry» |

# Інструкція користувача

## Робота з програмою

Після запуску виконавчого файлу з розширенням \*.exe, відкривається головне вікно програми (Рисунок 6.1).

Зображення, що містить текст, знімок екрана, Шрифт, число

Вміст, створений ШІ, може бути неправильним.

Рисунок 6.1 – Головне вікно програми

При натисненні кнопки ручного розмішування («Manual shuffle») програма перейде в даний режим(рисунок 6.2):

Зображення, що містить текст, знімок екрана, Барвистість, калькулятор

Вміст, створений ШІ, може бути неправильним.

Рисунок 6.2 – Режим ручного розмішування

Ручне розмішування відбувається шляхом переміщення плиток і виконується так само, як і при розв’язанні пазлу. Плитка переміщується на вільне місце за умови, що в момент натиснення на неї вона перебувала поряд з пустим місцем, але не по діагоналі.

За натиснення кнопки «Start» запускається процес ручного розв’язування і разом з тим лічильник ходів. В той же стан програми переводить і натиснення кнопки «Automatic shuffle» в головному меню(рисунок 6.3).

Зображення, що містить текст, знімок екрана, Барвистість, Шрифт

Вміст, створений ШІ, може бути неправильним.

Рисунок 6.3 – Режим ручного розв’язування

В даному стані в будь-який момент доступна кнопка автоматичного вирішення. За її натиснення програма спочатку переходить в стан розв’язку(рисунок 6.4), а потім у режим відображення знайденого шляху (послідовності переміщень плиток для розв’язку)(рисунок 6.5).

Зображення, що містить текст, знімок екрана, Барвистість

Вміст, створений ШІ, може бути неправильним.

Рисунок 6.4 – Стан автоматичного обчислення шляху

Зображення, що містить текст, знімок екрана, Барвистість

Вміст, створений ШІ, може бути неправильним.

Рисунок 6.5 – Режим відображення знайденого розв’язку

Після відображення всіх ходів знайденого розв’язку, або при утворенні впорядкованого поля шляхом ручного розв’язування, буде відображене вікно привітання з відображенням кількості кроків зроблених користувачем або за автоматично знайденим розв’язком(рисунок 6.6).

Зображення, що містить текст, знімок екрана, Шрифт, число

Вміст, створений ШІ, може бути неправильним.

Рисунок 6.6 – Відображення вікна привітання і результату

На натиснення кнопки «Retry» буде анульовано лічильник ходів і програма перейде в початковий стан. При натисненні кнопки «Close» буде закрите вікно програми.

## Системні вимоги

Системні вимоги до програмного забезпечення наведені в таблиці 6.1.

Таблиця 6.1 – Системні вимоги програмного забезпечення

|  |  |  |
| --- | --- | --- |
|  | Мінімальні | Рекомендовані |
| Операційна система | Windows® XP/Windows Vista/Windows 7/ Windows 8/Windows 10 (з останніми обновленнями) | Windows 7/ Windows 8/Windows 10  (з останніми обновленнями) |
| Процесор | Intel® Pentium® ІІІ  1.0 GHz або  AMD Athlon™ 1.0 GHz | Intel® Pentium® D або AMD Athlon™ 64 X2 |
| Оперативна пам'ять | 256 MB RAM (для Windows® XP) / 1 GB RAM (для Windows Vista/Windows 7/  Windows 8/Windows 10) | 2 GB RAM |
| Відеоадаптер | Intel GMA 950 з відеопам'яттю об'ємом не менше 64 МБ (або сумісний аналог) | |
| Дисплей | 800х600 | 1024х768 або краще |

Продовження таблиці 6.1

|  |  |  |
| --- | --- | --- |
|  | Мінімальні | Рекомендовані |
| Прилади введення | Клавіатура, комп’ютерна миша | |
| Додаткове програмне забезпечення | Microsoft .Net Framework 4.5.2 або вище | |

Висновки

Коротко описати, що було виконано в кожному розділі. Можливо описати шляхи покращення розробленого програмного забезпечення.

Перелік посилань

Додаток А Технічне завдання

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра

інформатики та програмної інженерії

Затвердив

Керівник \_\_Головченко М.М.\_\_\_\_

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_202\_ р.

Виконавець:

Студент \_\_\_\_<*ПІБ>\_\_\_\_\_\_\_\_\_\_*

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_202\_ р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання курсової роботи

на тему: <*Тема курсової роботи*>

з дисципліни:

«Основи програмування. Курсова робота»

Київ 202\_

* 1. *Мета*: Метою курсової роботи є …………..………..………………
  2. *Дата початку роботи*: «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 202\_ р.
  3. *Дата закінчення роботи*: «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 202\_ р.
  4. *Вимоги до програмного забезпечення*.

1. Функціональні вимоги:

* Можливість ……………………………………………………………….
* <*вимоги до функціональних характеристик*>.

1. Нефункціональні вимоги:

* Можливість ……………………………………………………………….
* <*вимоги до якості та сумісності програмних засобів тощо*>.
* Все програмне забезпечення та супроводжуюча технічна документація повинні задовольняти наступним ДЕСТам:

ДСТУ 3008 - 2015 - Розробка технічної документації.

* 1. *Стадії та етапи розробки програмного забезпечення*:

1. Розробка алгоритмічної складової програмного забезпечення (до\_\_.\_\_.202\_ р.)
2. Об’єктно-орієнтований аналіз предметної області завдання (до\_\_.\_\_.202\_ р.)
3. Об’єктно-орієнтоване проєктування програмного забезпечення (до \_\_.\_\_.202\_р.)
4. Розробка програмного забезпечення (до \_\_.\_\_.202\_р.)
5. Тестування розробленого програмного забезпечення (до \_\_.\_\_.202\_р.)
6. Демонстрація та захист програмного забезпечення (до \_\_.\_\_.202\_ р.)
   1. *Порядок контролю та приймання*. Поточні результати роботи над КР регулярно демонструються викладачу. Своєчасність виконання основних етапів графіку підготовки роботи впливає на оцінку за КР відповідно до критеріїв оцінювання.

Додаток Б Тексти програмного коду

*студента групи ІП-ХХ І курсу*

*Гуменського В.Л.*

(Обсяг програми (документа), арк., Кб)

*182 арк, 124 Кб*

(Вид носія даних)

*GitHub репозиторій*

(Найменування програми (документа))

*Тексти програмного коду програмного забезпечення вирішення задачі обернення матриць*

Тут вставляється назва файлу та його вміст