МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського
Кафедра
“Інформатика та програмна інженерія”

КУРСОВА РОБОТА

з Основ Програмування - 2: Модульне програмування

(назва дисципліни)

на тему: Задача розміщення ферзів

Студента (ки, ів) 1 курсу, групи _ІП-11_
Тихонова Федора Сергійовича
Спеціальності 121 «Інженерія програмного забезпечення»
Керівник
(посада, вчене звання, науковий ступінь,
прізвище та ініціали)
Головченко Максим Миколайович

Кількість балів: _____

Національна оцінка _____

Члени комісії

_____ _____

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

Київ- 2022 рік

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

(назва вищого навчального закладу)

Кафедра інформатики та програмної інженерії

Дисципліна Основи програмування

Напрям "ІПЗ"

Курс 1 Група ІП-11 Семестр 2

ЗАВДАННЯ

на курсову роботу студента

Тихонова Федора Сергійовича

1. Тема роботи - Задача розміщення ферзів

2. Строк здачі студентом закінченої роботи - 12.06.2022

3. Вихідні дані до роботи

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

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

6. Дата видачі завдання 10.02.2022

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів курсової роботи	Термін виконання	Підписи керівника,
1.	Отримання теми курсової роботи	10.02.2022	
2.	Підготовка ТЗ	02.05.2022	
3.	Пошук та вивчення літератури з питань курсової роботи	03.05.2022	
4.	Розробка сценарію роботи програми	04.05.2022	
6.	Узгодження сценарію роботи програми з керівником	04.05.2022	
5.	Розробка (вибір) алгоритму рішення задачі	04.05.2022	
6.	Узгодження алгоритму з керівником	04.05.2022	
7.	Узгодження з керівником інтерфейсу користувача	05.05.2022	
8.	Розробка програмного забезпечення	06.05.2022	
9.	Налагодження розрахункової частини програми	06.05.2022	
10.	Розробка та налагодження інтерфейсної частини програми	07.05.2022	
11.	Узгодження з керівником набору тестів для контрольного прикладу	25.05.2022	
12.	Тестування програми	26.05.2022	
13.	Підготовка пояснювальної записки	05.06.2022	
14.	Здача курсової роботи на перевірку	12.06.2022	
15.	Захист курсової роботи	15.06.2022	

Студент Тихонов Ф.С.

Керівник Муха І. П. "" _____20__ р.

ЗМІСТ

SOLUTIONS.PY

АНОТАЦІЯ

Пояснювальна записка до курсової роботи: 63 сторінки, 7 рисунків, 9 таблиць, 3 посилання.

Об’єкт дослідження: задача розміщення ферзів.

Мета роботи: дослідження методів розв’язання задачі розміщення ферзів за допомогою алгоритмів пошуку графів.

Вивчено Рекурсивний Пошук по Першому Найкращому Співпадінню та A*. Приведені змістовні постановки задач, їх індивідуальні математичні моделі, а також описано детальний процес розв’язання кожної з них.

Виконана програмна реалізація задачі розміщення ферзів.

ВСТУП

Дана робота присвячена вивченню розробки програмного забезпечення з використанням парадигми ООП, і стосується написання програмної реалізації Задачі розміщення ферзів.. Задача полягає у графічному представленні шахової дошки та реалізації розставлення у відповідному порядку.

ПОСТАНОВКА ЗАДАЧІ

Дана шахова дошка розміром 8x8. На ній розставити не більше 8 ферзів. За умови, що на дошці стоїть 8 і тільки 8 ферзів, відкрити для користувача способи розв’язання задачі, а саме - розставити 8 ферзів таким чином, щоб жодна з них не била іншу. Відповідні методи розв’язання цієї задачі - це RBFS & A*.

Створити графічний інтерфейс, який представляє собою шахову дошку

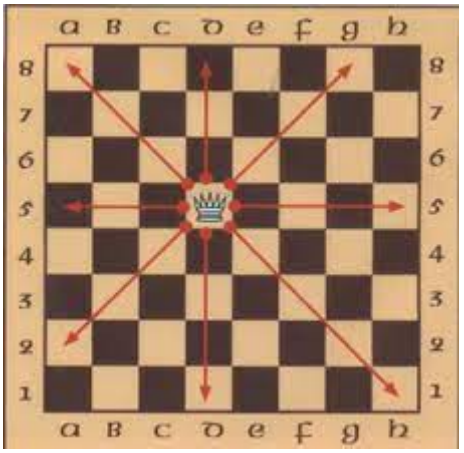
розміром 8x8 та матрицю активності - матрицю 8x8, заповнена нулями, де на деяких позиціях стоять одиниці, тобто ферзі. Вона необхідна для подальшої обробки самої себе з алгоритмами. Також створюються ще 2 кнопки - для розв’язання способом RBFS і A* відповідно.

Натисканням на будь-яке поле користувач може поставити ферзя, таким чином, програма заносить його позицію в “матрицю активності”. Відповідно, якщо натиснути на ферзя ще раз, він зникне з дошки та з матриці. Як тільки користувач розставив 8 ферзів будь-де на дошці, всі вільні поля перестають бути вільними для натискання, проте користувач може переставити існуючі ферзі. Для цього йому просто потрібно натиснути на потрібного ферзя та натиснути на необхідне поле.

ТЕОРЕТИЧНІ ВІДОМОСТІ

1) Означення задачі:

Задача розміщення ферзів - це проста, але нетривіальна задача перестановок. Головна мета - це розставити n ферзів на дошці розміром nxn клітинок. Але в нашій задачі ми візьмемо n = 8. Математично, варіантів перестановок 8 ферзів на дошці розміром 8x8 - це приблизно 4,4 млрд варіантів перестановок. Алгоритмами “грубої сили” було доведено, що з них - 92 є розв’язками. Як ходить ферзь продемонстровано на рисунку нижче.



2) Умови задачі:

Користувач має дошку 8x8, на якій він може розставляти ферзів. Натискаючи на пусту клітинку, він ставить туди ферзя. Відповідно, натискаючи на непусту клітинку, він цього ферзя знімає. Як тільки користувач розставив рівно 8 ферзів, всі вільні поля блокуються, тобто користувач не може поставити більше ферзів. Проте він може зняти існуючих ферзів та поставити їх на іншу клітинку.

Біля дошки, на якій користувач розставляє ферзів є 2 кнопки. Їх назви - це RBFS та A*. Ці кнопки відповідають за розв’язання відповідними алгоритмами. За замовчуванням, вони заблоковані, та

користувач не може на них натиснути. Але як тільки користувач розставив рівно 8 ферзів, то вони стають доступними для взаємодії. Якщо користувач зніме ферзя, то ці кнопки знову стануть недоступними.

Натиснувши на кнопку RBFS або A* програма створює нову дошку, на якій не можна нічого розставляти, але на якій розставлено правильним чином всіх ферзів. Дошки, які створюються, не накладаються одна на іншу, тобто можна порівняти результат обчислень.

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Основні змінні наведені у таблиці 1.

Таблиця 2.1 - Основні змінні програми

Змінна	Опис
_activityMatrix	Матриця, яка відображає розміщення ферзів на дошці. Заповнена 0 та 1. 1 - там, де стоїть 1 ферзь.
_tileList	Матриця, що складається з полів шахової дошки.
_solveRBFSButton	Кнопка, яка дає розв’язок алгоритмом RBFS.
_solveAStarButton	Кнопка, яка дає розв’язок алгоритмом A*.
row	Змінна, яка відповідає за номер рядка ($0 \leq row \leq 7$)
col	Змінна, яка відповідає за номер стовпця ($0 \leq col \leq 7$)
_activatedTiles	Змінна, яка відповідає за кількість

	розставлених ферзів (0 ≤ _activatedtiles ≤ 8)
colLst	Пронумерований у порядку зростання рядків, де стоять ферзі, список, який має елементи, які відповідають за номери колонок, де стоять ферзі.
directionLst	Пронумерований у порядку зростання рядків, де стоять ферзі, список, який має елементи, що відповідають за переміщення ферзя по рядку.

