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 Основ Програмування

(назва дисципліни)

на тему: «Судоку»

Студентки

1 курсу,групи ІП-13

Лисенко Анастасія Олегівна

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. вик. Головченко М. М |
|  | (підпис) |  | (вчене звання, науковий ступінь, прізвище та ініціали) |
|  |  |  | к. т. н. доц. Муха І. П. |
|  | (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-13 Семестр 2

**ЗАВДАННЯ**

на курсову роботу студента

**Лисенко Анастасії Олегівни**

(прізвище, ім’я, по батькові)

1. Тема роботи

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

КАЛЕНДАРНИЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Назва етапів курсової роботи | Термін виконання етапів роботи | Підписи керівника, студента |
| 1. | Отримання теми курсової роботи | 12.02.2022 |  |
| 2. | Підготовка ТЗ | 16.02.2022 |  |
| 3. | Пошук та вивчення літератури з питань курсової роботи | 20.02.2022 |  |
| 4. | Розробка сценарію роботи програми | 21.02.2022 |  |
| 5. | Узгодження сценарію роботи програми з керівником | 24.02.2022 |  |
| 6. | Розробка (вибір) алгоритму рішення задачі | 01.04.2022 |  |
| 7. | Узгодження алгоритму з керівником | 06.04.2022 |  |
| 8. | Узгодження з керівником інтерфейсу користувача | 06.04.2022 |  |
| 9. | Розробка програмного забезпечення | 20.04.2022 |  |
| 10. | Налагодження розрахункової частини програми | 10.05.2022 |  |
| 11. | Розробка та налагодження інтерфейсної частини програми | 28.05.2022 |  |
| 12. | Узгодження з керівником набору тестів для контрольного прикладу | 29.05.2022 |  |
| 13. | Тестування програми | 01.06.2022 |  |
| 14. | Підготовка пояснювальної записки | 02.06.2022 |  |
| 15. | Здача курсової роботи на перевірку | 12.06.2022 |  |
| 16. | Захист курсової роботи | 17.06.2022 |  |

Студент

(підпис)

Керівник Головченко М. М.

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

«12» червня 2022 р.

**Анотація**

Пояснювальна записка до курсової роботи: 65 сторінок, 10 рисунків, 15 таблиць, 3 посилання.

Об’єкт дослідження: головоломка «Судоку».

Мета роботи: дослідження генерації та розв’язання головоломки «Судоку».

Робота присвячена головоломці «Судоку», її генерації та рішенню. Було розроблено алгоритм для створення повністю коректно заповненої таблиці для головоломки «Судоку» та детально описано його. На його основі було розроблено алгоритм генерації самого початкового поля гри головоломки.

Виконана програмна реалізація алгоритмів генерації та розв’язання головоломки «Судоку», розроблений зручний графічний інтерфейс для взаємодії з алгоритмами.

ГОЛОВОЛОМКА «СУДОКУ», ГЕНЕРАЦІЯ ГОЛОВОЛОМКИ «СУДОКУ», РІШЕННЯ ГОВОЛОМКИ «СУДОКУ»

**ЗМІСТ**

[**Вступ** 6](#_Toc105719320)

[1 **Постановка задачі** 7](#_Toc105719321)

[2 **Теоретичні відомості** 9](#_Toc105719322)

[3 **Опис алгоритмів** 11](#_Toc105719323)

[3.1. Загальний алгоритм 11](#_Toc105719324)

[3.2. Алгоритм генерації грального поля 12](#_Toc105719325)

[3.3. Алгоритм перевірки заповненості поля та перевірки корекності заповнення: 15](#_Toc105719326)

[3.4. Алгоритм виведення підказок 16](#_Toc105719327)

[3.5. Алгоритм збереження даних 17](#_Toc105719328)

[4 **Опис програмного забезпечення** 18](#_Toc105719329)

[4.1. Діаграма класів програмного забезпечення 18](#_Toc105719330)

[4.2. Опис методів частин програмного забезпечення 19](#_Toc105719331)

[4.2.1. Стандартні методи 19](#_Toc105719332)

[4.2.2. Користувацькі методи 22](#_Toc105719333)

[5 **План тестування** 28](#_Toc105719334)

[5.1. План тестування 28](#_Toc105719335)

[5.2. Приклади тестування 28](#_Toc105719336)

[6 **Інструкція користувача** 35](#_Toc105719337)

[6.1. Робота з програмою 35](#_Toc105719338)

[6.2. Формат вхідних та вихідних даних 39](#_Toc105719339)

[**Висновки** 41](#_Toc105719340)

[**Посилання** 42](#_Toc105719341)

[Додаток А Технічне завдання 43](#_Toc105719342)

[Додаток Б Тексти програмного коду 46](#_Toc105719343)

