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 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(назва дисципліни)

на тему:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студентки 1 курсу, групи ІП-23

Федорєєвої Дарії Андріївни

Спеціальності 121 «Інженерія програмного забезпечення»

Керівник ст. вик. Головченко М.М.

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

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

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

|  |  |  |  |
| --- | --- | --- | --- |
| Члени комісії |  |  | ст. викю Головченко М. М. |
|  | (підпис) |  | (посада, вчене звання, науковий ступінь, прізвище та ініціали) |
|  |  |  | Асистент Вовк Є. А. |
|  | (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

Напрям "ІПЗ"

Курс 1 Група ІП-23 Семестр 2

ЗАВДАННЯ

на курсову роботу студентки

Федорєєвої Дарії Андріївни

(прізвище, ім’я, по батькові)

1. Тема роботи «Класичне судоку»

2. Строк здачі студентом закінченої роботи

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

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 рішення задачі |  |  |
| 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: усі сторінки мінус 1 сторінок, 10 рисунків, 11 таблиць, 4 посилання.

Мета роботи: програмна реалізація класичної гри в судоку з полем 9х9, розбитим на квадрати 3х3.

Вивчено метод Вacktracking для генерації валідної матриці гри та її розв'язку. Приведено змістовну постановку задачі, її математична модель, а також описано детальний процес розв’язання.

Виконано програмну реалізацію алгоритму Вacktracking для коректної функціональності класичної гри судоку, проведено її тестування, створена детальна інструкція користувача.

КЛАСИЧНЕ СУДОКУ

Зміст

[Вступ 5](#_Toc135587135)

[1 Постановка задачі 7](#_Toc135587136)

[2 Теоретичні відомості 8](#_Toc135587137)

[3 Опис алгоритмів 11](#_Toc135587138)

[3.1. Загальний алгоритм 11](#_Toc135587139)

[3.2. Алгоритм методу Вacktracking 13](#_Toc135587140)

[4 Опис програмного забезпечення 16](#_Toc135587141)

[4.1. Діаграма класів програмного забезпечення 18](#_Toc135587142)

[4.2. Опис методів частин програмного забезпечення 18](#_Toc135587143)

[4.2.1. Стандартні методи 18](#_Toc135587144)

[4.2.2. Користувацькі методи 21](#_Toc135587145)

[5 Тестування програмного забезпечення 24](#_Toc135587146)

[5.1. План тестування 24](#_Toc135587147)

[5.2. Приклади тестування 25](#_Toc135587148)

[6 Інструкція користувача 31](#_Toc135587149)

[Висновки 38](#_Toc135587151)

[Перелік посилань 39](#_Toc135587152)

[Додаток А Технічне завдання 40](#_Toc135587153)

[Додаток Б Тексти програмного коду 43](#_Toc135587154)

Вступ

Судоку – це захоплююча головоломка, яка захоплює мільйони людей по всьому світу. Мета гри полягає в заповненні дошки 9x9 числами від 1 до 9 включно таким чином, щоб кожна цифра зустрічалася лише один раз у кожному рядку, кожному стовпці та кожній підсітці 3x3.

Програмна реалізація судоку дозволяє вам насолоджуватися цією захоплюючою грою без необхідності мати фізичну дошку та олівці. Завдяки комп'ютерній програмі можна буде вирішувати головоломки судоку, розвивати своє логічне мислення та випробувати свої навички. Вона також має функцію автоматичного заповнення, яка може допомогти, якщо на певному етапі гри людина не може самостійно підібрати коректне число.

У пояснювальній записці буде детально розглянуто ідею основного алгоритму, що використовується у даній роботі, а також буде надано його покрокове пояснення. Крім того, будуть описані усі наявні методи та класи, які використовуються в програмі.

Курсова робота розглядатиме теоретичний аспект, необхідний для розуміння процесу генерації валідних головоломок судоку та обчислення коректних значень для заповнення комірок. Також буде розроблена програмна реалізація алгоритму, яка буде виконувати ці завдання. Детально розкрита робота алгоритму буде у другому розділі курсової роботи.

Для перевірки коректності роботи програмного забезпечення буде проведено повне тестування з різними вхідними значеннями, включаючи ситуації з некоректними вхідними даними. Таке тестування дозволить переконатися, що програма працює правильно та надійно.

З метою забезпечення зручного користування програмним забезпеченням, буде розроблена покрокова інструкція для користувачів. Ця інструкція допоможе користувачам ознайомитися з основними функціями програми та виконати головоломки судоку з легкістю.

Після завершення розробки програмного забезпечення будуть підведені підсумки виконаної роботи, де будуть оцінені результати, досягнуті в ході дослідження та розробки.

# Постановка задачі

Розробити програмну реалізацію класичної гри в судоку з полем 9х9, розбитим на квадрати 3х3. Обрати оптимальний метод для розробки вирішення головоломки судоку.

Гра Судоку складається з сітки чисел 9х9, де кожне число може бути в діапазоні від 1 до 9 включно. Спочатку певна кількість чисел розкривається, а метою гри є заповнення залишку сітки дійсними числами. Сітку поділяють на 9 блоків розміром 3х3. Головне правило гри судоку полягає в тому, що кожен рядок, стовпець та блок повинні містити числа від 1 до 9 включно лише один раз.

Програмне забезпечення повинно обробляти матрицю 9х9 та знаходити значення, що будуть коректними для вирішення головоломки.

Вихідними даними для даної роботи є згенерована матриця 9х9 з натуральними числами від 1 до 9 включно. Програмне забезпечення повинно генерувати гру, яка буде мати вірний розв’язок. Також воно передбачає перевірку коректності введених значень (неможливість введення неправильного значення в комірку); покрокове вирішення головоломки, відповідно до логіки гри, та автоматичне вирішення згенерованої задачі.

Для генерації та вирішення головоломки був обраний метод Вacktracking. Його основна ідея полягає у систематичному переборі всіх можливих комбінацій і зворотному відстеженні при знаходженні неприпустимого значення. Алгоритм зворотного відстеження є ефективним та широко використовуваним підходом для вирішення головоломок судоку. Він надає систематичний та рекурсивний метод дослідження можливих рішень, ефективно використовуючи зворотне відстеження при виникненні конфліктів. Його основна перевага полягає в тому, що він гарантує знайдення правильного розв'язку, якщо він існує.

# Теоретичні відомості

Гра Судоку задається квадратною матрицею розміром 9х9, що складається з 81 клітинки:

Сітку (матрицю) розділено на дев'ять блоків розміром 3 на 3. Деякі з 81 клітинок заповнені числами зі множини {1, 2, 3, 4, 5, 6, 7, 8, 9}. Ці заповнені клітинки називаються підказками. Метою гравця є заповнення всієї сітки, використовуючи дев'ять цифр таким чином, щоб кожен рядок, кожен стовпець та кожен блок містили кожне число лише один раз (єдине правило гри в судоку).

## Метод Вacktracking

Сутність методу Вacktracking полягає в пошуку кожної можливої комбінації для вирішення задач оптимізації. Основний принцип алгоритму Вacktracking для гри судоку полягає в тому, щоб працювати клітинка за клітинкою, з метою вирішення матриці для гри в судоку, що була згенерована та має розв’язок, за відповідним правилом.

У випадку невдалого підбору значень для комірок головоломки, алгоритм повертається на крок назад із метою підбору нових значень. (<https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=7fd55f655d49086e02f8cc8a25dfd613d157ef0b>)

Детальне пояснення роботи алгоритму.

Етапи роботи алгоритму зумовлюються правилами гри судоку та охоплюють п’ять позицій: початок (2.1.1), вибір значення (2.1.2), розміщення значення (2.1.3), зворотне відстеження (2.1.4), рішення (2.1.5).

* + 1. Початок роботи алгоритму.

На початковому етапі роботи, алгоритм створює порожню сітку судоку, яка відображає межі ігрового поля. Далі він визначає порожню клітинку для заповнення, адже користувач повинен отримати поле з декількома заповненими клітинками.

* + 1. Вибір значення.