Алгоритми розв´язку.

a) RBFS(рядок, напрямлення, дошка, список_колонок)

ЯКЩО напрямлення == 1:

ТО ЦИКЛ для нин_колонка від список_колонок[рядок] до 7:

ЯКЩО нин_колонка == 7:

ТО напрямлення[рядок] = -1

ЯКЩО ферзь ставиться безпечно:

ТО список_колонок[рядок] = нин_колонка

Дошка[рядок][нин_колонка] = 1

ПОВЕРТАЄМО RBFS(рядок+1, напрямлення, дошка, список_колонок)

ІНАКШЕ ЦИКЛ для нин_колонка від список_колонок[рядок] до 0:

ЯКЩО ферзь ставиться безпечно:

ТО список_колонок[рядок] = нин_колонка

Дошка[рядок][нин_колонка] = 1

ПОВЕРТАЄМО RBFS(рядок+1, напрямлення, дошка, список_колонок)

напрямлення[рядок] = 1

список_колонок[рядок] = 0

ЯКШО напрямлення[рядок] == 1:

ТО список_колонок[рядок-1] += 1

ІНАКШЕ список_колонок[рядок-1] -= 1

ПОВЕРТАЄМО RBFS(рядок+1, напрямлення, дошка, список_колонок)

КІНЕЦЬ

б) AStar(дошка):

напрямлення = отримати_напрямлення(дошка)

список_матриць = []

ряд = 0

нова_матриця = створитиНульову()

список_колонок = отримати_колонки(дошка)

лічильникПершогоРядка = 0

ПОКИ лічильникПершогоРядка != 7:

поставлено = 0

ЯКЩО напрямлення == 1:

ТО ЦИКЛ для нин_колонка від список_колонок[рядок] до 7:

ЯКЩО нин_колонка == 7:

ТО напрямлення[рядок] = -1

ЯКЩО ферзь ставиться безпечно:

ТО список_колонок[рядок] = нин_колонка

Нова_матриця[рядок][нин_колонка] = 1

рядок += 1

поставлено = 1

Дошка[рядок][нин_колонка] = 1

РОЗІРВАТИ

ІНАКШЕ ЦИКЛ для нин_колонка від список_колонок[рядок] до 0:

ЯКЩО поставлено == 1:

ТО РОЗІРВАТИ

ЯКЩО ферзь ставиться безпечно:

ТО список_колонок[рядок] = нин_колонка

Нова_матриця[рядок][нин_колонка] = 1

рядок += 1

поставлено = 1

Дошка[рядок][нин_колонка] = 1

РОЗІРВАТИ

ЯКЩО посатвлено == 0:

ТО напрямлення[рядок] = 1

список_колонок[рядок] = 0

ЯКШО напрямлення[рядок] == 1:

ТО список_колонок[рядок-1] += 1

ІНАКШЕ список_колонок[рядок-1] -= 1

Рядок -= 1

ЯКЩО рядок == 8:

ТО ЯКЩО матриця НЕ Є В список_матриць ТА перевірка == 1:

ТО список_матриць.додати(дошка)

ЯКШО напрямлення[рядок] == 1:

ТО список_колонок[рядок-1] += 1

ІНАКШЕ список_колонок[рядок-1] -= 1

рядок -= 1

ЯКЩО всі варіанти проглянуті:

ТО ПОВЕРНУТИ список_матриць

КІНЕЦЬ

1. Сценарій роботи з програмою:

1. Користувач розставляє ферзів

2. Користувач натискає на кнопки розв’язань відповідними алгоритмами

3. Користувач може розставляти будь-які комбінації ферзів та розв’язувати для них задачу, поки не закриє програму.

АНАЛІЗ АРХІТЕКТУРИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Діаграма класів продемонстрована на рисунку 4.1

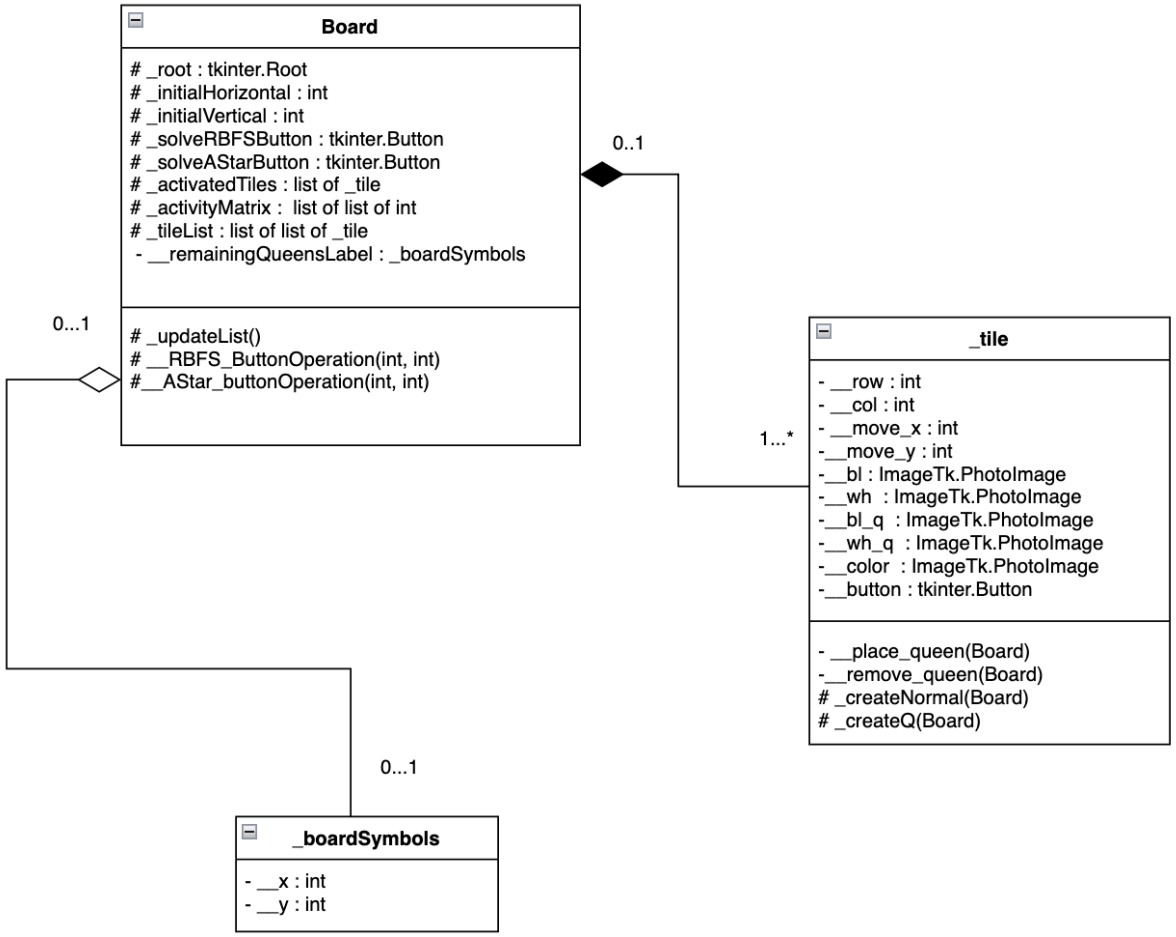


Рисунок 4.1 - Діаграма класів

Опис програмного забезпечення:

Таблиця 4.1 - Користувацькі методи

№ п/п	Назва класу	Назва методу	Призначення методу	Опис вхідних параметрів	Опис вихідних параметрів	Заголовний файл
1	board	__init__	Конструктор з параметрами для классу "дошка".	X: Переміщення по x y: Переміщення по y root: вікно	Немає	GUI.py
2	board	_updateList	Оновлює число ферзів, які залишилися	Немає	Немає	GUI.py
3	board	__RBFS_ButtonOperation	Розв’язує задачу за допомогою RBFS	x - зміщення по горизонталі, y - по вертикалі	Немає	GUI.py
4	board	__AStar_ButtonOperation	Розв’язує задачу за допомогою A*	x - зміщення по горизонталі, y - по вертикалі	Немає	GUI.py
5	_tile	__init__	Конструктор класу _tile	row - рядок, в якому ставиться поле col - колонка, в якій ставиться поле x - зміщення по	Немає	GUI.py

				горизонталі, y - по вертикалі someBoard - об’єкт класу “дошка”.		
6	_tile	__place_ queen	Ставить ферзя	someBoard - об’єкт класу “дошка”.	Немає	GUI.py
7	_tile	__remove_ queen	Прибираємо ферзя	someBoard - об’єкт класу “дошка”.	Немає	GUI.py
8	_tile	__create Normal	Створюємо неактивне пусте поле	someBoard - об’єкт класу “дошка”	Немає	GUI.py
9	_tile	__createQ	Створюємо неактивне поле з ферзем	someBoard - об’єкт класу “дошка”	Немає	GUI.py
10	_board Symb ols	__init__	Конструктор для створення символів біля дошки	x - зміщення по горизонталі, y - по вертикалі root - вікно	Немає	GUI.py