# **Вступ**

Головоломка «Судоку» - гра-головоломка, для вирішення якої від гравця очікується певний рівень логічного та критичного мислення, осмислення правил гри та їх повне розуміння. Мета гри – дозаповнити коректно згенероване ігрове поле розміром 9х9 клітинок числами від 1 до 9 так, щоб у кожному з дев’яти квадратів розміром 3х3, стовбці та рядку були числа від 1 до 9 без повторень. Ця головоломка має математичний характер, так як для розв’язання більш складних її версій треба вміти коректно працювати з числами та самостійно логічно виводити нові методи для рішення головоломки або опрацьовувати нові методи рішення з методичних матеріалів. Тому ця гра підійде для користувачів будь-якого віку, допоможе їм розвити логічне та критичне мислення та навчить роботі з числами.

Розроблене програмне забезпечення дає можливість як отримання автоматичного розв’язку створенної головоломки так і можливість користувачу самостійно заповнити згенеровану таблицю гри значеннями та спробувати себе в цій грі. Головною метою розробки цього програмного забезпечення – створення комфортного для гри та користування інтерфейсу, який дозволяє оптимально працювати з створеними алгоритмами. Під час розробки можна дослідити алгоритми розв’язку та генерації головоломки, покращити навички програмування, здобути практичних навичок розробки програмних забезпечень з графічним інтерфейсом у PyCharm.

В даній курсовій роботі детально розглянута теорія щодо головоломки «Судоку», яка буде потрібна для розуміння алгоритмів розробленої задачі та поставленної задачі. В цій пояснювальній записці буде описана головна ідея алгоритмів головоломки та покроковий її опис. Будуть описані усі використані класи та методи. Буде проведено тестування всіх алгоритмів програми на коректність їх роботи. Для комфортного розуміння та користування програмним забезпеченням буде описана інструкція користувача. По завершенню створення цієї записки та розробки програми будуть підбиті підсумки нашої роботи у висновку.

# **Постановка задачі**

Розробити програмне забезпечення, що буде генерувати випадкові розкладки чисел для гри в «Судоку» та буде дозволяти вирішувати їх на різних складностях:

а) легка(easy)

б) середня(medium)

в) складна(hard)

Вхідними даними для головоломки є вибрана складність, також вхідними даними є сукупність натуральних чисел, які лежать у межах від 1 до 9 і є розв’язуваною задачею головоломки «Судоку».

Якщо користувач захотів розв’язати головоломку власноруч, то програмне забезпечення повинно обробляти введені дані користувачем, а саме складність генеруємої головоломки, та розробляти розв’язну задачу, яка задовольнятиме вибрану складність, а потім обробляти розв’язок користувача для заданої головоломки.

Вихідними даними для даної головоломки буде сукупність цілих чисел у межах від 1 до 9, які є підказками для розв’язання згенерованої розкладки головоломки. Під час гри користувач може максимум отримати 3 підказки, якщо користувач хоче використати їх, то програмне забезпечення повинно базуючись на логічному алгоритмі відкрити одну з незаповнених клітинок необхідним для вирішення головоломки числом. За бажанням користувача, програмне забезпечення може автоматично вирішити згенеровану головоломку.

Якщо користувач повністю заповнив таблицю головоломки значеннями, після натискання користувача на кнопку графічного інтерфейсу, програмне забезпечення повинно вивести на екран повідомлення про правильність або неправильність введеного рішення, складність гри та кількість використаних підказок. За бажанням користувача, незакінчене рішення головоломки можна зберегти, програмне забезпечення повинно записати у текстовий файл прогрес розв’язання головоломки користувачем та дати йому можливість пізніше повернутися до гри.

Вихідними даними для данного випадку являється сукупність натуральних чисел від 1 до 9, що є розв’язком для згенерованої головоломки, який виводяться на екран. Також вихідними даними будуть збережені у текстовий файл код початкового поля гри(стрічка значень елементів початкової матриці гри) та код матриці рішень(стрічка значень елементів матриці рішень).

# **Теоретичні відомості**

2.1 Суть головоломки «Судоку»

Суть головоломки «Судоку»: Ігрове поле складається з квадрата, розміром 9х9, розділеного на менші квадрати із стороною 3х3 клітинки. Таким чином, все поле налічує 81 клітинку. У деяких з них вже на початку гри розташовані числа (від 1 до 9). Залежно від того, скільки клітинок вже заповнено, конкретні судоку можна віднести до легких або складних.

Мета головоломки — необхідно заповнити вільні клітинки цифрами від 1 до 9 так, щоб в кожному рядку, в кожному стовпці і в кожному малому квадраті 3×3, кожна цифра зустрічалася лише один раз. Вважається, що головоломка має одне рішення, проте зустрічаються судоку з помилками, чи спеціальні судоку з кількома варіантами розвитку.