Відповідно до правил судоку, алгоритм обирає число (від 1 до 9 включно) для заповнення попередньо визначеної ним порожньої клітинки (див. п. 2.1.1). Обравши таке число, алгоритм здійснює перевірку відповідності цього числа правилам судоку, а саме переконується в тому, що таке число не дублюється у тому ж рядку, стовпці або підсітці. Якщо правила не порушуються, то алгоритм переходить до наступного кроку «розміщення значення», а якщо має місце певна невідповідність, то він обирає інше число і повторює цей крок.

* + 1. Розміщення значення.

Знайшовши валідне число, алгоритм розміщує його в порожній клітинці, після чого переходить до визначення наступної порожньої клітинки і повторює кроки 2.1.2 і 2.1.3. В момент, коли порожніх клітинок не залишається, головоломка вважається вирішеною.

* + 1. Зворотне відстеження.

В процесі реалізації кроків 2.1.2 та 2.1.3 є можливість виникнення конфлікту чисел, усунення якого є призначенням етапу «зворотного відстеження». Якщо в обраній алгоритмом поточній клітинці не можна розмістити жодного валідного числа, він повертається до попередньої клітинки. У такому випадку остання підлягає очищенню, після чого алгоритм повертається до попередньої клітинки з кількома можливими іншими варіантами. Далі алгоритм обирає наступне доступне значення і повторює кроки 2.1.2 – 2.1.4. Алгоритм продовжує процес зворотного відстеження допоки не буде знайдене валідне значення, якщо ж останнє не може бути встановлене в жодній попередній клітинці, вважається, що головоломку вирішити неможливо.

* + 1. Рішення.

В ході розв’язання кросворду алгоритм повторює кроки 2.1.2 – 2.1.4, поки всі клітинки не будуть заповнені валідними числами. Як тільки знайдено валідне рішення, головоломка є вирішеною.

В процесі реалізації, алгоритм *backtracking* характеризується певними властивостями, які зводяться до ефективності та оптимізації (а), перевірки валідності головоломки (б) та аналізу складності згенерованого кросворду:

а) Ефективність та оптимізація. Ефективність алгоритму зворотного відстеження залежить від кількості порожніх клітинок. Різні техніки оптимізації можуть покращити продуктивність алгоритму. До таких належать: використання правил судоку для заздалегідь виключення недійсних варіантів; евристики, тобто надання переваги порожнім клітинкам з найменшою кількістю доступних варіантів чисел; попереднє опрацювання, яке полягає в ідентифікації та заповненні клітинок з унікальними можливими значеннями; порушення симетрії, тобто уникнення зайвого дослідження симетричних рішень.

б) Перевірка валідності головоломки. В деяких випадках головоломки Судоку можуть не мати валідного рішення. Алгоритм зворотного відстеження може встановити нерозв’язність головоломки, досліджуючи всі можливі варіанти без знаходження рішення (див. п. 2.1.4). Алгоритм зупиняється, коли всі клітинки заповнені або констатує неможливість розміщення валідного числа в жодну клітинку, яка підлягає заповненню.

в) Аналіз складності згенерованого кросворду. Часова складність алгоритму зворотного відстеження для головоломок судоку залежить від характеристик головоломки, зокрема від кількості порожніх клітинок. У найгіршому випадку алгоритм має експоненційну часову складність, але застосування передбачення обмежень і евристик може значно покращити його продуктивність.

# Опис алгоритмів

Перелік всіх основних змінних та їхнє призначення наведено в таблиці 3.1.

Таблиця 3.1 – Основні змінні та їхні призначення

|  |  |
| --- | --- |
| Змінна | Призначення |
| board | Валідна матриця гри |
| screen | Для відображення відповідного вікна гри (HOME/PLAY/END) |
| mouse\_x | Координата мишки користувача по осі Ох |
| mouse\_у | Координата мишки користувача по осі Оу |
| \_start\_time | Для підрахунку часу під час вирішення головоломки |
|  | Кортеж із індексами ряду та колонки матриці, де ще немає правильної відповіді для вирішення головоломки |
| num | Змінна цілочисельного типу від 1 до 9 включно для перевірки чисел на повторення у рядку/колонці/підсітці |
| i | Каунтер у циклі |
| j | Каунтер у циклі |
| c | Каунтер у циклі |
| r | Каунтер у циклі |

## Загальний алгоритм

1. ПОЧАТОК
2. Згенерувати валідну матрицю для гри в судоку.
3. Записати початкову матрицю гри в файл.
4. ЦИКЛ
   1. ЯКЩО screen дорівнює «HOME», ТО зчитати дії користувача.
      1. ЯКЩО користувач натискає на кнопку закриття вікна, ТО завершити програму.
      2. КІНЕЦЬ
      3. ЯКЩО користувач натискає на кнопку «Generate Board!», ТО
         1. screen дорівнює «PLAY»
         2. Ініціалізація \_start\_time для запуску таймеру.
   2. ЯКЩО screen дорівнює «PLAY», ТО зчитати дії користувача.
      1. ЯКЩО користувач натискає на кнопку закриття вікна, ТО завершити програму.
      2. КІНЕЦЬ
      3. ЯКЩО користувач обирає клітинку для вводу даних для вирішення гри в судоку, ТО зчитати координати курсора користувача у змінні mouse\_x, mouse\_y та за ними знайти (обрахувати) ряд та колонку матриці board, для введення даних користувача.
      4. ЯКЩО користувач натискає кнопку на клавіатурі, ТО
         1. ЯКЩО натиснутий символ чи цифра, ТО перевірити її валідність для гри в судоку.
         2. ЯКЩО натиснутий пробіл, ТО надати рішення до головоломки.
      5. ЯКЩО неможливо знайти пусту комірку для заповнення (ряд та колонку в матриці гри), ТО
         1. Зупинити таймер
         2. screen дорівнює «END»
         3. Запис даних гри у файл.
   3. ЯКЩО screen дорівнює «END», ТО зчитати дії користувача.
      1. ЯКЩО користувач натискає на кнопку закриття вікна, ТО завершити програму.
      2. КІНЕЦЬ
      3. ЯКЩО користувач натискає на кнопку «Generate New Game», ТО
         1. screen дорівнює «PLAY»
         2. Ініціалізація \_start\_time запуску таймеру
         3. Генерація нової матриці для гри в судоку
         4. Запис початкової матриці у файл.

## Алгоритм методу Вacktracking

1. ПОЧАТОК
2. Перевірити чи матриця (board), передана параметром, має значення 0 у будь-якому з рядків за допомогою функції find\_empty(board):
   1. ЯКЩО матриця має не має значення 0 у будь-якому з рядків, ТО повернути TRUE.

ІНАКШЕ:

* + 1. Ініціалізація змінної position значенням, що поверне функція find\_empty(grid): .
    2. ЦИКЛ перебору чисел від 1 до 9 включно
       1. Перевірити за допомогою функції valid чи значення змінної і немає в рядку, стовпці чи квадратному секторі 3 на 3 в матриці (board): valid(board, i, posotion):
          1. ЯКЩО функція valid повернула TRUE:

Присвоюємо значення змінної і матриці (board) за індексами .

Перевірити за допомогою функції solve чи матриця (board) має розв’язок із доданим значенням і.