Таблиця 4.2 Стандартні методи

1	Tk	__init__	Створення вікна	Немає	Повертає об’єкт “вікно”	tkinter.py
2	Tk	geometry	Створення вікна відповідної величини	Рядок формату - {ширина}x{висота}	Немає	tkinter.py
3	Tk	title	Створення назви вікна	Рядок тексту	Немає	tkinter.py
4	Tk	mainloop	Зациклення вікна	Немає	Немає	tkinter.py
5	Button	__init__	Створення кнопки	text - надпис на кнопці image - картинка на кнопці command - функція, за яку відповідає кнопка	Повертає об’єкт “кнопка”	tkinter.py
6	Button	destroy	Видалення кнопки	немає	немає	tkinter.py
7	Button	place	Поставлення кнопки у конкретному положенні	x - зміщення по горизонталі y - зміщення по вертикалі	Немає	tkinter.py

8	PhotoImage	__init__	Створення об’єкту “зображення”	Файл	Об’єкт класу зображення	ImageTk
9	list	append	Додавання об’єкту до списку	Об’єкт	Немає	Python

ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Маємо основні частини програми, які необхідно протестувати:

1) Тестування розміщення ферзів(не більше 8)
2) Тестування прибирання ферзів
3) Тестування кнопок розв’язання
4) Тестування перестановок та розв’язання:
   4.1) Прибрати ферзів
   4.2) Поставити нову комбінацію ферзів
   4.3) Розв’язати для нової комбінації

Таблиця 5.1

Мета тесту	Тестування розміщення ферзів
Початковий стан програми	Програма відкрита, дошка пуста
Вхідні дані	Немає
Схема проведення тесту	Розставлення будь-якої кількості ферзів
Очікуваний результат	Після розміщення 8-го ферзя, програма не дасть можливості ставити більше. А кнопки алгоритмів повинні розблокуватися. Лічильник ферзів має оновитися кожен раз після поставлення ферзя.

Стан програми після проведення випробувань	Програма успішно спрацювала, після розміщення восьмого ферзя всі вільні поля заблоковані. Лічильник оновлюється.

Таблиця 5.2

Мета тесту	Тестування прибирання ферзів.
Початковий стан програми	Розміщено від 1 до 8 ферзів на дошці.
Вхідні дані	Немає
Схема проведення тесту	Маємо розстановку ферзів. Потрібно її прибрати.
Очікуваний результат	Пуста дошка. Якщо на дошці були 8 ферзів, то після прибирання першого ферзя програма має заблокувати кнопки розв’язання. Лічильник ферзів має оновитися кожен раз після прибирання ферзя.
Стан програми після проведення випробувань	Після прибирання ферзів на дошці, програма успішно видала пусту шахову дошку. У випадку, якщо на дошці 8 ферзів, прибирання першого ферзя

	блокує розв’язання. Лічильник оновлюється.

Таблиця 5.3

Мета тесту	Розв’язання обома алгоритмами.
Початковий стан програми	Програма відкрита і має певну розстановку 8 ферзів.
Вхідні дані	8 Розставлених ферзів.
Схема проведення тесту	Натискаємо на кнопки розв’язання, отримуємо 2 нові дошки з розв’язками задачі.
Очікуваний результат	2 нові дошки з розв’язками задачі.
Стан програми після проведення випробувань	Програма правильно видала розв’язки задачі.

Таблиця 5.4

Мета тесту	Переставити ферзів, отримати новий розв’язок.
Початковий стан програми	Маємо розстановку ферзів, та розв’язок для них.
Вхідні дані	Немає.

Схема проведення тесту	Прибираємо ферзів. Розставляємо їх так, як потрібно, розв’язуємо для них задачу.
Очікуваний результат	Дві нові дошки з розв’язками.
Стан програми після проведення випробувань	Програма успішно видала нові розв’язки для представленої комбінації.

ІНСТРУКЦІЯ КОРИСТУВАЧА

1) Запускаємо програму шляхом відкриття через середовище або через файл з росширенням app.

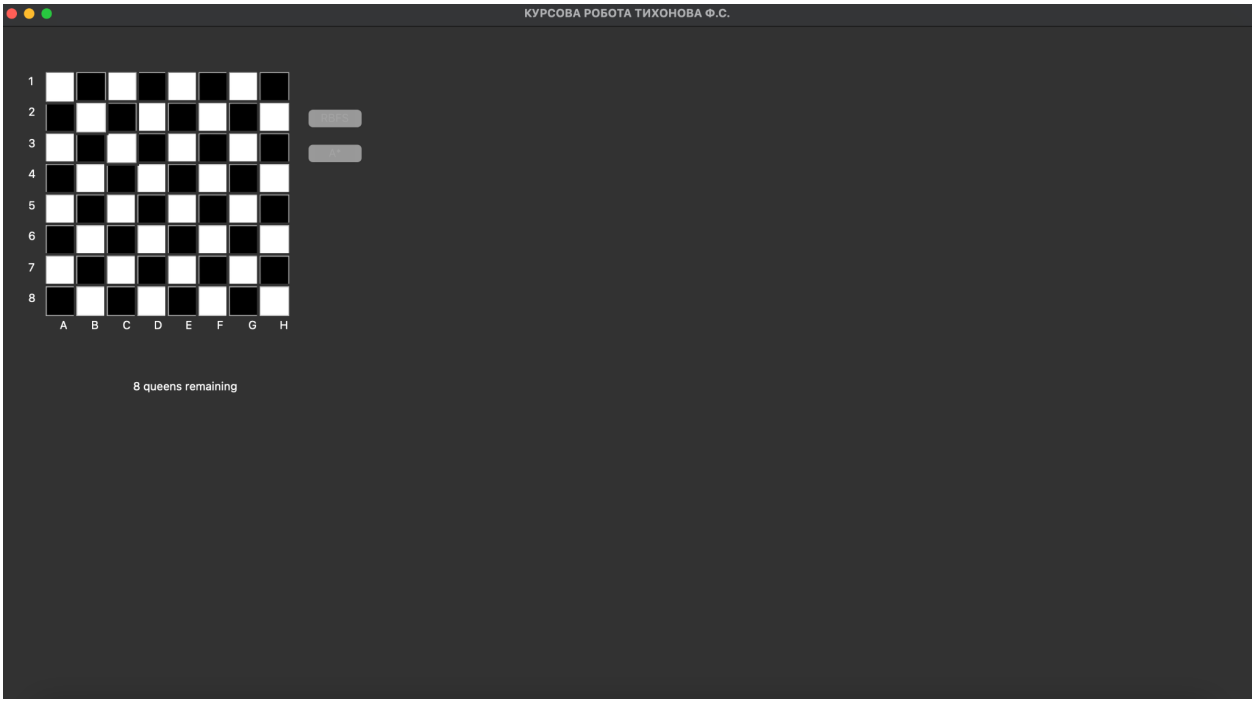


Рисунок 6.1 Головне вікно

2) Для того, щоб поставити ферзя, потрібно натиснути на відповідне поле лівою кнопкою миші.