На рисунку 2.1 зображене можливе стартове поле для Судоку розміром 9х9.



Рисунок 2.1 – Поле головоломки

Генерація головоломки буде починатися з повної генерації 1, 5 та 9 квадратів розміром 3х3 випадковими числами від 1 до 9 за умови, що числа не повторюються. Далі, базуючись на згенерованих квадратах, за допомогою алгоритма заповнюємо всі інші квадрати розміром 3х3. Після повної генерації поля ми рекурсивно видаляємо статичну кількість клітинок(залежить від вибраної складності) таким чином, щоб задача мала розв’язок і причому тільки один коректний. Для цього в алгоритмі створена окрема функція перевірки на одне можливе рішення початкової розкладки гри.

Для надання гравцю підказок, програма використовує алгоритм логічного рішення Судоку, рекурсивно знаходить найвакантніше місце для відкриття клітинки з правильною відповіддю. Перевірка на вакантність деякого числа від 1 до 9 на певну позицію в таблиці відбувається за допомогою перевірки квадрата розміром 3х3 на наявність в ньому шуканого числа. Після цього йде перевірка стовпця та рядка, в яких знаходиться шукана клітинка на наявність в них того самого числа. У разі негативного результату пошуку в усіх з вище виділених умов, клітинка вважається вакантною для деякого числа і відкривається нашому користувачу у якості підказки. У разі повного заповнення поля гри напівкоректними значеннями програма виділяє для себе коректно заповнені клітинки та не включає їх у розглядання на вакантність. У разі повністю коректного заповнення поля гри, натискання кнопки для видання підказок переведе користувача на сторінкау результатів гри.

Алгоритм закінчує свою роботу після повного заповнення гравцем поля гри і подальшого натискання їм кнопки графічного інтерфейсу «Check» («Перевірити»). Далі алгоритм робить перевірку на однаковість початково згенерованого повного поля та власне поля заповненого користувачем. За умови їх співпадання програма буде виводити привітання для користувача, у протилежному випадку виведеться повідомлення про програш та коректно заповнене поле гри. Алгоритм також закінчує свою роботу після натискання користувачем кнопки графічного інтерфейсу «Solve» («Вирішити»).

# **Опис алгоритмів**

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

Таблиця 3.1 – Основні змінні та їхні призначення

|  |  |
| --- | --- |
| Змінна | Призначення |
| mode | Складність згенерованого поля гри |
| game\_matrix[0] | Початкове згенероване поле для гри/Поточне поле для гри |
| game\_matrix[1] | Вирішене поле для гри |
| hint\_cells | Контейнер з позиціями підказок на полі |
| file\_insides | Ознака наявності збережених даних з минулої гри |

## Загальний алгоритм

1. ПОЧАТОК
2. ЯКЩО вихідний файл збереження має в собі дані про минулу генерацію поля(*file\_insides == True*), ТО перейти до заповнення пустого поля даними значеннями, ІНАКШЕ вибрати складність та перейти до алгоритму генерації грального поля(пункт 3.2).
3. ЯКЩО натиснута кнопка з назвою «*Solve*», ТО вивести на екран автоматично вирішену головоломку(початково згенеровану матрицю рішень *game\_matrix[1]*).
4. ЯКЩО натиснута кнопка з назвою «*Check*», ТО перейти до алгоритму перевірки заповненості поля *game\_matrix[0]* та перевірки корекності цього заповнення(пункт 3.3).
5. ЯКЩО натиснута кнопка з назвою «*Hint*», ТО перейти до алгоритму виведення підказок(пункт 3.4).
6. ЯКЩО натиснута кнопка з назвою «*Save*», ТО перейти до алгоритму збереження даних(пункт 3.5).
7. КІНЕЦЬ