ЯКЩО функція solve повернула TRUE, ТО повернути TRUE. ІНАКШЕ присвоїти матриці (board) 0 за індексами

1. Повернути FALSE.
2. КІНЕЦЬ

## Алгоритм функції valid(board, num, position):

1. ПОЧАТОК
2. ЦИКЛ перебору чисел від 0 до 9 :

2.1 Перевірити чи передане число num дорівнює якомусь із чисел з переданого рядку матриці (grid)

2.1.1 ЯКЩО num дорівнює якомусь числу із рядка матриці (grid), ТО повернути FALSE

3. ЦИКЛ перебору чисел від 0 до 9 :

3.1 Перевірити чи передане число num дорівнює якомусь із чисел з переданого стовпця матриці (board)

3.1.1 ЯКЩО num дорівнює якомусь числу із стовпця матриці (board) , ТО повернути FALSE

4. ЦИКЛ перебору чисел від до

…,

4.1. ЦИКЛ перебору чисел від до

…,

4.1.1. ЯКЩО num дорівнює якомусь числу із обрахованого квадратного сектора 3 на 3, , ТО повернути FALSE

5. Повернути TRUE

6. КІНЕЦЬ

## Алгоритм функції find\_empty(board):

1. ПОЧАТОК
2. ЦИКЛ перебору чисел від 0 до 9 :
   1. ЦИКЛ перебору чисел від 0 до 9 :
   2. ЯКЩО значення матриці (grid) за індексами і та j дорівнює 0, , ТО повернути і та j
3. Повернути FALSE
4. КІНЕЦЬ

## Алгоритм функції generate():

1. ПОЧАТОК
2. Ініціалізувати квадратну матрицю 9 на 9 з усіма значеннями 0
3. Ініціалізувати змінну row рандомним значенням від 0 до 9
4. Ініціалізувати змінну col рандомним значенням від 0 до 9
5. Ініціалізувати змінну num рандомним значенням від 1 до 9 включно
6. ЦИКЛ перебору чисел від 0 до 30 :

6.1. ЦИКЛ поки функція valid не поверне значення TRUE АБО поки матриця (grid) за індексам row i col не буде дорівнювати 0:

6.1.1. Ініціалізувати змінну row рандомним значенням від 0 до 9

6.1.2. Ініціалізувати змінну col рандомним значенням від 0 до 9

6.1.3. Ініціалізувати змінну num рандомним значенням від 1 до 9 включно

6.2. Присвоїти значення num матриці (board) за індексам row i col

7. Копіювання матриці (board) у список copy\_grid

8. ЯКЩО матриця copy\_grid не може бути вирішена за правилом гри в судоку, функція solve повертає значення FALSE, solve(copy\_grid), ТО матриці (board) присвоюємо значення, яке повертає функція generate

9. Повернути матрицю board

10. КІНЕЦЬ

## Алгоритм функції find\_best\_indexes(board):

1. ПОЧАТОК
2. Ініціалізувати список для пустих комірок
3. Цикл перебору чисел від 0 до 9
   1. Цикл перебору чисел від 0 до 9
      1. ЯКЩО елемент матриці board за індексами i та j дорівнює 0, ТО кількість можливих чисел для цієї комірки дорівнює 0
         1. ЦИКЛ для num від 1 до 9 включно

* + - * 1. ЯКЩО функція is\_valіd\_num повернула TRUE, ТО кількість можливих чисел для цієї комірки збільшити на 1
    1. У список пустих комірок додати індекси комірки та кількість можливих у ній чисел

1. Відсортувати список пустих комірок за кількістю у них можливих чисел за зростанням
2. КІНЕЦЬ

## Алгоритм функції is\_valid\_num(board, num, i, j):

1. ПОЧАТОК
2. ЦИКЛ для с від 0 до 9
   1. ЯКЩО число в матриці board зі індексами i та с дорівнює num, ТО повернути FALSE
3. ЦИКЛ для r від 0 до 9
   1. ЯКЩО число в матриці board зі індексами r та j дорівнює num, ТО повернути FALSE
4. ЦИКЛ перебору чисел від до

…,

4.1. ЦИКЛ перебору чисел від до

…,

* + 1. ЯКЩО число в матриці board зі індексами r та j дорівнює num, ТО повернути FALSE

1. Повернути TRUE
2. КІНЕЦЬ

# Опис програмного забезпечення

## У склад програмної реалізації гри судоку входять кілька класів, що забезпечують різні функціональні можливості. Головним класом є Main (4.1.1); клас Screen (4.1.2) є базовим для опису функціональності кнопок; клас Game (4.1.3) відповідає за генерацію, перевірку головоломки та взаємодію з користувачем через графічний інтеріейс, HomeScreen (4.1.4) та EndScreen (4.1.5) також відповідають за взіємодію з користувачем через графічний інтерфейс.

### Main

Клас Main – головний клас програми. Полями класу є \_minutes, \_seconds, \_start\_time. Методами класу є \_write\_to\_file (для збереження усіх даних гри у файл), \_timer (для запуску таймера на період розв’язування головоломки судоку) та метод game (призначений для запуску головного циклу програмного забезпечення та взаємодії з користувачем).

### Game

Клас Game призначений для взаємодії з користувачем під час вирішення головоломки через графічний інтерфейс. Полями класу є \_solver, \_board, \_initial\_board, \_mouse\_active, \_key\_active, \_info, \_mistakes, \_selected\_col, \_selected\_row, \_location. У цьому класі прописані необхідні гетери та сетери, методи draw\_game (для побудови матриці головоломки із початковими підказками), find\_location (для знаходження обраної секції для вводу даних користувача у матрицю за координатами місцезнаходження курсору), draw\_sel\_box (для обведення обраної секції), detect\_keys (для отримання введених користувачем даних), draw\_num (для показу на екрані введених користувачем цифр), \_finalize\_key (для підтвердження валідності введених користувачем даних), draw\_mistakes (для виведення на екран зроблених користувачем помилок). Також присутні методи valid (для визначення чи підходить введене число для цієї матриці), find\_empty (для знаходження пустої комірки у матриці гри), solve (для рекурсивного розв’язання головоломки), generate (для генерації валідної матриці для головоломки), \_find\_best\_indexes (для знаходження кількості можливих значень у пустій комірці матриці гри) та \_is\_valid\_num (для перевірки можливості вставки висла в пусту комірку).

### Screen

Клас Screen призначений для опису метода \_button (для відображення та функціональності кнопок), який буде далі використовуватися у класах, які наслідуються від класу Screen, для взаємодії з користувачем. В ньому прописані такі поля: \_size\_x, \_button\_x, \_size\_y, \_button\_y, \_main\_text, \_small\_text, \_button\_text, \_button\_active, \_mouse.

### HomeScreen

Клас HomeScreen також призначений для взаємодії з користувачем через графічний інтерфейс. Він наслідується він від класу Screen, а в процесі роботи за командою користувача переходить на вікно класу Game. В класі HomeScreen наявний метод draw\_home, функціонал якого полягає в тому, щоб відобразити назву головоломки та кнопку початку гри, після активації якої буде згенерована валідна матриця.

### EndScreen

Клас EndScreen призначений для взаємодії з користувачем через графічний інтерфейс. Наслідується він від класу Screen. В процесі роботи за командою користувача переходить на вікно класу Game. В ньому прописаний метод draw\_over, за допомогою якого відображається кількість помилок, час, за який була пройдена гра та кнопка для генерації нової головоломки.

Діаграма класів програмного забезпечення зображена на рисунку 4.1:

Изображение выглядит как текст, диаграмма, План, Параллельный

Автоматически созданное описание

Рисунок 4.1 – Діаграма класів

## Опис методів частин програмного забезпечення

Програмне забезпечення включає стандартні та користувацькі методи. Стандартні методи - вбудовані в мову програмування або фреймворк і використовуються для загальних завдань. Користувацькі методи - створюються програмістами для вирішення конкретних завдань у програмі.

### Стандартні методи

У таблиці 1.1 наведено перелік та опис усіх стандартних функцій, використаних у курсовій роботі.