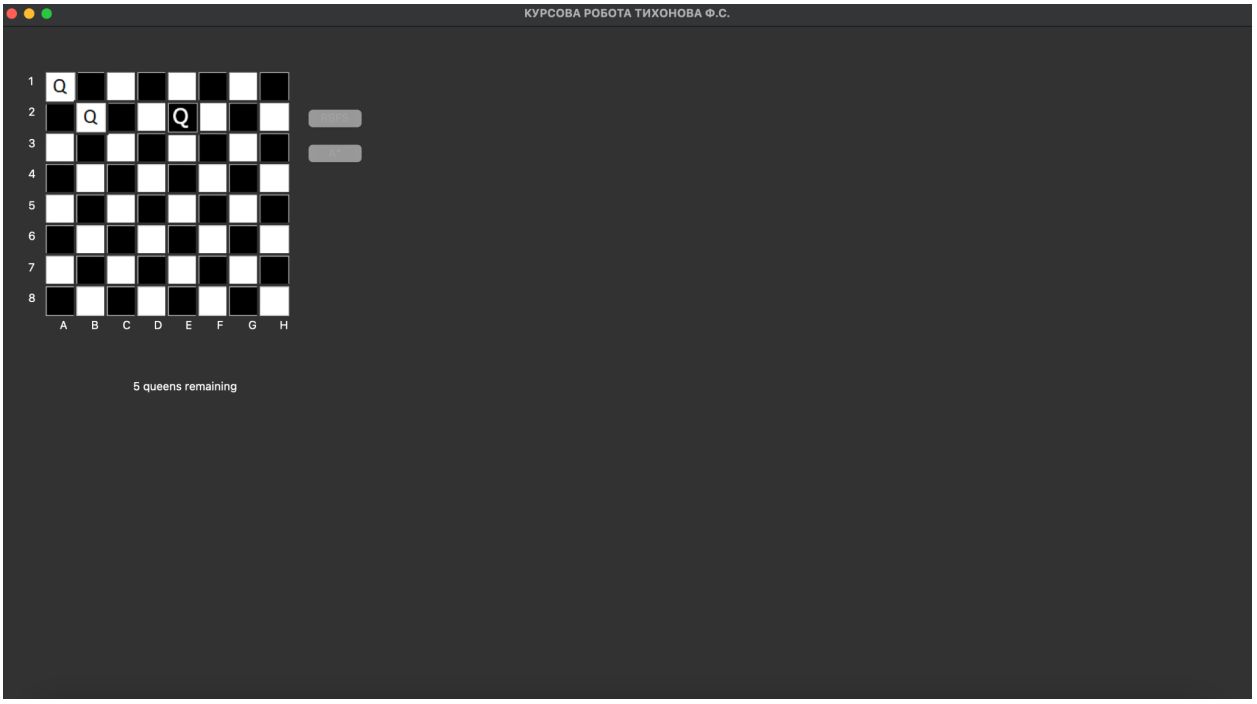


Рисунок 6.2 - Поставлені ферзі

3) Для того, щоб прибрати ферзя, треба натиснути по полю з ним лівою кнопкою миші.

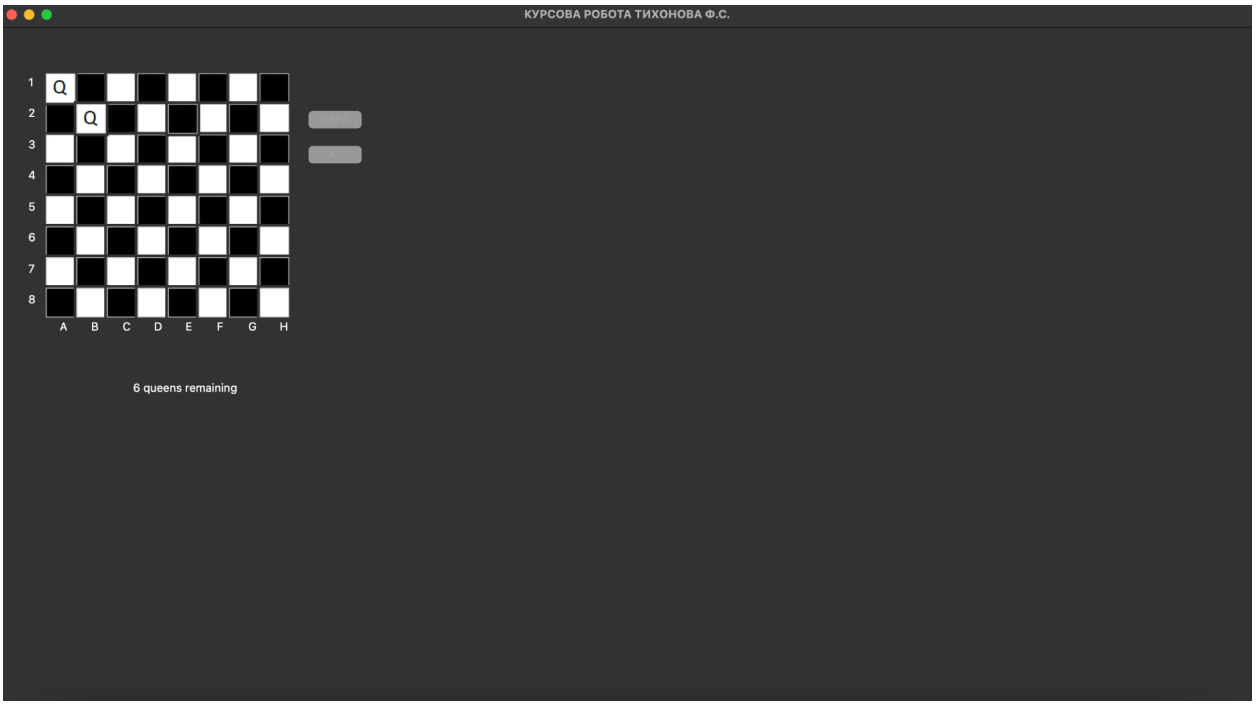


Рисунок 6.3 - прибрали одного ферзя

4) Користувач може розставити до 8 ферзів. Як тільки 8-й ферзь буде поставлений, більше не можна буде ставити, бо вільні поля будуть заблоковані.

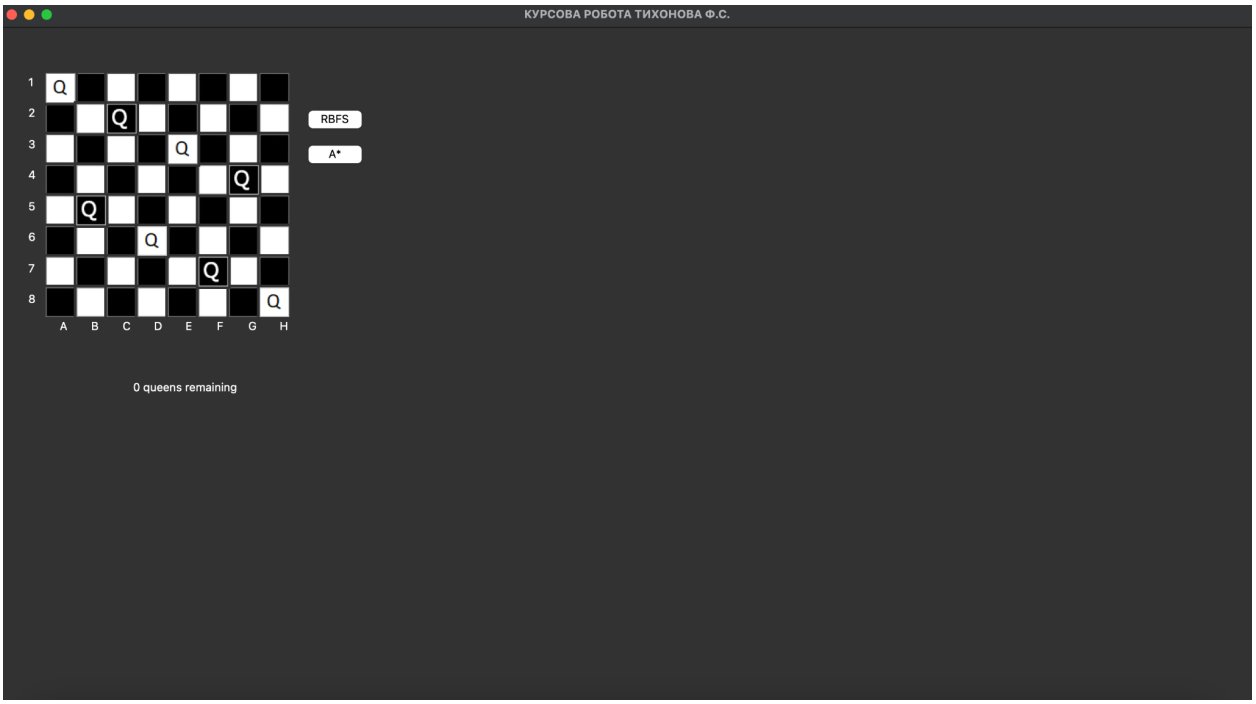


Рисунок 6.4 - Всі ферзі розставлені.

Хоч і більше ферзів ставити не можна, користувач може прибрати будь-якого ферзя і поставити його в інше місце.