## Алгоритм генерації грального поля

* + 1. Функція рекурсивного вирішення *solve.*

1. ПОЧАТОК
2. ЦИКЛ проходу по всім елементам поля (матриці значень *game\_matrix[0]*):
   1. ЯКЩО елемент дорівнює *0*, ТО*\_spacesAvailable* прирівняти до позицій цього елементу в матриці, ІНАКШЕ *\_spacesAvailable = False*
      1. ЯКЩО *\_spacesAvailable == False,* ТО ПОВЕРНУТИ *True*
      2. ЦИКЛ проходу по значенням від *1 до 9* ітератором *i*:
         1. ЯКЩО значення клітинки позиції зі *\_spacesAvailable* дорівнює *0* та клітинку зі значенням ітератор циклу *i* не можна знайти ні в стобці, ні в рядку, ні в квадраті(*3х3*), якому вона належить, ТО даному значенню клітинки таблиці присвоїти значення ітератора циклу:
            1. ЯКЩО функція рекурсивного вирішення(пункт 3.2.1.) == *True*, ТО ПОВЕРНУТИ поле (матрицю значень *game\_matrix[0]* ).
            2. Значення клітинки позиції *\_spacesAvailable* прирівняти до *0.*
3. ПОВЕРНУТИ *False*
4. КІНЕЦЬ
5. ПОЧАТОК
6. ЦИКЛ по всім елементам порожньої матриці розміром *9х9*:
   1. ЯКЩО індекс рядка та стовбчика є індексами квадратів головної діагоналі*(([0;2] [0;2]) ([3;5] [3;5]) ([6;8] [6;8]))*, ТО заповнити ці квадрати випадковими значеннями, які не повторюються від *1 до 9*.
7. ЦИКЛ проходу по всім елементам грального поля(матриці):
   1. ЯКЩО поточне значення елемента дорівнює *0*, ТО локальній змінні (*\_\_num*) присвоїти випадкове значення від *1 до 9*:
      1. ЯКЩО значення перевіряємої клітинки дорівнює *0* та клітинку зі значенням змінної *\_\_num* не можна знайти ні в стобці, ні в рядку, ні в квадраті(*3х3*), якому вона належить, ТО даному значенню клітинки таблиці присвоїти значення локальної змінної *\_\_num*:
         1. ЯКЩО Функція рекурсивного вирішення *solve* (пункт 3.2.1.) *== True,* ТО ПОВТОРИТИ з (підпункту 3 «ЦИКЛ проходу по всім елементам грального поля(матриці)») до того моменту, як матриця рішень буде повністю сформована.
8. ЯКЩО складність рівня гри *mode* == *«easy»*, ТО змінна *\_squares\_to\_remove* = *36*; ЯКЩО складність рівня гри *mode* == *«medium»*, ТО змінна *\_squares\_to\_remove* = *42*; ЯКЩО складність рівня гри *mode* == *«hard»*, ТО змінна *\_squares\_to\_remove* = *48*
9. ЯКЩО індекс рядка та стовбчика є індексами *([0;2] [0;2])*, ТО ПОКИ *\_counter<4,* видаляти з цього квадрату ненульові елементи. Після кінця циклу значення*\_counter* обнулити*.*
   1. Інкрементувати значення*\_counter.*
10. ЯКЩО індекс рядка та стовбчика є індексами *([3;5] [3;5])*, ТО ПОКИ *\_counter<4,* видаляти з цього квадрату ненульові елементи. Після кінця циклу значення*\_counter* обнулити*.*
    1. Інкрементувати значення*\_counter.*
11. ЯКЩО індекс рядка та стовбчика є індексами *([6;8] [6;8])*, ТО ПОКИ *\_counter<4,* видаляти з цього квадрату ненульові елементи. Після кінця циклу значення*\_counter* обнулити*.*
    1. Інкрементувати значення*\_counter.*
12. Значення змінної *\_squares\_to\_remove* зменшити на *12*.
13. ПОКИ *\_counter < \_squares\_to\_remove:*
    1. Змінним *\_row, \_col* присвоюємо випадкові значення від *0 до 8*.
    2. ЯКЩО значення в клітинки з позицією [*\_row, \_col*] != 0, ТО локальній змінній *n* присвоюємо значення цієї клітинки, а її саму обнуляємо.
       1. Задля перевірки на кількість рішень Судоку, виконуємо знаходження кількості всіх вільних клітинок(з нульовими значеннями) *\_z.*
       2. ЦИКЛ проходу по значенням від *1* до *\_z +1* ітератором *i:*
          1. Створюємо локальну копію матриці гри та прирівнюємо *\_k* до *1*.
          2. ЦИКЛ проходу по всім елементам матриці гри.
             1. ЯКЩО значення елементу == *0*, та *\_k* == *i*, ТО присвоїти *\_row, \_col* індекси елементу матриці.
             2. Інкрементувати значення \_k.
          3. ЦИКЛ проходу ітератором n по елементам значеннями від *1 до 9*:
             1. ЯКЩО значення перевіряємої клітинки з індексами *\_row, \_col* дорівнює *0* та клітинку зі значенням змінної *n* не можна знайти ні в стобці, ні в рядку, ні в квадраті(*3х3*), якому вона належить, ТО даному значенню клітинки таблиці присвоїти значення локальної змінної *n*

ЯКЩО Функція рекурсивного вирішення (пункт 3.2.1.) *== True,* ТО всі значення таблиці перевести в код(створити стрічку значень матриці) та додати до множини рішень початкової матриці, ІНАКШЕ присвоїти значенню таблиці *0*.