Таблиця 4.1 – Стандартніметоди

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва класу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 1 |  | sum | Знаходить суму усіх елементів матриці | Послідовність чисел | Число |  |

Продовження таблиці 4.1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 |  | open | Відкриває файл(якщо можливо) | Строка/символ | Файловий об’єкт |  |
| 3 |  | deepcopy | Робить глибоку копію об’єкту | Матриця | Матриця | copy |
| 4 |  | init | Ініціалізує pygame |  |  | pygame |
| 5 | time | Clock |  |  | Об’єкт класу pygame.time.Clock | pygame |
| 6 | time | get\_ticks | Повертає пройдений час в секундах |  | Число | pygame |
| 7 | Clock | tick |  | Число |  | pygame |
| 8 | display | set\_mode | Створює графічне вікно | Кортеж з двох числових елементів | Об’єкт класу pygame.display.set\_mode | pygame |
| 9 | display | set\_caption | Прописує назву створеного графічного вікна | Символ/строка | Об’єкт класу pygame.display.set\_caption | pygame |
| 10 | font | SysFont | Створює об’єкт із прописаним шрифтом | Строка та число (розмір шрифту) | Об’єкт класу pygame.font.SysFont | pygame |
| 11 | font | get\_rect | Створює прямокутну область охоплення тексту |  | Об’єкт класу pygame.font.SysFont.render.get\_rect | pygame |
| 12 | font | render | Виконує рендериг тексту | Символ/строка, булевий параметр, колір | Об’єкт класу pygame.font.SysFont.render | pygame |

Продовження таблиці 4.1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 13 | font | blit | Виконує відображення тексту на графічному вікні | Об’єкт класу pygame.font.SysFont.render, Об’єкт класу pygame.font.SysFont.render.get\_rect |  | pygame |
| 14 | draw | line | Відображає лінії на графічно1му інтерфейсі | Число (відступ від краю екрану), колір, координата по вертикалі, кінцева координата лінії, товщина |  | pygame |
| 15 | draw | rect | Відображає прямокутник на графічному вікні | Колір, координати місцезнаходження прямокутника, ширина та висота, товщина лінії |  | pygame |
| 16 |  | sleep | Затримання роботи програми на вказану кількість секунд | Число |  | time |

### Користувацькі методи

У таблиці 1.2 наведено перелік та опис усіх користувацьких функцій, використаних в даній курсовій роботі.

Таблиця 4.2– Користувацькі методи

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва класу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 1 | Main | \_write\_to\_file | Для запису всіх даних гри у файл | Матриця гри в судоку, кількість помилок, час |  | main |
| 2 | Main | \_timer | Для запису тривалості гри | Булевий параметр (чи відкрито ігрове вікно) | Кортеж з двох числових елементів (хвилини, секунди) | main |
| 3 | Main | game | Запуск усієї програми |  |  | main |
| 4 | Solver | valid | Визначення валідності числа для матриці для розв’язання гри в судоку | Матриця гри, число, що буде перевірятися, індекси позиції, для якої буде перевірятися число | Булеве значення | game |
| 5 | Solver | find\_empty | Визначення чи є вільною хоча б одна комірка в матриці для гри в судоку | Матриця згенерованої гри | Якщо комірка вільна, індекси, якщо ні – False | game |
| 6 | Solver | solve | Для розв’язання головоломки судоку (визначення чи згенерована матриця є валідною) | Матриця згенерованої гри (на початку генерації може не мати розв’язку), під час гри матриця завжди має розв’язок | Булеве значення | game |

Продовження таблиці 4.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 7 | Solver | generate | Для генерації валідної матриці для гри в судоку |  | Початкову матрицю гри в судоку | game |
| 8 | Game | draw\_game | Для відображення матриці гри |  |  | game |
| 9 | Game | find\_location | Для знаходження місцеположення курсору користувача на графічному інтерфейсі | Два числа (координати курсора) |  | game |
| 10 | Game | draw\_sel\_box | Для обведення виділеної підсекції |  |  | game |
| 11 | Game | detect\_keys | Для початкової обробки введених даних користувачем | Символ |  | game |
| 12 | Game | draw\_num | Для відображення цифр |  |  | game |
| 13 | Game | \_finilize\_key | Для кінцевої перевірки чисел введених користувачем |  |  | game |
| 14 | Game | draw\_mistakes | Для відображення кількості помилок |  |  | game |
| 15 | Screen | button | Для відображення активних кнопок на графічному інтерфейсі |  |  | screen |

Продовження таблиці 4.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 16 | HomeScreen | draw\_home | Для відображення початкового вікна графічного інтерфейсу (назву гри та активної кнопки) |  |  | screen |
| 17 | EndScreen | draw\_orver | Для відображення кінцевого вікна графічного інтерфейсу (кількість помилок, тривалість гри та активної кнопки) |  |  | screen |

# Тестування програмного забезпечення

Під час розробки програмного забезпечення може статися так, що певні деталі і аспекти не отримують достатньої уваги або пропускаються через неуважність розробників. Такі пропуски можуть мати серйозні наслідки, зокрема, призводити до некоректної роботи програми або навіть спричиняти її збої. Щоб запобігти виникненню таких проблем і забезпечити належну якість продукту, необхідно проводити тестування. Для перевірки правильності роботи програми ми здійснемо тестування за наступними критеріями:

## 

1) Тестування правильності введених значень.

а. Тестування при введенні некоректних символів.

б. Тестування при введенні порожніх значень.

2) Тестування коректності роботи при введенні коректних даних.

а. Перевірка коректності роботи при введенні чисел від 1 до 9 включно.

3) Тестування коректності вирішення головоломки.

а. Перевірка коректності роботи методу Вacktracking.

## Приклад тестування

Проведемо певну кількість тестувань для кожного пункту плану, результати запишемо до таблиць 5.1 – 5.8:

### Тестування правильності введених значень:

Таблиця 5.1 – Приклад роботи програми при введенні некоректних символів

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість введення некоректних даних. |
| Початковий стан програми | Відкрито вікно програми у режимі «Play», обраний четвертий рядок та 5 колонка для введення значень. |
| Вхідні дані | № а й k \* / |
| Схема створення тесту | Введення некоректних даних. |
| Очікуваний результат | Програма не має показувати та обробляти некоректні дані. |
| Стан програми після проведення випробувань | Обрана комірка для вводу даних залишилася пустою. |

Таблиця 5.2 – Приклад роботи програми при введенні некоректних символів

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість введення некоректних даних. |
| Початковий стан програми | Відкрито вікно програми у режимі «Play», обраний третій рядок та шоста колонка для введення значень |
| Вхідні дані | К ? mm = + |
| Схема створення тесту | Введення некоректних даних. |
| Очікуваний результат | Програма не має показувати та обробляти некоректні дані. |

Продовження таблиці 5.2

|  |  |
| --- | --- |
| Стан програми після проведення випробувань | Обрана комірка для вводу даних залишилася пустою. |

Таблиця 5.3 – Приклад роботи програми при введенні порожніх значень

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість введення некоректних даних. |
| Початковий стан програми | Відкрито вікно програми у режимі «Play», обраний четвертий рядок та 5 колонка для введення значень. |
| Вхідні дані |  |
| Схема створення тесту | Введення некоректних даних. |
| Очікуваний результат | Програма не має показувати та обробляти некоректні дані. |
| Стан програми після проведення випробувань | Обрана комірка для вводу даних залишилася пустою. |

Таблиця 5.4 – Приклад роботи програми при введенні порожніх значень

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість введення некоректних даних. |
| Початковий стан програми | Відкрито вікно програми у режимі «Play», обраний третій рядок та шоста колонка для введення значень. |
| Вхідні дані |  |
| Схема створення тесту | Введення некоректних даних. |
| Очікуваний результат | Програма не має показувати та обробляти некоректні дані. |
| Стан програми після проведення випробувань | Обрана комірка для вводу даних залишилася пустою. |

Таблиця 5.5 – Приклад роботи програми при введенні коректних даних (чисел від 1 до 9 включно)

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність роботи з коректними даними |
| Початковий стан програми | Відкрито вікно програми у режимі «Play», обраний третій рядок та шоста колонка для введення значень. |
| Вхідні дані | 2 |
| Схема створення тесту | Введення коректних даних. |
| Очікуваний результат | Виведення на екран числа, що було введено до натискання кнопки Enter. За умови виконання усіх вимог для вирішення головоломки число залишається на екрані, інакше – зникає. |
| Стан програми після проведення випробувань | Обрана комірка для вводу даних залишилася пустою адже введене число не підходило для розв’язку головоломки. |

Таблиця 5.6 – Приклад роботи програми при введенні коректних даних (чисел від 1 до 9 включно)

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність роботи з коректними даними |
| Початковий стан програми | Відкрито вікно програми у режимі «Play», обраний сьомий рядок та восьма колонка для введення значень |
| Вхідні дані | 9 |
| Схема створення тесту | Введення коректних даних. |