5) Для розв’язання задачі потрібно поставити на дошку 8 і тільки 8 ферзів. Після 8 ферзя буде відкрито доступ до кнопок RBFS та A*. Для розв’язання потрібно натиснути на них. Після цього буде створено нове шахове поле, на якому буде розв’язок. На ній не можна буде нічого розставляти або прибирати.

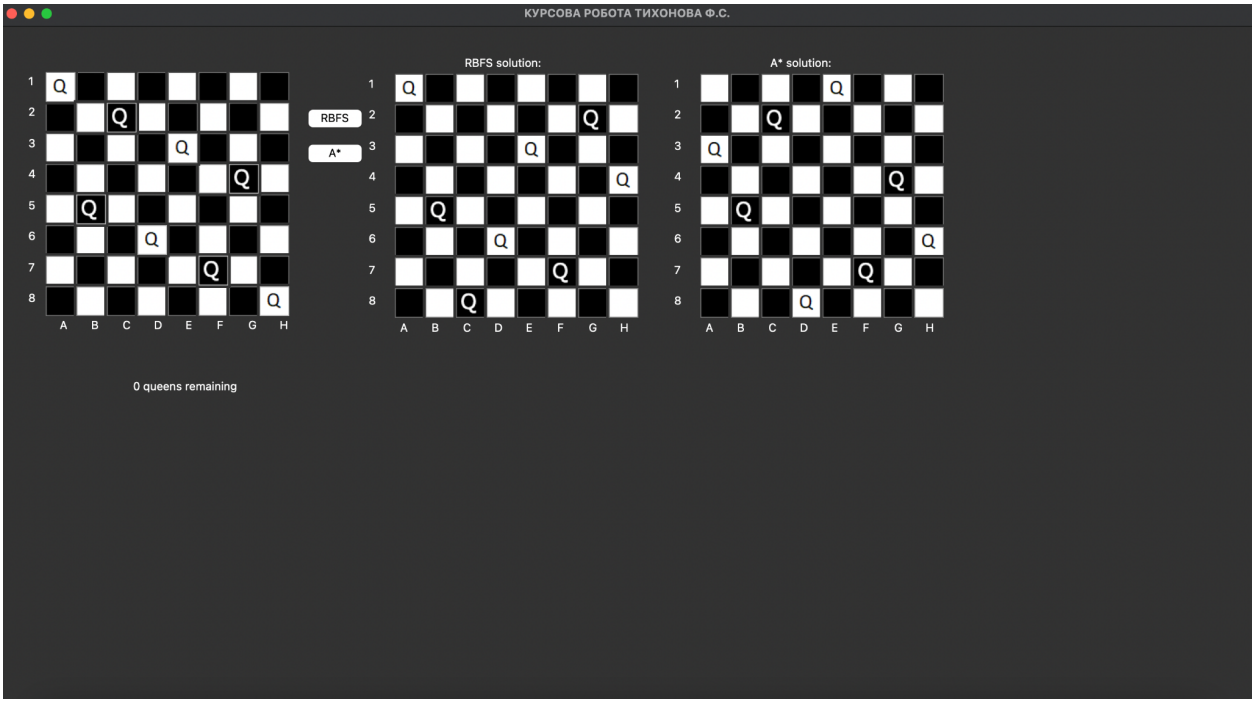


Рисунок 6.5 - Розв’язки задачі.

Таблиця 6.1 - Системні специфікації для роботи з програмою.

	Мінімальні	Рекомендовані
Операційна система	MacOS 10.11	MacOS 12.1+
Процесор	Intel® Pentium® III 1.0 GHz або AMD Athlon™ 1.0 GHz	Intel® Pentium® D або AMD Athlon™ 64 X2
Оперативна пам'ять	1 GB RAM	2 GB RAM
Відеоадаптер	Intel GMA 950 з відеопам'яттю об'ємом не менше 64 МБ (або сумісний аналог)	

Дисплей	1920x1080	1920x1080 або краще

АНАЛІЗ І УЗАГАЛЬНЕННЯ РЕЗУЛЬТАТІВ

Головною задачею курсової роботи була реалізація програми для розв’язання задачі розміщення ферзів наступними методами: RBFS та A*.

Критичні ситуації у роботі програми виявлені не були. Всі методи та кнопки були перевірені та відповідають дійсності та працюють корректно.

Зробимо порівняння різних розстановок ферзів:

Таблиця 7.1 - Аналіз кількості ітерацій в залежності від розстановок.

Варіант розстановки	Параметри тестування	Метод	
		RBFS	A*
Вертикально	Кількість ітерацій	2675029	16424184
	Кількість елементарних операцій (млн.)	1.7	70
Горизонтально	Кількість ітерацій	2771025	16433148
	Кількість елементарних операцій (млн.)	1.7	70
По діагоналі	Кількість ітерацій	2571021	16424184
	Кількість елементарних операцій (млн.)	1.7	70
Випадково	Кількість ітерацій	2475102	15832143
	Кількість елементарних операцій (млн.)	1.7	63

За результатами тестування можна зробити такі висновки:

1) Всі методи працюють задовільно незалежно від розстановки ферзів

2) Хоч і RBFS швидше, A* підбирає таку розстановку, яка є найближчою до заданої матриці.

ВИСНОВОК

Отже, під час написання курсової роботи на тему “Задача розміщення ферзів”, я навчився використовувати концепти ООП. Також я навчився створювати графічний інтерфейс, спроектував архітектуру програми. Були продемонстровані теоретичні відомості, описи, діаграми, тестування, інструкції.

ПЕРЕЛІК ПОСИЛАНЬ

Задача розміщення ферзів - вікі

A*

RBFS

ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра
інформатики та програмної інженерії

Затвердив

Керівник Головченко М.М.

«___»_____2022 р.

Виконавець:

Студент Тихонов Ф. С.

«___» _____2022 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання курсової роботи

на тему: “Задача розміщення ферзів”

з дисципліни:

«Основи програмування»

Київ 2022

1. *Мета*: Метою курсової роботи є розробка програми яка розв'язує задачу розміщення ферзів алгоритмами

A*, RBFS з початковою розстановкою, яку задає користувач.

2. *Дата початку роботи*: «2» травня 2022 р.
3. *Дата закінчення роботи*: «___»_____ 202_ р.
4. *Вимоги до програмного забезпечення*.

1) Функціональні вимоги:
   - Відображення шахового поля(8x8) графічним шляхом
   - Можливість розміщувати ферзі на шаховому полі
   - Можливість перевірки введених даних
   - Можливість обирати алгоритм розв’язання
   - Можливість зберігання даних у текстовому файлі
   - Відображення роботи алгоритму
2) Нефункціональні вимоги:
   - Можливість запускати програму на macOS Monterey 12.3.1
   - Все програмне забезпечення та супроводжуюча технічна документація повинні задовольняти наступним ДЕСТам:

ГОСТ 29.401 - 78 - Текст програми. Вимоги до змісту та оформлення.

ГОСТ 19.106 - 78 - Вимоги до програмної документації.

ГОСТ 7.1 - 84 та ДСТУ 3008 - 2015 - Розробка технічної документації.

5. *Стадії та етапи розробки*:

1) Об'єктно-орієнтований аналіз предметної області задачі (до__.__.202_ р.)
2) Об'єктно-орієнтоване проектування архітектури програмної системи (до __.__.202_р.)
3) Розробка програмного забезпечення (до __.__.202_р.)
4) Тестування розробленої програми (до __.__.202_р.)
5) Розробка пояснювальної записки (до __.__.202_ р.).
6) Захист курсової роботи (до __.__.202_ р.).

6. *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

Тексти програмного коду “Задачі розміщення ферзів”

courseWork

Github

Вид носія даних

Обсяг 30 сторінок, 47 кб

COURSE.PY

```
import GUI


if __name__ == '__main__':
    GUI.main()
```

GUI.PY

```
import tkinter as tk
from PIL import ImageTk, Image
import GenericMatrixOperations as GMO
import Solutions



class board:
    """
    Клас "дошка". Відповідає за графічне зображення та функціональність дошки.
    """
    def __init__(self, x, y, root):
        """
```

Конструктор з параметрами для классу "дошка". В списку параметрів передаються 3 параметри:

:param x: Як далеко перемістити дошку по горизонталі (в пікселях)

:param y: Як далеко перемістити дошку по вертикалі (в пікселях)

:param root: Вікно, в якому розміщується дошка

Ініціалізує дошку як об'єкт, разом з матрицею, початковою кількістю ферзів та кнопками.