* + 1. ЯКЩО довжина множини рішень початкової матриці *!= 1*, ТО значенню клітинки матриці повернути минуле значення *n* та ПРОДОВЖИТИ видалення клітинок.
    2. Інкрементувати значення*\_counter.*

1. Зберегти значення початкової матриц і *game\_matrix[0]* та значення матриці рішень *game\_matrix[1]*.
2. КІНЕЦЬ.

## Алгоритм перевірки заповненості поля та перевірки корекності заповнення:

1. ПОЧАТОК
2. ЦИКЛ проходу по всім елементам поточної матриці гри.
   1. ЯКЩО елемент поточної матриці гри *== 0* (незаповнений), ТО додати позицію цього елементу до лісту незаповнених позицій.
3. ЯКЩО кількість цих позицій *!= 0* , ТО виділити ці клітинки для користувача як незаповнені, ІНАКШЕ почати перевірку коректності заповнення *game\_matrix[0]*:
   1. ЯКЩО введені значення поточного поля гри *game\_matrix[0] ==* матриці рішень *game\_matrix[1], ТО* вивести повідомлення про виграш користувача, ІНАКШЕ вивести повідомлення про програш.
4. КІНЕЦЬ

## Алгоритм виведення підказок

1. ПОЧАТОК
2. Кількість підказок *hint\_amn* прирівняти до 0
3. ЯКЩО *hint\_amn* <3, ТО:
   1. ЦИКЛ проходу по всім елементам поточної матриці значень *game\_matrix[0]*:
      1. ЯКЩО елемент за поточним індексом = елементу за поточним індексом матриці рішень *game\_matrix[1]*, ТО додати це значення до матриці коректних значень, ІНАКШЕ додати *0* та додати позицію цього елементу до ліста коректних порожніх клітинок.
      2. ЯКЩО матриця коректних значень повністю співпадає з матрицею рішень *game\_matrix[1]*, ТО вивести повідомлення про перемогу користувача(пункт 3.3 (підпункт 3.3)) та ЗАКІНЧИТИ роботу знаходження підказок.
      3. ЦИКЛ проходу по всіх коректних порожніх клітинках:
         1. Присвоєння локальним змінним *\_row, \_col* значення позиції поточної порожньої клітинки.
         2. ЦИКЛ проходу по ітератору *i* по елементам від *1 до 9*:
            1. ЯКЩО значення поточної порожньої клітинки дорівнює *0* та клітинку зі значенням ітератору *i* циклу не можна знайти ні в стобці, ні в рядку, ні в квадраті(3х3), якому вона належить, ТО дане значення клітинки додати до локального ліста *num\_amn.*
         3. ЯКЩО кількість елементів в *num\_amn >1* АБО *num\_amn = 0,* ТО *num\_amn* обнулити.
         4. ЯКЩО кількість елементів в *num\_amn =1,* ТО поточній матриці поля гри game\_matrix[0] за позицією *\_row, \_col* присвоїти *num\_amn[0];* Кількість підказок *hint\_amn* збільшити на одиницю, Поточну позицію *\_row, \_col* видалити з ліста коректних порожніх клітинок та ПОВЕРНУТИ *\_row, \_col* як позиції шуканої підказки.
4. Вивести підказку за позицією *\_row, \_col* на екран та додати її до ліста позицій *hint\_cells*.
5. КІНЕЦЬ

## Алгоритм збереження даних

1. ПОЧАТОК
2. Запис до файлу коду початкової матриці, коду матриці рішень *game\_matrix[1]* код поточної матриці *game\_matrix[0]*(стрічка значень елементів матриці), рівня складності *mode*, кількість використаних підказок та код позицій підказок *hint\_cells*(стрічка значень позицій підказок). Запис до цього файлу відбувається під час натискання кнопки з назвою *“Save”.* Початкова матриця та матриця рішень *game\_matrix[1]* записується в ще один файл в обов’язковому порядку під час генерації поля гри.
3. Під час натискання кнопок з назвою *“Solve”, “Check”* при повністю заповненному полі та *“Hint”* під час повністю коректно заповненому полі файл збереження очищається.
4. КІНЕЦЬ

# **Опис програмного забезпечення**

## Діаграма класів програмного забезпечення

Діаграма класів програмного забезпечення зображена на рисунку 4.1:

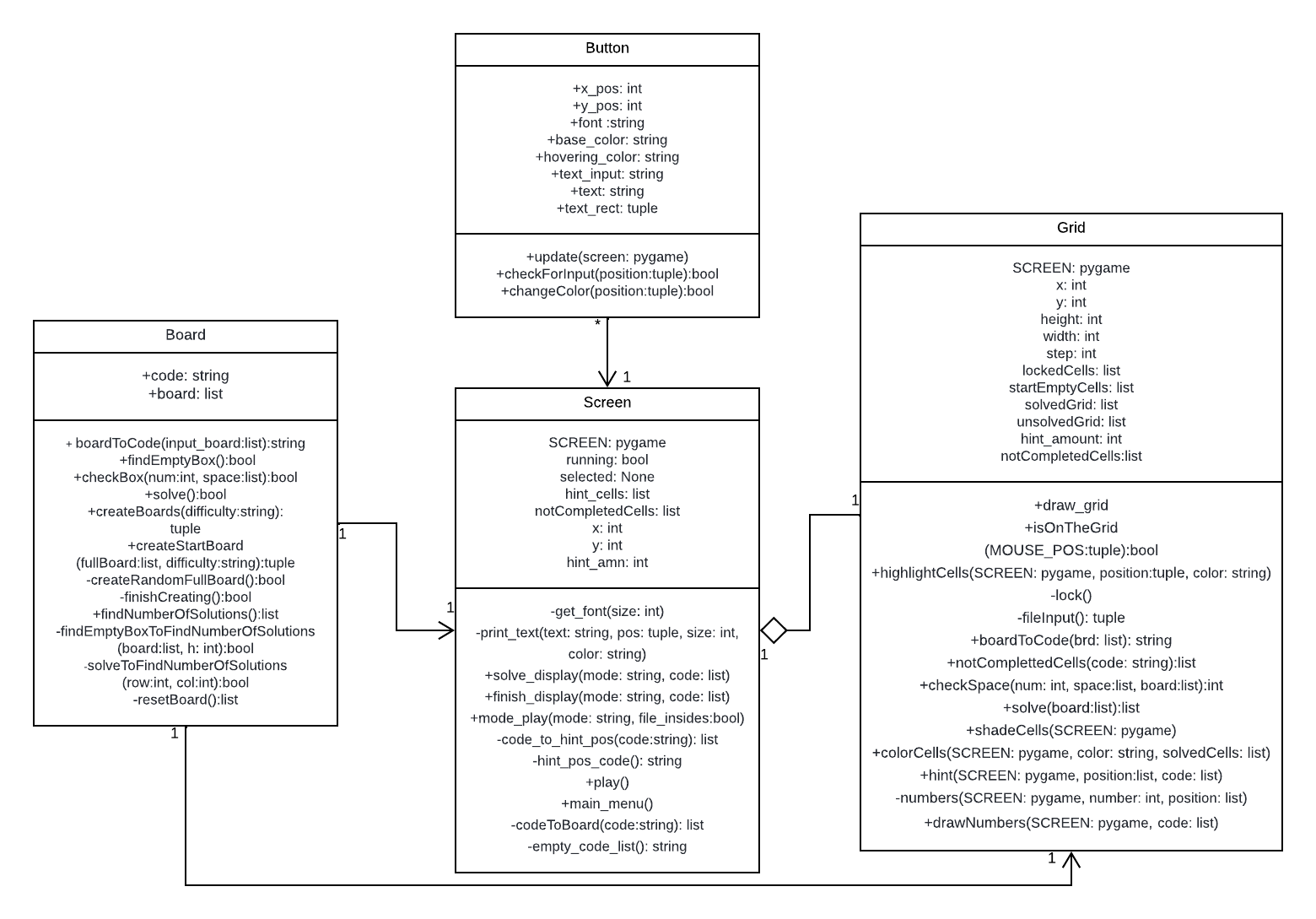


Рисунок 4.1 – Діаграма класів

У моїй програмі наявно 4 класи. Головний клас –Screen, який має в собі функції ,які відповідають за запуск вікон. Клас Grid слугує для відображення гріду поля та роботи з виводом чисел, підказок на екран. Також він слугує для графічного виділення клітинки. Клас Grid напряму використовується в основному класі Screen та об’єкт цього класу є атрибутом класу Screen.

Клас Board є основним класом логіки програми, саме він відповідає за генерацію, головоломки, яка вирішується. Результат функції цього класу використовується в класі Screen для генерації початкового поля.

Клас Button слугує для створення інтерактивних кнопок, він використовується n-ну кількість разів в основному класі Screen. Між собою клас Board та клас Grid залежні, бо без матриці значень поля гри не можна вивести ті самі значення на екран.

## Опис методів частин програмного забезпечення

### Стандартні методи

У таблиці 4.1 наведено стандартні методи, які були використані для реалізації поставленої задачі.

