**Інститут прикладної математики та фундаментальних наук**

**Кафедра прикладної математики**

**Звіт**

**до лабораторної роботи №1**

**з дисципліни**

**“Математичні основи штучного інтелекту”**

**студента групи ПМ-31**

**Подубінського Назара**

**Прийняв:**

**Пабирівський В.В.**

**Львів 2023**

**Тема:** Алгоритм відпалу.

**Мета:** Навчитись використовувати алгоритм відпалу і розв’язати задачу розстановки N шахових ферзів.

**Завдання**

Розв’язати із використанням алгоритму відпалу задачу розстановки N шахових ферзів на шаховій дошці розміру N×N таким чином, аби жоден ферзь не загрожував будь-якому іншому.

**Етапи виконання завдання**

1. Вивчити із використанням запропонованих літературних джерел зміст алгоритму відпалу.

2. Обрати для визначеності фіксоване значення N (для прикладу, стандартний розмір шахової дошки 8×8, тому можна покласти N=8).

3. Реалізація допоміжних функцій випадкової зміни розв’язку та початкової ініціалізації.

4. Реалізація допоміжної функції для оцінки розв’язку.

5. Реалізація допоміжної функції копіювання одного розв’язку в інший.

6. Реалізація допоміжної функції виводу результату на екран у вигляді шахової дошки.

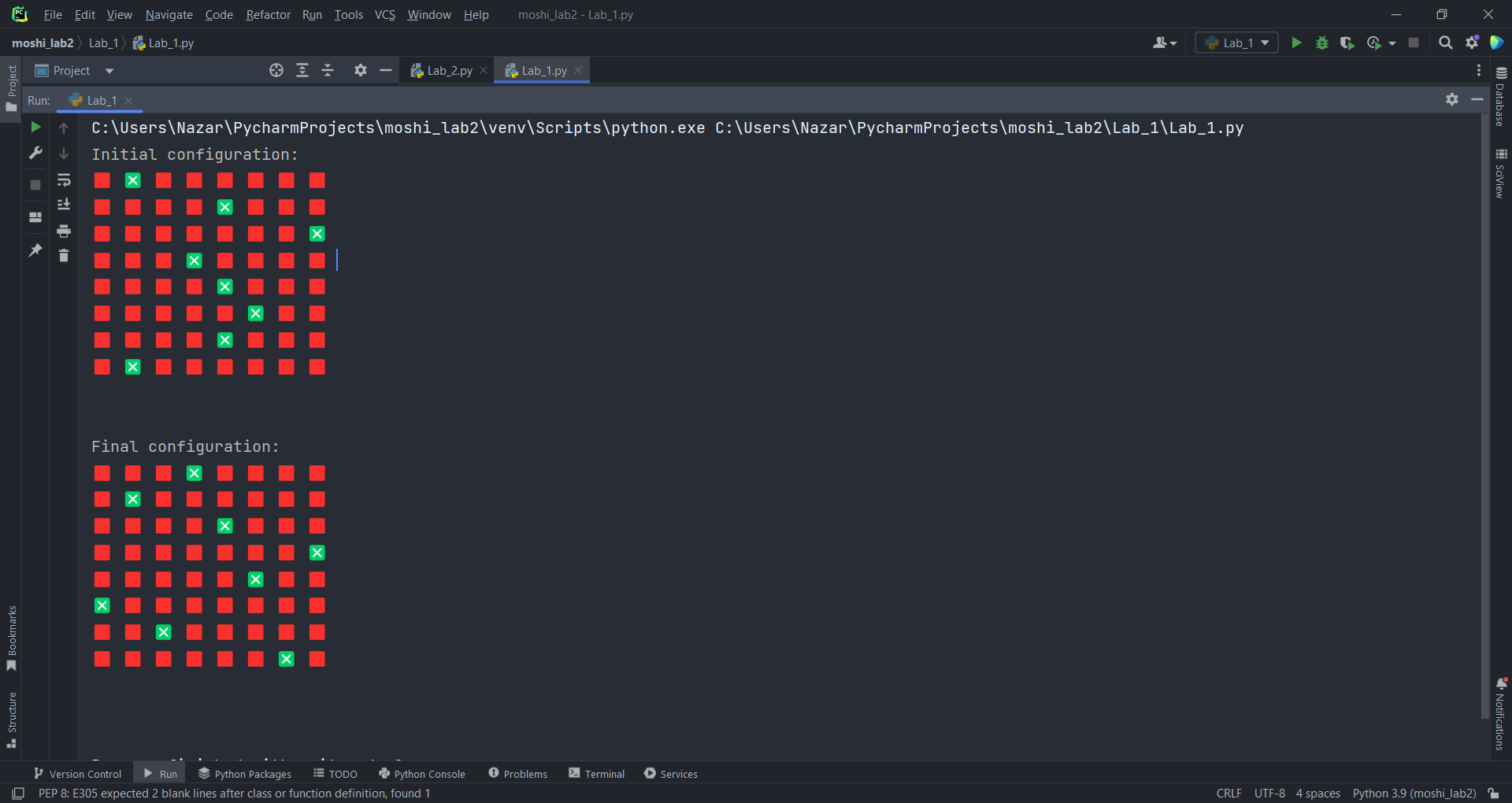
7. Безпосередня реалізація алгоритму відпалу.