Продовження таблиці 5.6

|  |  |
| --- | --- |
| Очікуваний результат | Виведення на екран числа, що було введено до натискання кнопки Enter. За умови виконання усіх вимог для вирішення головоломки число залишається на екрані, інакше – зникає. |
| Стан програми після проведення випробувань | Обрана комірка для вводу даних була заповнена введеним числом адже воно підходило для розв’язку головоломки. |

Таблиця 5.7 – Приклад тестування коректності роботи (обчислень правильних значено) методу Вacktracking

|  |  |
| --- | --- |
| Мета тесту | Перевірити алгоритм Вacktracking при роботі з числами від 1 до 9 включно. |
| Початковий стан програми | Відкрито вікно програми у режимі «Play», обраний восьмий рядок та сьома колонка для введення значень. |
| Згенерована матриця для гри в судоку | |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | 9 |  |  | 4 |  | 6 |  | 2 | |  | 7 |  | 6 |  |  |  |  |  | | 6 | 1 |  |  |  |  |  | 8 | 9 | | 1 |  |  |  |  |  |  |  |  | |  | 6 |  | 8 |  | 3 |  |  |  | |  | 4 |  | 7 | 1 |  |  |  |  | | 2 |  |  |  |  | 8 |  |  | 4 | |  |  |  |  | 6 |  |  |  |  | |  | 3 | 7 |  | 2 |  |  | 5 |  | |

Продовження таблиці 5.7

|  |  |
| --- | --- |
| Вхідні дані | 2 |
| Схема створення тесту | Введення коректних даних. |
| Очікуваний результат | |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | 9 |  |  | 4 |  | 6 |  | 2 | |  | 7 |  | 6 |  |  |  |  |  | | 6 | 1 |  |  |  |  |  | 8 | 9 | | 1 |  |  |  |  |  |  |  |  | |  | 6 |  | 8 |  | 3 |  |  |  | |  | 4 |  | 7 | 1 |  |  |  |  | | 2 |  |  |  |  | 8 |  |  | 4 | |  |  |  |  | 6 |  | **2** |  |  | |  | 3 | 7 |  | 2 |  |  | 5 |  | |
| Стан програми після проведення випробувань | Обрана комірка для вводу даних була заповнена введеним числом адже воно підходило для розв’язку головоломки. |

Таблиця 5.8 – Приклад тестування коректності роботи (обчислень правильних значено) методу Вacktracking

|  |  |
| --- | --- |
| Мета тесту | Перевірити алгоритм Вacktracking при роботі з числами від 1 до 9 включно. |
| Початковий стан програми | Відкрито вікно програми у режимі «Play», обраний восьмий рядок та сьома колонка для введення значень. |

Продовження таблиці 5.8

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Згенерована матриця для гри в судоку | |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | 9 |  |  | 4 |  | 6 |  | 2 | |  | 7 |  | 6 |  |  |  |  |  | | 6 | 1 |  |  |  |  |  | 8 | 9 | | 1 |  |  |  |  |  |  |  |  | |  | 6 |  | 8 |  | 3 |  |  |  | |  | 4 |  | 7 | 1 |  |  |  |  | | 2 |  |  |  |  | 8 |  |  | 4 | |  |  |  |  | 6 |  |  |  |  | |  | 3 | 7 |  | 2 |  |  | 5 |  | |
| Вхідні дані | 8 |
| Схема створення тесту | Введення коректних даних. |
| Очікуваний результат | |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | 9 |  |  | 4 |  | 6 |  | 2 | |  | 7 |  | 6 |  |  |  |  |  | | 6 | 1 |  |  |  |  |  | 8 | 9 | | 1 |  |  |  |  |  |  |  |  | |  | 6 |  | 8 |  | 3 |  |  |  | |  | 4 |  | 7 | 1 |  |  |  |  | | 2 |  |  |  |  | 8 |  |  | 4 | |  |  |  |  | 6 |  |  |  |  | |  | 3 | 7 |  | 2 |  |  | 5 |  | |
| Стан програми після проведення випробувань | Обрана комірка для вводу даних залишилася пустою адже введене число не підходило для розв’язку головоломки. |

# Інструкція користувача

## Робота з програмою

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

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Рисунок 6.1 – Початкове вікно програми

Програма генерує матрицю 9х9 виконуючи всі умови гри в судоку (головоломка має розв’язок). Для того, щоб почати гру потрібно натиснути на кнопку «Generate Board!». Після цього відкривається ігрове вікно (рисунок 6.2) із згенерованою матрицею (згенеровані матриці не повторюються), лічильником помилок (для чисел, що не підходять для вирішення судоку) та таймером для головоломки 9х9.

Изображение выглядит как текст, снимок экрана, линия, число

Автоматически созданное описание

Рисунок 6.2 – Ігрове вікно програми

Після початку гри для введення будь-яких значень у комірки матриці користувач повинен обрати комірку та клікнути на неї мишею, після чого вона буде обведеню зеленим кольором і готовою для вводу даних (рисунок 6.3).

Изображение выглядит как текст, снимок экрана, число, линия

Автоматически созданное описание

Рисунок 6.3 – Ігрове вікно програми з обраною коміркою

Дані користувач повинен вводити з клавіатури. За умови введення символів, що не використовуються для надання підказок чи вирішення гри, вони не будуть відображатися на ігровому полі. За умови введення чисел від 1 до 9 включно, вони будуть відображатися на екрані до натиснення кнопки Enter (рисунок 6.4).

Изображение выглядит как текст, снимок экрана, число, прямоугольный

Автоматически созданное описание

Рисунок 6.4 – Ігрове вікно програми з введеним значенням

Якщо введене число не підходить для вирішення головоломки, як показано на рисунку 6.4, введене число у комірку буде відображатися до моменту натискання клавіші Enter, після чого комірка знову буде пустою та лічильник помилок збільшується на 1 (рисунок 6.5).

Изображение выглядит как текст, снимок экрана, число, линия

Автоматически созданное описание

Рисунок 6.5 – Ігрове вікно програми після введення неправильного значення

Якщо введене число підходить для вирішення головоломки, воно залишиться у комірці після натискання клавіші Enter (рисунок 6.6).

Изображение выглядит как текст, снимок экрана, число, линия

Автоматически созданное описание

Рисунок 6.6 – Ігрове вікно програми з правильно введеним значенням

За бажанням користувач може отримати підказку для знаходження відповіді обраної комірки. Для цього йому необхідно обрати комірку, клікнути на неї мишею та натиснути кнопку «\» на клавіатурі («\» на екрані відображатися не буде), після цього з’явиться коректне значення для вирішення головоломки у обраній комірці (рисунок 6.7).

Изображение выглядит как текст, снимок экрана, число, линия

Автоматически созданное описание

Рисунок 6.7 – Ігрове вікно програми після використання підказки («\»)

За бажанням користувач може отримати відповідь на всю головоломку та закінчити поточну гру. Для цього йому необхідно натиснути клавішу Space на клавіатурі, після чого користувач кілька секунд буде бачити повністю вирішену головоломку на екрані (рисунок 6.8).

Изображение выглядит как текст, снимок экрана, число, календарь

Автоматически созданное описание

Рисунок 6.8 – Ігрове вікно програми з автоматично вирішеною головоломкою

Користувач бачить вирішену головоломку кілька секунд, після чого з’являється кінцеве вікно програми на якому будуть зображені кількість помилок, які зробив користувач, та час, за який була вирішена головоломка. За бажанням користувач може згенерувати нову головоломку, натиснувши на кнопку «Generate New Game!» (рисунок 6.9)

Изображение выглядит как текст, снимок экрана, Шрифт, Бренд

Автоматически созданное описание

Рисунок 6.9 – Кінцеве вікно програми

## Формат вхідних та вихідних даних

Користувачу подається частково заповнена матриця для гри судоку розміром 9 на 9. Користувач повинен заповнювати кожну пусту комірку числами від 1 до 9 включно.