Таблиця 4.1 – Стандартні методи

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 1 | list | append | Додає елемент до списку | Список, значення | - | builtins.py |
| 2 | - | open | Відкриває файл або потік | Файл або потік, його зміст | - | builtins.py |
| 3 | IO | write | Записує дані в файл | Файл, дані для запису | Файл з записаними даними | typing.py |
| 4 | - | len | Знаходить довжину колекції або стрічки | Колекція або стрічка | Довжина стрічки або колекції | builtins.py |
| 5 | Random | choice | Вибирає випадковий елемент з колекції | Колекція | Елемент колекції | random.py |
| 6 | Random | randint | Вибирає випадковий елемент з проміжку | Ліміти для генерації | Число з цього ліміту | random.py |

Продовження таблиці 4.1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 7 | str | join | З’єднує задані елементи через сепаратор | Колекція стрічок | Суцільна стрічка | builtins.py |
| 8 | copy | deepcopy | Виконує глибоку копію об’єкту | Об’єкт | Копія об’єкту | copy.py |
| 9 | list | remove | Видаляє першу зустріч об’єкту в колекції | Колекція | - | builtins.py |
| 10 | IO | readlines | Зчитує стрічки початкового файлу | Файл | Колекція стрічок даних файлу | typing.py |
| 11 | IO | truncate | Очищає файл, або видаляє деяку кількість даних | Файл, розмір даних на видалення | - | typing.py |
| 12 | UserString | isdigit | Перевіряє, чи є символ числом | Число | Булеве значення | \_\_init\_\_.py |
| 13 | os | stat | Проводить статистичний запит за заданим шляхом | Шлях до файлу | - | nt.py |

Продовження таблиці 4.1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № п/п | | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 14 | | Font | render | Створює блок тексту | Вхідний текст,  значення, яке вказує чи треба пікселізація,  базовий колірЄ  вторинний колір | Об’єкт класу Pygame.text | pygame |
| 15 | Surface | | get\_rect | Створення прямокутника, в якому знаходиться текст | Позиція створення прямокутника | Об’єкт класу Pygame.rect | pygame |
| 16 | Rect | | left, right, bottom, top | Координати ліва, права, верху і низу прямокутника | - | Координати | pygame |
| 17 | pygame.draw | | rect | Малювання прямокутника | Колір, поверхня та об'єкт классу pygame.Rect | - | pygame |
| 18 | pygame.draw | | line | Малювання лінії | Поверхня, колір, стартова та кінцеві позиції | - | pygame |
| 19 | pygame.font | | Font | Ініціалізація шрифту | Шрифт та його розмір | - | pygame |

Продовження таблиці 4.1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 20 | pygame.mouse | get\_pos | Отримання координат курсора мишки | - | Координати х та у | pygame |
| 21 | pygame.event | get | Отримання подій з черги | - | Події в черзі | pygame |
| 22 | pygame.display | update | Оновлення частин екрану | Поле або весь екран | - | pygame |
| 23 | Surface | fill | Замалювання кольором | Колір та прямокутник для заливки | - | pygame |
| 24 | pygame | init | Ініціалізація pygame | - | - | pygame |
| 25 | pygame.display | set\_mode | Встановити розмір вікна | Розмір | - | pygame |
| 26 | pygame.display | set\_caption | Встановити назву програми | Назва | - | pygame |

### Користувацькі методи

У таблиці 4.2 наведено користувацькі методи, які були застосовані для реалізації задачі.

Таблиця 4.2 – Користувацькі методи

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 1 | Button | update | Відображає кнопку на екрані | Об’єкт класу та екран | - | button.py |

Продовження таблиці 4.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 2 | Button | checkForInput | Перевіряє, чи знаходиться курсор мишки над кнопкою | Об’єкт класу та позиція курсору мишки | Булеві значення | button.py |
| 3 | Button | changeColor | Змінює колір тексту кнопки при наведенні на неї курсору мишки. | Об’єкт класу та позиція курсору мишки | - | button.py |
| 4 | Grid | draw\_grid | Малює поле гри | Координати, колір | - | grid.py |
| 5 | Grid | isOnTheGrid | Перевіряє знаходження курсора мишки над полем гри | Координати курсора мишки | Булеве значення або позиція | grid.py |
| 6 | Grid | highlightCells | Виділяє іншим кольором обрану клітинку на полі гри | Екран, колекція позицій, колір | - | grid.py |
| 7 | Grid | lock | Створює 2 списки, де перший початково задані числа, які користувач не може змінювати, а друга- початково відкриті клітинки | Початкова матриця поля гри | Колекції заблокованих та вільних клітинок | grid.py |
| 8 | Grid | fileInput | Записує в текстовий файл початковий код гри та код рішення гри | Початкова матриця та матриця рішення | Код початкової матриці, матриці рішень | grid.py |