8. Знайти інші алгоритми розв’язку задачі та спробувати розробити програмний код для їх реалізації.

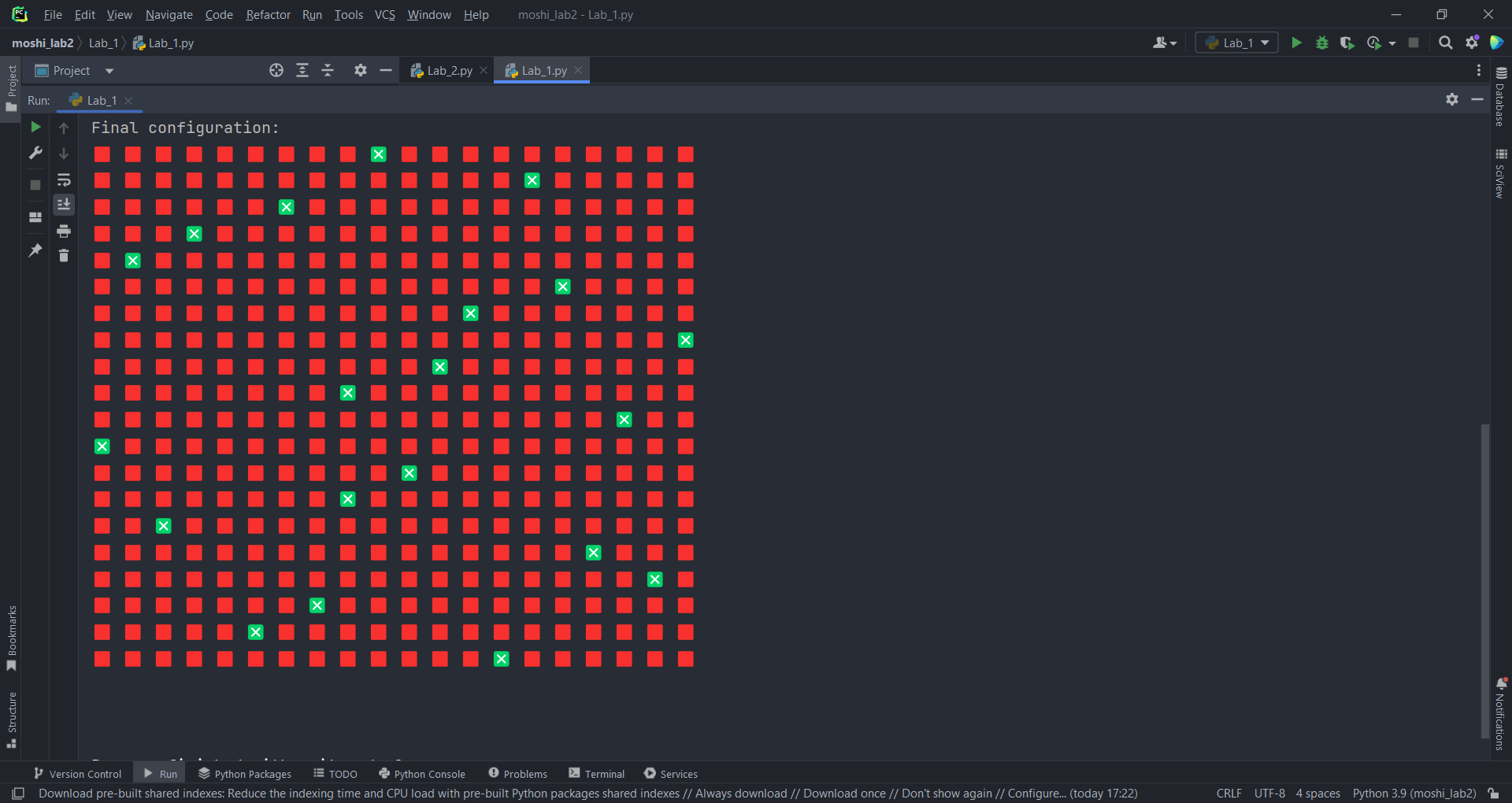
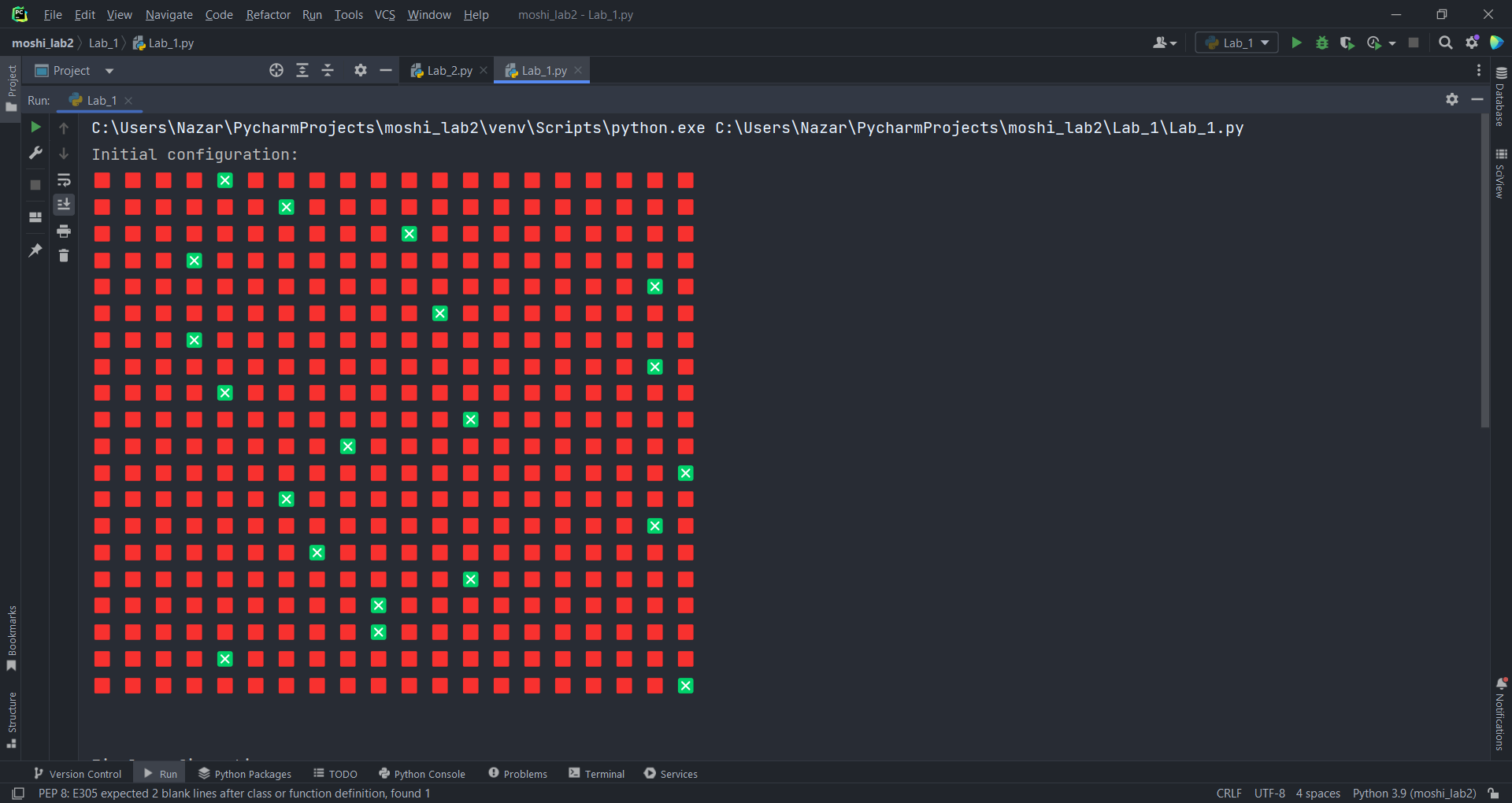
**Програмна реалізація**