Результатом роботи програми є виведення на екран вирішеної користувачем головоломки та запису усіх даних про гру у файл (згенерована матриця, повністю заповнена матриця гри, кількість помилок та час, за який була вирішена головоломка).

## Системні вимоги до програмного забезпечення наведені в таблиці 6.1

Таблиця 6.1 – Системні вимоги програмного забезпечення

|  |  |  |
| --- | --- | --- |
|  | **Мінімальні** | **Рекомендовані** |
| **Операційна система** | Windows 7/ Windows 8/Windows 10/ Windows 11 (з останніми обновленнями) | Windows 10/ Windows 11 (з останніми обновленнями) |
| **Процесор** | Intel® Core® i5-4690K 3.50 GHz або AMD FX-8300 3.3 GHz | Intel® Core® i5-10400F 2.90 GHz або AMD Ryzen 5 1600 3.2 GHz |

Продовження таблиці 6.1

|  |  |  |
| --- | --- | --- |
| **Оперативна пам’ять** | 1 GB | 4 GB |
| **Відеоадаптер** | NVIDIA GeForce 940MX (або сумісний аналог) | |
| **Дисплей** | 1280х1024 | 1920х1080 |
| **Прилади введення** | Комп’ютерна миша, клавіатура | |
| **Додаткове програмне забезпечення** | Python 3.11.2 pygame 2.3.2  time 3.10.11 random 3.10.11  copy 3.10 | |

# висновки

Після вдалого завершення проекту я реалізувала класичну гру судоку, використовуючи алгоритм backtracking. Моє програмне забезпечення повністю відповідає всім вимогам, які були зазначені у технічному завданні. Зокрема, воно здатне генерувати валідні головоломки з можливістю надання підказок у вигляді неможливості введення некоректних значень у комірку та автоматичного розв'язання, а також зберігати результати гри у текстовому файлі.

Я докладно вивчила суть алгоритму backtracking та адаптувала його для програмної реалізації гри судоку. В результатах своїх досліджень я надала детальний опис методу backtracking, який використовувала у моєму програмному забезпеченні. Я також надала псевдокод цього методу для кращого сприйняття принципу його роботи.

У моєму описі програмного забезпечення я детально розповіла про його структуру та зв'язки між класами. Я пояснила, які поля використовуються у кожному класі, а також описала всі використані функції, включаючи як стандартні функції, так і користувацькі функції, які я розробила для потреб гри судоку.

Після завершення розробки програмного забезпечення я провела тестування, щоб переконатися, що програма працює коректно. Я перевірила різні сценарії гри, включаючи генерацію валідних головоломок, надання підказок (введення некоректних значень), автоматичне розв'язання та збереження результатів гри. Тести підтвердили, що програма працює коректно і задовольняє всі вимоги.

Крім того, я скомпонувала докладну користувацьку інструкцію для гри судоку. Ця інструкція містить пояснення правил гри, опис інтерфейсу програми, а також кроки, які користувач повинен виконати для того, щоб почати та успішно завершити гру.

В цілому, моє програмне забезпечення є повноцінною реалізацією гри судоку з використанням алгоритму backtracking, воно відповідає всім вимогам технічного завдання і надає зручний та зрозумілий інтерфейс для користувачів.

Перелік посилань

1. SudokuSolver. URL: <https://sudokuspoiler.com/sudoku/sudoku9> (дата звернення: 25.04.2023)
2. Study on the Performance Characteristics of Sudoku Solving Algorithms. URL: <http://surl.li/hhogg> (дата звернення: 21.03.2023)
3. A study of Sudoku Solving Algorithms. URL: <http://surl.li/hhojc> (дата звернення: 20.04.2023)

Додаток А Технічне завдання

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра

інформатики та програмної інженерії

Затвердив

Керівник \_\_Головченко М.М.\_\_\_\_

«7» березня 2023 р.

Виконавець:

Студентка Федорєєва Дарія Андріївна

«7» березня 2023 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання курсової роботи

на тему: «Класичне судоку»

з дисципліни:

«Основи програмування»

Київ 2023

* 1. *Мета*: Метою курсової роботи є програмна реалізація класичної гри в судоку з полем 9х9, розбитим на квадрати 3х3.
  2. *Дата початку роботи*: «07» березня 2023 р.
  3. *Дата закінчення роботи*: «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 202\_ р.
  4. *Вимоги до програмного забезпечення*.

1. Функціональні вимоги:

* Можливість генерації головоломки
* Можливість введення даних для вирішення головоломки (введення значень в комірки головоломки)
* Можливість перевірки правильності вирішеної головоломки
* Можливість надання підказок: неможливість введення неправильного значення в комірку
* Можливість автоматичного вирішення головоломки
* Можливість покрокового вирішення головоломки, відповідно до логіки гри
* Можливість збереження у текстовий файл

1. Нефункціональні вимоги:

* Розробку програмного забезпечення виконати мовою Python.
* Можливість запускати програмне забезпечення на операційній системі Windows 10 та вище
* 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.

* 1. *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.).
   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-23 І курсу*

*Федорєєвої Дарії Андірївни*

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

*14 арк, 124 Кб*

(Вид носія даних)

*Жорсткий диск*

(Найменування програми (документа))

*Тексти програмного коду програмного забезпечення класичного судоку*

**main.py**

import time  
from config import\*  
from game import Game  
from screen import HomeScreen, EndScreen  
  
  
class Main:  
 def \_\_init\_\_(self):  
 self.\_minutes = 0  
 self.\_seconds = 0  
 self.\_start\_time = 0  
  
 @staticmethod  
 def \_write\_to\_file(board, mistakes=0, timer=()):  
 with open("sudoku\_file\_manager", 'a') as file:  
 if sum(sum(row) for row in board) < 405:  
 file.write("SUDOKU GAME:\n")  
 file.write("\tPUZZLE TO SOLVE:\n\n")  
 for row in board:  
 file.write('\t' \* 2 + ' '.join(map(str, row)) + '\n')  
 else:  
 file.write("\n\tSOLVED PUZZLE:\n\n")  
 for row in board:  
 file.write('\t' \* 2 + ' '.join(map(str, row)) + '\n')  
 file.write("\nMistakes: " + str(mistakes))  
 file.write("\nIn time " + str(timer[0]) + ":" + str(timer[1]) + "\n" \* 3)  
  
 def \_timer(self, stop=False):  
 if stop:  
 second = int((pygame.time.get\_ticks() - self.\_start\_time) / 1000)  
 self.\_minutes = second // 60  
 self.\_seconds = second - self.\_minutes \* 60  
 if self.\_seconds == 60:  
 self.\_seconds = 0  
 compound = "Timer: " + str(self.\_minutes).zfill(2) + ":" + str(self.\_seconds).zfill(2)  
 text = LOWER\_FONT.render(compound, True, COL\_BLACK)  
 SCREEN.blit(text, (WINDOW\_SIZE - MARGIN - WINDOW\_SIZE \* 0.17, WINDOW\_SIZE \* 0.925))  
 else:  
 return self.\_minutes, self.\_seconds  
  
 def game(self):  
 screen = "HOME"  
 run, playing = True, True  
 sudoku = Game()  
 home = HomeScreen()  
 end = EndScreen()  
 self.\_write\_to\_file(sudoku.game\_board)  
 time\_played = 0, 0  
  
 while playing:  
 if screen == "END":  
 time.sleep(3)  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 playing = False  
 if event.type == pygame.MOUSEBUTTONDOWN:  
 if end.button\_active:  
 self.\_start\_time += pygame.time.get\_ticks()  
 sudoku = Game()  
 self.\_write\_to\_file(sudoku.game\_board)  
 screen = "PLAY"  
 if run:  
 time.sleep(3)  
 SCREEN.fill(BACKGROUND\_COL\_WIGHT)  
 end.draw\_over(sudoku.mistakes, time\_played)  
 run = False  
  