"""

```
self._root = root
self._initialHorizontal = x
self._solveRBFSButton = tk.Button(text='RBFS', command=lambda: self.__RBFS_ButtonOperation(450, 52))
self._solveRBFSButton.place(x=x + 300, y=y + 40)
self._solveRBFSButton["state"] = "disable"
self._solveAStarButton = tk.Button(text='A*', command=lambda: self.__AStar_buttonOperation(800, 52))
self._solveAStarButton.place(x=x + 300, y=y + 80, width=69)
self._solveAStarButton["state"] = "disable"
self._InitialVertical = y
self._activatedTiles = []
self._activityMatrix = GMO.createEmpty()
board._boardSymbols(30, 30, root)
self._tileList = []
for row in range(0, 8):
    self._tileList.append([])
    for col in range(0, 8):
        t = board._tile(row, col, x, y, self)
```

```
            self._tileList[row].append(t)
        self.__remainingQueensLabel = tk.Label(text=f'{8 -
len(self._activatedTiles)} queens remaining')
        self.__remainingQueensLabel.place(x=self._initialHorizontal + 100,
y=self._InitialVertical + 350)


    def _updateList(self):
        """
        Оновлює лічильник ферзів.
        """
        self.__remainingQueensLabel = tk.Label(text=f'{8 -
len(self._activatedTiles)} queens remaining')
        self.__remainingQueensLabel.place(x=self._initialHorizontal + 100,
y=self._InitialVertical + 350)


    def __RBFS_ButtonOperation(self, x, y):
        """
        Розв'язує задачу розміщення ферзів за допомогою RBFS-алгоритму, а
саме:
        1) Створює нову шахову дошку.
        2) Оброблює матрицю згідно з алгоритмом.
        3) Правильним чином розставляє ферзів на дошці, робить всі поля
недоступними для натискання.

        :param x: відстань від дошки відносно головної дошки по горизонталі
(у пікселях)
        :param y: відстань від дошки відносно головної дошки по вертикалі (у
пікселях)
        """
        label = tk.Label(text="RBFS solution:")
```

```
        label.place(y=y-22, x=x+80)
        matrix = GMO.copy(self._activityMatrix)
        if GMO.Check(matrix) is True:
            matrix = matrix
        else:
            matrix = Solutions.RBFS_solution(matrix)
        board._boardSymbols(self._initialHorizontal + x - 80, self._InitialVertical -
y + 35, self._root)
        for row in range(0, 8):
            for col in range(0, 8):
                tmp = board._tile(row, col, x, y, self)
                if matrix[row][col] == 1:
                    tmp._createQ(self)
                else:
                    tmp._createNormal(self)
        matrix.clear()


    def __AStar_buttonOperation(self, x, y):
        """
        Розв'язує задачу розміщення ферзів за допомогою A*-алгоритму, а
саме:
        1) Створює нову шахову дошку.
        2) Оброблює матрицю згідно з алгоритмом.
        3) Правильним чином розставляє ферзів на дошці, робить всі поля
недоступними для натискання.

        :param x: відстань від дошки відносно головної дошки по горизонталі
(у пікселях)
```

:param y: відстань від дошки відносно головної дошки по вертикалі (у пікселях)

"""

label = tk.Label(text='A* solution:')

label.place(x=x+80, y=y-22)

matrix = GMO.copy(self._activityMatrix)

if GMO.Check(matrix) is True:

matrix = matrix

else:

matrix = Solutions.AStarSolution(matrix)

for i in range(0, 8):

print(matrix[i])

board._boardSymbols(self._initialHorizontal + x - 80, self._InitialVertical - y + 35, self._root)

for row in range(0, 8):

for col in range(0, 8):

tmp = board._tile(row, col, x, y, self)

if matrix[row][col] == 1:

tmp._createQ(self)

else:

tmp._createNormal(self)

matrix.clear()

class _boardSymbols:

"""

Клас "символи на дошці". Не несе в собі функціональності,

тільки розставляє символи біля дошки для того, щоб користувачу було легше

зрозуміти положення ферзя.

```
        """
        def __init__(self, x, y, root):
            """
            Конструктор з параметрами. Створює символи біля дошки.
            :param x: Здвиг по горизонталі(якщо дошка теж зміщена по горизонталі).
            :param y: Здвиг по вертикалі(якщо дошка теж зміщена по вертикалі).
            :param root: Вікно, в якому існує дошка.
            """
            self.__x = x
            self.__y = y
            for c in range(1, 9):
                _label = tk.Label(root, text=f'{c}')
                _label.place(x=self.__x, y=self.__y + c * 35.5 - 15)
            for c in range(1, 9):
                _label = tk.Label(root, text=f'{chr(c + 64)}')
                _label.place(x=self.__x + c * 36, y=self.__y + 300)


    class _tile:
        """
        Клас "поле". Відповідає за графічне зображення поля на дошці. Графічно є кнопкою,
        натиснувши на яку можна прибрати або поставити ферзя.
        """
        def __init__(self, row, col, move_x, move_y, someBoard):
            """
            Конструктор з параметрами для класу "поле". Приймає наступні параметри:
```

:param row: ряд, в якому знаходиться поле.

:param col: колонка, в якій знаходиться поле.

:param move_x: зміщення колонки в горизонталі (якщо дошка зміщена, то й поле теж).

:param move_y: зміщення колонки в вертикалі (якщо дошка зміщена, то й поле теж).

:param someBoard: об'єкт "дошка", на якій ми ставимо поле.

"""

```
self.__row = row
self.__col = col
self.__move_x = move_x
self.__move_y = move_y
self.__bl = ImageTk.PhotoImage(Image.open("Black.png"))
self.__wh = ImageTk.PhotoImage(Image.open("White.png"))
self.__bl_q = ImageTk.PhotoImage(Image.open("BlackWithQ.png"))
self.__wh_q = ImageTk.PhotoImage(Image.open("WhiteWithQ.png"))
if row % 2 == 1:
    if col % 2 == 1:
        self.__color = self.__wh
    else:
        self.__color = self.__bl
else:
    if col % 2 == 1:
        self.__color = self.__bl
    else:
        self.__color = self.__wh
self.__button = tk.Button(someBoard._root, image=self.__color,
command=lambda: self.__place_queen(someBoard))
self.__button.place(x=35 * row + move_x, y=35 * col + move_y)
```

```
    def __place_queen(self, someBoard):
        """
        Метод, яким ми ставимо ферзя на поле. Змінює вид поля графічно та ставить одиницю в матриці активності.
        :param someBoard: дошка, на якій ми ставимо ферзя.
        """
        someBoard._activityMatrix[self.__row][self.__col] = 1
        self.__button.destroy()
        if self.__color == self.__wh:
            self.__color = self.__wh_q
        else:
            self.__color = self.__bl_q
        self.__button = tk.Button(someBoard._root, image=self.__color, command=lambda: self.__remove_queen(someBoard))
        self.__button.place(x=35 * self.__row + self.__move_x, y=35 * self.__col + self.__move_y)
        someBoard._activatedTiles.append(self)
        someBoard._updateList()
        if len(someBoard._activatedTiles) >= 8:
            for d in range(0, 8):
                for j in range(0, 8):
                    someBoard._tileList[d][j].__button["state"] = "disable"
            for d in range(len(someBoard._activatedTiles)):
                someBoard._activatedTiles[d].__button["state"] = "normal"
            someBoard._solveRBFSButton["state"] = "normal"
            someBoard._solveAStarButton["state"] = "normal"


    def _createQ(self, someBoard):
```

```
        """
```

Створюємо поле, на якому стоїть ферзь, і якого не можна прибрати.

:param someBoard: шахова дошка, на якій ми бажаємо створити поле та поставити ферзя.

```
        """
        self.__bl_q = ImageTk.PhotoImage(Image.open("BlackWithQ.png"))
        self.__wh_q = ImageTk.PhotoImage(Image.open("WhiteWithQ.png"))
        self.__button.destroy()
        if self.__color == self.__wh:
            self.__color = self.__wh_q
        else:
            self.__color = self.__bl_q
        self.__button = tk.Button(someBoard._root, image=self.__color,
command=lambda: self.__place_queen(someBoard))
        self.__button.place(x=35 * self.__row + self.__move_x, y=35 *
self.__col + self.__move_y)
        self.__button["state"] = "disable"

    def _createNormal(self, someBoard):
        """
```

Створюємо пусте поле, на яке не можна поставити ферзя.