*import* random  
*import* math  
  
*class* Chessboard:  
 *def* \_\_init\_\_(*self*, n, board=*None*):  
 *self*.n = n  
 *if* board *is not None*:  
 *self*.board = board  
 *else*:  
 *self*.board = *self*.random\_board()  
  
 *def* random\_board(*self*):  
 *return* [random.randint(0, *self*.n-1) *for* \_ *in* range(*self*.n)]  
  
 *def* print\_board(*self*):  
 *for* row *in self*.board:  
 line = ""  
 *for* col *in* range(*self*.n):  
 *if* col == row:  
 line += "❎ "  
 *else*:  
 line += "🟥 "  
 print(line)  
 print("\n")  
  
 *def* attacks(*self*):  
 count = 0  
 *for* i *in* range(*self*.n):  
 *for* j *in* range(i + 1, *self*.n):  
 *if self*.board[i] == *self*.board[j] *or* abs(*self*.board[i] - *self*.board[j]) == abs(i - j):  
 count += 1  
 *return* count  
  
 *def* energy(*self*):  
 *return self*.attacks()  
  
 *def* neighbor(*self*):  
 new\_board = list(*self*.board)  
 i = random.randint(0, *self*.n-1)  
 j = random.randint(0, *self*.n-1)  
 *while* new\_board[i] == j:  
 j = random.randint(0, *self*.n-1)  
 new\_board[i] = j  
 *return* new\_board  
  
*class* SimulatedAnnealing:  
 *def* \_\_init\_\_(*self*, temperature=1.0, cooling\_rate=0.999):  
 *self*.temperature = temperature  
 *self*.cooling\_rate = cooling\_rate  
  
 *def* acceptance\_probability(*self*, old\_energy, new\_energy):  
 *if* new\_energy < old\_energy:  
 *return* 1.0  
 *else*:  
 *return* math.exp((old\_energy - new\_energy) / *self*.temperature)  
  
 *def* anneal(*self*, chessboard):  
 old\_energy = chessboard.energy()  
 *while self*.temperature > 0.001 *and* old\_energy > 0:  
 new\_board = chessboard.neighbor()  
 new\_energy = Chessboard(chessboard.n, board=new\_board).energy()  
 *if self*.acceptance\_probability(old\_energy, new\_energy) > random.random():  
 chessboard.board = new\_board[:]  
 old\_energy = new\_energy  
 *self*.temperature \*= *self*.cooling\_rate  
 *return* chessboard.board  
  
n = 20  
*# size of chessboard (n x n)*chessboard = Chessboard(n) *# generate initial configuration*print("Initial configuration:")  
chessboard.print\_board()  
  
annealer = SimulatedAnnealing()  
board = annealer.anneal(chessboard) *# find solution using simulated annealing*print("Final configuration:")  
Chessboard(n, board).print\_board()

**1.Результат виконання при N=8**



**2.Результат виконання при N=20**



**Висновок**: На цій лабораторній я навчився використовувати алгоритм відпалу і розв’язав задачу розстановки N шахових ферзів мовою Python.