 if screen == "PLAY":  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 playing = False  
 if event.type == pygame.MOUSEBUTTONDOWN:  
 mouse\_x, mouse\_y = pygame.mouse.get\_pos()  
 sudoku.find\_location(mouse\_x, mouse\_y)  
 if event.type == pygame.KEYDOWN:  
 if sudoku.mouse\_active:  
 sudoku.detect\_keys(event)  
 if event.key == pygame.K\_SPACE:  
 sudoku.solve(sudoku.game\_board)  
 if event.key == pygame.K\_BACKSLASH:  
 sudoku.detect\_keys(event, True)  
 SCREEN.fill(BACKGROUND\_COL\_WIGHT)  
 self.\_timer(True)  
 sudoku.draw\_game()  
 sudoku.draw\_mistakes()  
  
 if not sudoku.find\_empty(sudoku.game\_board):  
 time\_played = self.\_timer(False)  
 self.\_write\_to\_file(sudoku.game\_board, sudoku.mistakes, time\_played)  
 screen = "END"  
 sudoku.mouse\_active = False  
  
 if sudoku.mouse\_active:  
 sudoku.draw\_sel\_box()  
  
 if sudoku.key\_active:  
 sudoku.draw\_num()  
  
 if screen == "HOME":  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 playing = False  
 if event.type == pygame.MOUSEBUTTONDOWN:  
 if home.button\_active:  
 self.\_start\_time = pygame.time.get\_ticks()  
 screen = "PLAY"  
 SCREEN.fill(BACKGROUND\_COL\_WIGHT)  
 home.draw\_home()  
  
 pygame.display.update()  
 CLOCK.tick(30)  
  
  
Main().game()

**game.py**

import copy  
import random  
from config import\*  
  
  
class Game:  
 def \_\_init\_\_(self):  
 self.\_game\_board = Game.generate()  
 self.\_initial\_board = tuple(map(tuple, self.\_game\_board))  
 self.\_mouse\_active = False  
 self.\_key\_active = False  
 self.\_info = ''  
 self.\_mistakes = 0  
 self.\_hints = 0  
 self.\_selected\_col = 0  
 self.\_selected\_row = 0  
 self.\_location = [0, 0]  
  
 @staticmethod  
 def generate():  
 board = [[0 for i in range(9)] for j in range(9)]  
 row = random.randrange(9)  
 col = random.randrange(9)  
 num = random.randrange(1, 10)  
  
 for i in range(30):  
 while not Game.valid(board, num, (row, col)) or board[row][col] != 0:  
 row = random.randrange(9)  
 col = random.randrange(9)  
 num = random.randrange(1, 10)  
 board[row][col] = num  
  
 copy\_grid = copy.deepcopy(board)  
 if not Game.solve(copy\_grid):  
 board = Game.generate()  
  
 return board  
  
 @staticmethod  
 def valid(board, num, position):   
 for i in range(9):  
 if num == board[position[0]][i]:  
 return False  
 for i in range(9):  
 if num == board[i][position[1]]:  
 return False   
 for i in range((position[0] // 3) \* 3, (position[0] // 3) \* 3 + 3):  
 for j in range((position[1] // 3) \* 3, (position[1] // 3) \* 3 + 3):  
 if num == board[i][j]:  
 return False  
 return True  
  
 @staticmethod  
 def find\_empty(board):  
 for i in range(9):  
 for j in range(9):  
 if board[i][j] == 0:  
 return i, j   
 return False   
 @staticmethod  
 def solve(board):  
 if not Game.find\_empty(board):  
 return True  
 else:  
 position = Game.find\_empty(board)  
 for i in range(1, 10):  
 if Game.valid(board, i, position):  
 board[position[0]][position[1]] = i   
 if Game.solve(board):  
 return True   
 board[position[0]][position[1]] = 0  
 return False   
  
 @staticmethod  
 def find\_best\_indexes(board):  
 empty\_cells = []  
  
 for row in range(9):  
 for col in range(9):  
 if board[row][col] == 0:  
 options = 0  
  
 for num in range(1, 10):  
 if Game.is\_valid(board, num, row, col):  
 options += 1  
  
 empty\_cells.append((row, col, options))  
  
 empty\_cells.sort(key=lambda x: (x[2]))  
 return empty\_cells  
  
 @staticmethod  
 def is\_valid(board, num, row, col):  
 for c in range(9):  
 if c != col and board[row][c] == num:  
 return False  
  
 for r in range(9):  
 if r != row and board[r][col] == num:  
 return False  
  
 start\_row = (row // 3) \* 3  
 start\_col = (col // 3) \* 3  
 for r in range(start\_row, start\_row + 3):  
 for c in range(start\_col, start\_col + 3):  
 if (r != row or c != col) and board[r][c] == num:  
 return False  
 return True  
  
 @property  
 def game\_board(self):  
 return self.\_game\_board  
  
 @property  
 def mouse\_active(self):  
 return self.\_mouse\_active  
  
 @mouse\_active.setter  
 def mouse\_active(self, mouse\_active):  
 self.\_mouse\_active = mouse\_active  
  
 @property  
 def key\_active(self):  
 return self.\_key\_active  
  
 @property  
 def mistakes(self):  
 return self.\_mistakes  
  
 def draw\_game(self):  
 increment = MARGIN  
 for i in range(SECTION + 1):   
 pygame.draw.line(SCREEN, COL\_BLACK, (MARGIN, increment), (WINDOW\_SIZE - MARGIN, increment), 3)   
 pygame.draw.line(SCREEN, COL\_BLACK, (increment, MARGIN), (increment, WINDOW\_SIZE - MARGIN), 3)   
 increment += SECTION\_SIZE  
  
 increment = MARGIN  
 for i in range(SQUARE\_NUM + 1):   
 pygame.draw.line(SCREEN, COL\_BLACK, (MARGIN, increment), (WINDOW\_SIZE - MARGIN, increment))   
 pygame.draw.line(SCREEN, COL\_BLACK, (increment, MARGIN), (increment, WINDOW\_SIZE - MARGIN))   
 increment += SQUARE\_SIZE  
  
 increment\_x = increment\_y = MARGIN + SQUARE\_SIZE // 2  
  
 for row in range(9):  
 for col in range(9):   
 if self.\_game\_board[row][col] != 0 and self.\_game\_board[row][col] != self.\_initial\_board[row][col]:  
 text = FONT.render(str(self.\_game\_board[row][col]), True, COL\_BLACK)  
 text\_rect = text.get\_rect()  
 text\_rect.center = (increment\_x, increment\_y)  
 SCREEN.blit(text, text\_rect)  
 elif self.\_initial\_board[row][col] != 0:  
 text = FONT.render(str(self.\_game\_board[row][col]), True, (0, 0, 255))  
 text\_rect = text.get\_rect()  
 text\_rect.center = (increment\_x, increment\_y)  
 SCREEN.blit(text, text\_rect)  
  
 increment\_x += SQUARE\_SIZE  
 increment\_x = MARGIN + SQUARE\_SIZE // 2  
 increment\_y += SQUARE\_SIZE  
 text = FONT.render("Press space-bar to solve", True, COL\_BLACK)  
 text2 = FONT.render("Press back-slash to show hint", True, COL\_BLACK)  
 SCREEN.blit(text, (MARGIN + 115, WINDOW\_SIZE \* 0.0155))  
 SCREEN.blit(text2, (MARGIN + 82, WINDOW\_SIZE \* 0.06))  
  
 def find\_location(self, mouse\_x, mouse\_y):   
 self.\_selected\_col = int((mouse\_x - MARGIN) // SQUARE\_SIZE)   
 self.\_selected\_row = int((mouse\_y - MARGIN) // SQUARE\_SIZE)   
 self.\_location = (MARGIN + (self.\_selected\_col \* int(SQUARE\_SIZE)),  
 MARGIN + (self.\_selected\_row \* int(SQUARE\_SIZE)))   
 if (mouse\_x > MARGIN) and (mouse\_x < WINDOW\_SIZE - MARGIN) and (mouse\_y > MARGIN) and \  
 (mouse\_y < WINDOW\_SIZE - MARGIN) and self.\_game\_board[self.\_selected\_row][self.\_selected\_col] == 0:  
 self.\_info = ''  
 self.mouse\_active = True  
 else:  
 del self.\_selected\_row  
 del self.\_selected\_col  
 del self.\_location  
 self.mouse\_active = False  
 self.\_key\_active = False  
  