:param someBoard: шахова дошка, на якій ми бажаємо створити поле.

```
        """
        self.__bl_q = ImageTk.PhotoImage(Image.open("BlackWithQ.png"))
        self.__wh_q = ImageTk.PhotoImage(Image.open("WhiteWithQ.png"))
        self.__button.destroy()
        self.__button = tk.Button(someBoard._root, image=self.__color,
command=lambda: self.__place_queen(someBoard))
```

```
        self.__button.place(x=35 * self.__row + self.__move_x, y=35 *
self.__col + self.__move_y)
        self.__button["state"] = "disable"


    def __remove_queen(self, someBoard):
        """
        Прибираємо ферзя з певного поля. На його місці в матриці
активності ставимо нуль.
        :param someBoard: дошка, з якої ми прибираємо ферзя.
        """
        someBoard._activityMatrix[self.__row][self.__col] = 0
        self.__button.destroy()
        if self.__color == self.__bl_q:
            self.__color = self.__bl
        else:
            self.__color = self.__wh
        newButton = tk.Button(someBoard._root, image=self.__color,
command=lambda: self.__place_queen(someBoard))
        newButton.place(x=35 * self.__row + self.__move_x, y=35 * self.__col
+ self.__move_y)
        self.__button = newButton
        someBoard._activatedTiles.remove(self)
        someBoard._updateList()
        if len(someBoard._activatedTiles) < 8:
            for k in range(0, 8):
                for j in range(0, 8):
                    someBoard._tileList[k][j].__button["state"] = "normal"
            someBoard._solveRBFSButton["state"] = "disable"
            someBoard._solveAStarButton["state"] = "disable"
```

```
def main():
    """
    Основна програма. Створює об'єкт класу "дошка", на якому
виконуються всі операції.
    """
    root = tk.Tk()
    root.title("КУРСОВА РОБОТА ТИХОНОВА Ф.С.")
    root.geometry('1920x1080')
    board(50, 50, root)
    root.mainloop()
```

GENERICMATRIXOPERATIONS.PY

```
import random


def transpose(matrix):
    """
    Транспонування матриці.
    :param matrix: матриця 8x8.
    :return: транспонована матриця 8x8.
    """
    newMatrix = []
    for i in range(0, 8):
        newMatrix.append([0, 0, 0, 0, 0, 0, 0, 0])
    for i in range(0, 8):
        for j in range(0, 8):
            newMatrix[i][j] = matrix[j][i]
```

```
    return newMatrix


def isPresent(lst):
    """
    Перевірка, чи є в списку одиниця.
    :param lst: список на 8 елементів.
    :return: 0/1
    """
    for i in range(0, 8):
        if lst[i] == 1:
            return False
    return True


def getListOfFreeRows(matrix):
    """
    Повертає список індексів рядків, де немає одиниць, тобто, вони повністю заповнені нулями.
    :param matrix: матриця 8x8.
    :return: список індексів рядків.
    """
    lst = []
    for i in range(0, 8):
        if isPresent(matrix[i]) is True:
            lst.append(i)
    return lst
```

```
def createEmpty():
    """
    Створює матрицю 8x8, повністю заповнену нулями.
    :return: матрицю 8x8, повністю заповнена нулями
    """
    newMatrix = []
    for row in range(0, 8):
        newMatrix.append([])
    for rowID in range(0, 8):
        for colID in range(0, 8):
            newMatrix[rowID].append(0)
    return newMatrix


def eraseRow(matrix, row):
    """
    Заповнює певний рядок матриці нулями.
    :param matrix: матриця 8x8
    :param row: рядок (від нуля до семи включно)
    """
    for i in range(0, 8):
        matrix[row][i] = 0


def createRandom():
    RandomMatrix = []
    for i in range(0, 8):
```

```
        RandomMatrix.append([0, 0, 0, 0, 0, 0, 0, 0])
    counter = 0
    while counter != 8:
        RandomX = random.randint(0, 7)
        RandomY = random.randint(0, 7)
        if RandomMatrix[RandomX][RandomY] != 1:
            RandomMatrix[RandomX][RandomY] = 1
            counter += 1
    return RandomMatrix


def getCoords(matrix):
    """
    Повертає список координат віх одиниць в матриці.
    :param matrix: матриця 8x8
    :return: повертає матрицю на 8 списків,
    елементами яких на першій позицій стоїть рядок, а на другій стовпчик.
    """
    lst = []
    for i in range(0, 8):
        for j in range(0, 8):
            if matrix[i][j] == 1:
                lst.append([i, j])
    return lst


def placeQueensOnDifferentRows(matrix):
    """
```

```
    Якщо в матриці декілька одиниць на одному рядку, то функція
розставляє їх на вільні рядки.
    :param matrix: матриця 8x8, у якій 8 одиниць, а інші - нулі.
    :return: матриця 8x8, у якої всі одиниці розставлені на різних рядках.
    """
    freeLst = getListOfFreeRows(matrix)
    if len(freeLst) == 0:
        return matrix
    else:
        print(freeLst)
        counter = 0
        lstCounter = 0
        for i in range(0, 8):
            for j in range(0, 8):
                if counter == 0 and matrix[i][j] == 1:
                    counter = 1
                elif counter > 0 and matrix[i][j] == 1:
                    matrix[i][j] = 0
                    matrix[freeLst[lstCounter]][j] = 1
                    lstCounter += 1
                if j == 7:
                    counter = 0
        return matrix


def Check(matrix):
    """
    Перевірка, чи є в матриці хоч одна пара одиниць,
```

```
    які знаходяться по діагоналі, по вертикалі
    або по горизонталі одна від іншої.
    :param matrix: матриця 8x8.
    :return: 0/1
    """
    lst = getCoords(matrix)
    for anItem in lst:
        for item in lst:
            if item != anItem:
                if anItem[0] == item[0]:
                    return False
                if anItem[0] - item[0] == anItem[1] - item[1]:
                    return False
                if anItem[0] + anItem[1] == item[0] + item[1]:
                    return False
                if anItem[1] == item[1]:
                    return False
    return True


def movement(colLst):
    """
    Список, який ми передаємо як параметр - це список індексів колонок одиниць в матриці,
    пронумеровані відповідно до рядка. тобто, в якщо 5 елемент в списку має значення 4,
    то це означає, що в 5 рядку в 4 колонці стоїть одиниця.
    Якщо колонка дорівнює семи, то заносимо в список -1. Інакше заносимо 1.
```

```
    :param colLst: список колонок одиниць матриці.
    :return: список з 8 елементів, які є 1 або -1.
    """
    directionList = []
    for num in colLst:
        if num == 7:
            directionList.append(-1)
        else:
            directionList.append(1)
    return directionList


def findCol(matrix, row):
    """
    Повертає номер колонки, в якій знаходиться одиниця в рядку певної матриці.
    :param matrix: матриця 8x8.
    :param row: рядок довжиною 8.
    :return:
    """
    for i in range(0, 8):
        if matrix[row][i] == 1:
            return i


def getCols(coordsLst):
    """
    Повертає список колонок рядків.
```

```
    :param coordsLst: список координатів одиниць.
    :return: список колонок.
    """
    newLst = []
    for item in coordsLst:
        newLst.append(item[1])
    return newLst


def copy(matrix):
    """
    Копіює матрицю.
    :param matrix: матриця 8x8.
    :return: копійована матриця 8x8.
    """
    m = []
    for i in range(0, 8):
        m.append([0, 0, 0, 0, 0, 0, 0, 0])
    for i in range(0, 8):
        for j in range(0, 8):
            m[i][j] = matrix[i][j]
    return m
```

SOLUTIONS.PY

```
import GenericMatrixOperations as GMO
import sys
import math

sys.setrecursionlimit(10000)
```

```
def isSafe(matrix, row, col):
    """
    Перевірка, чи б'є ферзь будь-якого іншого ферзя на дошці:
    :param matrix: матриця, тобто відображення дошки.
    :param row: рядок ферзя.
    :param col: колонка ферзя.
    :return: 0/1.
    """
    coordsLst = GMO.getCoords(matrix)
    for item in coordsLst:
        if not (item[0] == row and item[1] == col):
            if row == item[0]:
                return False
            if row - item[0] == col - item[1]:
                return False
            if row + col == item[0] + item[1]:
                return False
            if col == item[1]:
                return False
    return True


def AStar(SomeMatrix):
    """
    Метод розв'язання задачі розміщення ферзів за допомогою методу A*.
```

Алгоритм полягає у знаходженні будь-яких розв'язків задачі, які "здається" ведуть

до розв'язку.