 def draw\_sel\_box(self):  
 pygame.draw.rect(SCREEN, SELECT\_COL\_LIGHT\_GREEN, (self.\_location[0], self.\_location[1],  
 SQUARE\_SIZE, SQUARE\_SIZE), 4)  
  
 def detect\_keys(self, info, hint=False):  
 if hint:  
 board\_copy = copy.deepcopy(self.\_game\_board)  
 indexes = Game.find\_best\_indexes(board\_copy)[0]  
 self.\_selected\_row, self.\_selected\_col, \_ = indexes  
 for i in range(1, 10):  
 self.\_info = str(i)  
 self.\_finalize\_key(True)  
  
 if info.unicode in NUMBERS:  
 self.\_key\_active = True  
 self.\_info = info.unicode   
 elif info.key == 13 and self.\_info in NUMBERS:  
 self.\_finalize\_key()  
 else:  
 self.\_key\_active = False  
  
 def draw\_num(self):  
 text = FONT.render(self.\_info, True, COL\_BLACK)  
 text\_rect = text.get\_rect()  
 text\_rect.center = (self.\_location[0] + SQUARE\_SIZE // 2, self.\_location[1] + SQUARE\_SIZE // 2)  
 SCREEN.blit(text, text\_rect)  
  
 def \_finalize\_key(self, hint=False):  
 board\_copy = copy.deepcopy(self.\_game\_board)  
  
 board\_copy[self.\_selected\_row][self.\_selected\_col] = int(self.\_info)  
 if Game.valid(self.\_game\_board, int(self.\_info), (self.\_selected\_row, self.\_selected\_col)) \  
 and Game.solve(board\_copy):  
 self.\_game\_board[self.\_selected\_row][self.\_selected\_col] = int(self.\_info)  
 elif not hint:  
 self.\_mistakes += 1  
 self.mouse\_active = False  
 self.\_key\_active = False  
 self.\_info = ''  
  
 def draw\_mistakes(self):  
 text = LOWER\_FONT.render("Mistakes " + str(self.\_mistakes), True, COL\_BLACK)  
 SCREEN.blit(text, (MARGIN, WINDOW\_SIZE \* 0.925))

**screen.py**

from config import\*  
  
  
class Screen:  
 def \_\_init\_\_(self):  
 self.\_size\_x = WINDOW\_SIZE \* 0.4   
 self.\_button\_x = WINDOW\_SIZE // 2 - self.\_size\_x // 2   
 self.\_size\_y = WINDOW\_SIZE \* 0.1  
 self.\_button\_y = WINDOW\_SIZE \* 0.6 - self.\_size\_y // 2  
 self.\_main\_text = pygame.font.SysFont("Calibri", int(self.\_size\_x \* 0.5))  
 self.\_small\_text = pygame.font.SysFont("Calibri", int(self.\_size\_x \* 0.1))  
 self.\_button\_text = pygame.font.SysFont("Calibri", int(self.\_size\_x \* 0.12), 1.5)  
 self.\_button\_active = False  
 self.\_mouse = [0, 0]  
  
 @property  
 def button\_active(self):  
 return self.\_button\_active  
  
 def \_button(self):  
 self.\_mouse = pygame.mouse.get\_pos()  
  
 if (self.\_button\_x + self.\_size\_x) > self.\_mouse[0] > self.\_button\_x and \  
 (self.\_button\_y + self.\_size\_y) > self.\_mouse[1] > self.\_button\_y:   
 pygame.draw.rect(SCREEN, S\_BUTTON\_COL\_DARK\_BLUE,  
 (int(self.\_button\_x), int(self.\_button\_y), int(self.\_size\_x), int(self.\_size\_y)))  
 self.\_button\_active = True  
 else:   
 pygame.draw.rect(SCREEN, BUTTON\_COL\_LIGHT\_BLUE,  
 (int(self.\_button\_x), int(self.\_button\_y), int(self.\_size\_x), int(self.\_size\_y)))  
 self.\_button\_active = False  
  
  
class HomeScreen(Screen):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
  
 def draw\_home(self):  
 text = self.\_main\_text.render("Sudoku", True, COL\_BLACK)  
 text\_rect = text.get\_rect()  
 text\_rect.center = (WINDOW\_SIZE // 2, WINDOW\_SIZE \* 0.45)  
 SCREEN.blit(text, text\_rect)  
 text = self.\_small\_text.render("Created by Fedorieieva IP-23", True, COL\_BLACK)  
 SCREEN.blit(text, (WINDOW\_SIZE \* 0.025, WINDOW\_SIZE \* 0.925))  
  
 self.\_button()  
 text = self.\_button\_text.render("Generate Board!", True, BACKGROUND\_COL\_WIGHT)  
 text\_rect = text.get\_rect()  
 text\_rect.center = (WINDOW\_SIZE // 2, WINDOW\_SIZE \* 0.6)  
 SCREEN.blit(text, text\_rect)  
  
  
class EndScreen(Screen):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.\_button\_x = WINDOW\_SIZE // 2  
 self.\_main\_text = pygame.font.SysFont("Calibri", int(self.\_size\_x \* 0.4))  
 self.\_button\_text = pygame.font.SysFont("Calibri", int(self.\_size\_x \* 0.075), 1.5)  
  
 def draw\_over(self, mistakes, time):  
 text = self.\_main\_text.render("GAME OVER!", True, COL\_BLACK)  
 text\_rect = text.get\_rect()  
 text\_rect.center = (WINDOW\_SIZE // 2, WINDOW\_SIZE \* 0.35)  
 SCREEN.blit(text, text\_rect)  
 compound = "You finished with " + str(mistakes) + " mistakes and in the time " + \  
 str(time[0]).zfill(2) + ":" + str(time[1]).zfill(2)  
 text = FONT.render(compound, True, COL\_BLACK)  
 text\_rect = text.get\_rect()  
 text\_rect.center = (WINDOW\_SIZE // 2, WINDOW\_SIZE // 2)  
 SCREEN.blit(text, text\_rect)  
  
 self.\_button()  
 text = self.\_button\_text.render(" " \* 19 + "Generate New Game!", True, BACKGROUND\_COL\_WIGHT)  
 text\_rect = text.get\_rect()  
 text\_rect.center = (WINDOW\_SIZE // 2, WINDOW\_SIZE \* 0.6)  
 SCREEN.blit(text, text\_rect)

**config.py**

import pygame  
  
  
WINDOW\_SIZE = 700  
MARGIN = 75  
  
SQUARE\_NUM = 9  
SECTION = SQUARE\_NUM // 3  
  
SECTION\_SIZE = (WINDOW\_SIZE - MARGIN \* 2) / SECTION   
SQUARE\_SIZE = (WINDOW\_SIZE - MARGIN \* 2) / SQUARE\_NUM   
NUMBERS = ('1', '2', '3', '4', '5', '6', '7', '8', '9')  
  
COL\_BLACK = (0, 0, 0)  
SELECT\_COL\_LIGHT\_GREEN = (102, 205, 170)  
BUTTON\_COL\_LIGHT\_BLUE = (30, 144, 255)  
S\_BUTTON\_COL\_DARK\_BLUE = (24, 116, 205)  
BACKGROUND\_COL\_WIGHT = (255, 255, 255)  
  
FONT\_SIZE = int((WINDOW\_SIZE - MARGIN \* 2) \* 0.06)  
LOWER\_FONT\_SIZE = int(FONT\_SIZE \* 0.75)  
  
CLOCK = pygame.time.Clock()  
  
pygame.init()  
  
SCREEN = pygame.display.set\_mode((WINDOW\_SIZE, WINDOW\_SIZE))  
  
pygame.display.set\_caption("Sudoku Fedorieieva")  
  
FONT = pygame.font.SysFont("calibri", FONT\_SIZE)  
LOWER\_FONT = pygame.font.SysFont("calibri", LOWER\_FONT\_SIZE)