Принцип роботи алгоритму:

1) розв'язок починається з того, що змінній row, тобто, ряд, присвоюється значення 0.

2) на відповідний ряд, за який відповідає змінна row, ставиться ферзь, якщо це є безпечним.

3) Якщо весь ряд не є безпечним, то змінюємо позицію ферзя на минулому ряду.

4) Якщо матриця повна, то її додаємо в список матриць.

Вибір колонки працює наступним чином:

Початкова колонка - це колонка матриці, де початково стояв ферзь.

Евристичною функцією є функція pickClosest(), в принципі - це евклідова евристика.

Якщо в даній колонці ферзь б'ється будь-яким іншим ферзем, то ставимо його у колонку справа від нього.

Так йдемо, поки не дійшли або до безпечного поля, або доки не дійшли до крайнього поля справа.

У другому випадку, ми йдемо в іншу сторону, тобто, якщо початкова колонка ферзя - 4, то спочатку йдемо

від 4 до 8, а потім від 4 до 0.

Якщо ферзь стоїть безпечно, то ми змінюємо його номер колонки, ставимо ферзя на відповідній позиції та йдемо далі.

Інакше, якщо весь ряд не є безпечним, то заповнюємо ряд нулями, змінюємо колонку минулого ряда на наступний та

переміщаємося на ряд назад.

:param SomeMatrix: матриця з ферзями, кожний з яких стоїть на окремому рядку.

```
    :return: список розв'язків, тобто, матриць з правильно розставленими
ферзями.
    """
    newMatrix = GMO.createEmpty()
    matrixLst = []
    row = 0
    directionList = GMO.movement(GMO.getCoords(SomeMatrix))
    colList = GMO.getCols(GMO.getCoords(SomeMatrix))
    firstRowCounter = 0
    while firstRowCounter != 7:
        placed = 0
        if directionList[row] == 1:  # Рухаємося до правого краю дошки
            for tmpCol in range(colList[row], 8):
                if tmpCol == 7:
                    directionList[row] = -1
                newMatrix[row][tmpCol] = 1
                if isSafe(newMatrix, row, tmpCol) is True:
                    colList[row] = tmpCol
                    row += 1
                    placed = 1
                    break
                else:
                    newMatrix[row][tmpCol] = 0
        if row != 8:
            for tmpCol in range(colList[row] - 1, -1, -1):  # Рухаємося до лівого
краю дошки
                if placed == 1:
                    break
                newMatrix[row][tmpCol] = 1
```

```
            if isSafe(newMatrix, row, tmpCol) is True:
                colList[row] = tmpCol
                placed = 1
                row += 1
                break
            else:
                newMatrix[row][tmpCol] = 0
        if placed != 1:  # Ідемо на рядок назад
            if directionList[row - 1] == 1:
                colList[row - 1] += 1
            else:
                colList[row - 1] -= 1
            GMO.eraseRow(newMatrix, row)
            colList[row] = 0
            directionList[row] = 1
            GMO.eraseRow(newMatrix, row - 1)
            row -= 1
    if row == 8:  # Додаємо матрицю до списку потенційних розв'язків
        if GMO.Check(newMatrix) is True and newMatrix not in matrixLst:
            matrixLst.append(GMO.copy(newMatrix))
        GMO.eraseRow(newMatrix, 7)
        if directionList[7] == 1:
            colList[7] += 1
        else:
            colList[7] -= 1
        row -= 1
        if directionList[0] == -1 and colList[0] == 1:
            return matrixLst
```

```
def pickClosest(currMatrix, matrices):
    """
    Відбирає матрицю,"найближчу" до початкової.
    :param currMatrix: початкова матриця, до якої потрібно знайти найближчу матрицю
    :param matrices: список матриць, з яких шукаємо найближчу
    :return: найближча матриця.
    """

    def countDifference(oneMatrix, anotherMatrix):
        """
        Порівняння двох конкретних матриць, тобто, знаходження відстані між ферзями однієї та
        іншої матриць. Функція порівнює відстані між відповідними ферзями зі списку координат
        ферзів, який вона отримує за допомогою функції getCoords()
        :param oneMatrix: матриця 8x8
        :param anotherMatrix: інша матриця 8x8
        :return: сумарна відстань між всіма ферзями.
        """
        Sum = 0
        lstOfGiven = GMO.getCoords(oneMatrix)
        lstOfSolved = GMO.getCoords(anotherMatrix)
        for queenNum in range(0, 8):
            Sum += math.sqrt((lstOfSolved[queenNum][0] -
lstOfGiven[queenNum][0]) ** 2 +
```

```
                    (lstOfSolved[queenNum][1] - lstOfGiven[queenNum][1]) **
2)
        return Sum


    smallest = countDifference(currMatrix, matrices[0])
    closest = matrices[0]
    for comparedMatrix in matrices:
        if countDifference(currMatrix, comparedMatrix) < smallest:
            smallest = countDifference(currMatrix, comparedMatrix)
            closest = comparedMatrix
    return closest



def AStarSolution(matrix):
    """
    Послідовність функцій, яка дає розв'язок задачі розміщення ферзів для
дошки 8x8 методом A*.
    :param matrix: матриця 8x8, в якій є 8 одиниць та інші елементи - нулі.
    :return: матриця-розв'язок.
    """
    newMatrix = GMO.copy(matrix)
    newMatrix = GMO.placeQueensOnDifferentRows(matrix)
    solvedMatrix = pickClosest(matrix, AStar(newMatrix))
    return solvedMatrix



def RBFS(newMatrix, colList, direction, row):
    """
    Функція розв'язання задачі розміщення ферзів алгоритмом RBFS.
```

```
    (Рекурсивний пошук по першому найкращому співпадінню.)
    Працює схожим чином на A*. За виключенням, що рекурсія - це головний
    елемент цієї функції. Один з параметрів, який ми передаємо -
    це номер рядка, в якому ми знаходимося.
    За допомогою нього, та ряду інших змінних можемо переміщатися з
    рядка на рядок і правильним чином розставляти ферзів.
    :param newMatrix: матриця, в якій ми розставляємо ферзів. Важливо, на
початку - це пуста матриця,
    тобто, заповнена нулями.
    :param colList: список колонок де знаходяться ферзі. Номери розставлені
    відповідно до рядків.
    :param direction: список чисел, за допомогою яких ми можемо зрозуміти,
в яку сторону нам
    потрібно рухати ферзів.
    :param row: рядок, в якому ми зараз знаходимося.
    :return: матриця-розв'язок.
    """
    if row >= 8:
        return newMatrix
    else:
        if direction[row] == 1:  # Рухаємося до правого краю дошки
            for column in range(colList[row], 8):
                if column == 7:
                    direction[row] = -1
                newMatrix[row][column] = 1
                if isSafe(newMatrix, row, column) is True:
                    colList[row] = column
                    return RBFS(newMatrix, colList, direction, row + 1)
                else:
```

```
                newMatrix[row][column] = 0
        for column in range(colList[row] - 1, -1, -1):  # Рухаємося до лівого краю дошки
            newMatrix[row][column] = 1
            if isSafe(newMatrix, row, column):
                colList[row] = column
                return RBFS(newMatrix, colList, direction, row + 1)
            newMatrix[row][column] = 0
        if direction[row - 1] == 1:  # Ідемо на рядок назад
            colList[row - 1] += 1
        else:
            colList[row - 1] -= 1
        GMO.eraseRow(newMatrix, row)
        colList[row] = 0
        direction[row] = 1
        GMO.eraseRow(newMatrix, row - 1)
        return RBFS(newMatrix, colList, direction, row - 1)


def RBFS_solution(matrix):
    """
    Послідовність функцій, необхідна для розв'язання задачі розміщення ферзів
    шляхом RBFS.
    :param матриця 8x8, в якій є 8 одиниць та інші елементи - нулі.
    :return: матриця-розв'язок.
    """
    matrix = GMO.placeQueensOnDifferentRows(matrix)
    colLst = GMO.getCols(GMO.getCoords(matrix))
```

```
directionLst = GMO.movement(colLst)
newMatrix = GMO.createEmpty()
matrix = RBFS(newMatrix, colLst, directionLst, 0)
return matrix
```