Продовження таблиці 4.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 9 | Grid | boardToCode | Перетворює значення матриці в код(суцільну стрічку чисел) | Матриця значень | Код матриці | grid.py |
| 10 | Grid | notComplettedCells | Перевіряє, чи користувач заповнив поле гри повністю | Матриця значень поля гри | Колекція вільних клітинок | grid.py |
| 11 | Grid | checkSpace | Перевіряє, чи на місце деякої клітинки підходить задане значення | Значення, позиція клітинки, матриця значень поля | Значення клітинки або нічого | grid.py |
| 12 | Grid | solve | Логічно дає підказку базуючись на даному полі гри | Матриця значень поля гри | Позиція шуканої клітинки або нічого | grid.py |
| 13 | Grid | shadeCells | Зафарбовує клітинки початкової розкладки сірим кольором | Екран, колекція позицій шуканих клітинок | - | grid.py |
| 14 | Grid | colorCells | Зафарбовує клітинки заданої колекції даним коьором | Екран, колір та колекція позицій клітинок | - | grid.py |
| 15 | Grid | hint | Зафарбовує клітинку-підказку зеленим кольором | Екран, значення підказки, її позиція | - | grid.py |

Продовження таблиці 4.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 16 | Grid | numbers | Ініціалізує значення клітинки числом з заданим шрифтом | Екран, число, позиція клітинки цього числа |  | grid.py |
| 17 | Grid | drawNumbers | Малює на екрані числа-значення поля гри | Матриця значень поля гри, екран | - | grid.py |
| 18 | Board | boardToCode | Перетворює матрицю значень поля гри на код(суцільну стрічку чисел) | Матриця значень поля гри | Код(суцільну стрічку чисел) | board.py |
| 19 | Board | findEmptyBox | Знаходить першу вільну клітинку(з нульовим значенням) у матриці поля гри | Матриця поля гри | Позиція знайденої клітинки | board.py |
| 20 | Board | checkBox | Перевіряє, чи на місце деякої клітинки підходить задане значення | Значення, позиція клітинки, матриця значень поля | Булеве значення | board.py |
| 21 | Board | solve | Вирішує головоломку використовуючи рекурсію | Матриця значень поля гри | Булеве значення | board.py |
| 22 | Board | createBoards | Створює колекцію матриць, яка складається з початкової матриці та матриці рішень | Складність рівня гри | Колекція матриць | board.py |

Продовження таблиці 4.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 23 | Board | createStartBoard | Створює початкову матрицю з матриці рішень | Матриця рішень та складність генерації | Колекція матриць рішення та початкової | board.py |
| 24 | Board | createRandomFullBoard | Генерує матрицю рішень | - | Матриця рішень | board.py |
| 25 | Board | finishCreating | Догенеровує матрицю рішень за допомогою рекурсії | - | Матриця рішень | board.py |
| 26 | Board | findNumberOfSolutions | Знайти кількість рішень для головоломки | Початкова матриця | Колекція кодів рішень | board.py |
| 27 | Board | findEmptyBoxToFindNumberOfSolutions | Знаходить першу вільну клітинку(з 0 значенням) у матриці, використовується при генерації початкової матриці | Матриця поля гри, номер шуканої вільної клітинки | Позиція шуканої вільної клітинки | board.py |
| 28 | Board | solveToFindNumberOfSolutions | Вирішує початкову матрицю, використовується для знаходження к-сті рішень | Матриця поля гри, позиція клітинки | Матриця рішень | board.py |
| 29 | Board | resetBoard | Очищає матрицю | - | Нульова матриця | board.py |

Продовження таблиці 4.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 30 | Screen | get\_font | Ініціалізує шрифт | Розмір | - | screen.py |
| 31 | Screen | print\_text | Виводить текст на екран | Текст, позиція, розмір, колір | - | screen.py |
| 32 | Screen | solve\_display | Відкриває вікно автоматичного вирішення | Складність, матриця | - | screen.py |
| 33 | Screen | finish\_display | Відкриває вікно результату | Складність, матриця | - | screen.py |
| 34 | Screen | mode\_play | Вікриває основне поле гри | Складність, наявність даних у файлі, файл виводу | - | screen.py |
| 35 | Screen | code\_to\_hint\_pos | Перетворює стрічку позицій підказок в коректні позиції | Стрічка файлу з кодом підказок | Колекція позицій підказок | screen.py |
| 36 | Screen | hint\_pos\_code | Перетворює позиції підказок в стрічку позицій | Позиції підказок | Стрічка підказок  (код) | screen.py |
| 37 | Screen | play | Відкриває вікно вибору складності | - | - | screen.py |
| 38 | Screen | main\_menu | Відкриває вікно основного меню | Файл виводу та його розмір | - | screen.py |
| 39 | Screen | codeToBoard | Перетворює код (стрічка значень матриці) в матрицю поля гри | Код(стрічка значень матриці) | Матриця поля гри | screen.py |
| 40 | Screen | empty\_code\_list | Генерує колекцію з двох порожніх матриць | - | Колекція з двох порожніх матриць | screen.py |

# **План тестування**

Для перевірки коректності роботи програми складемо план тестування і опрацюємо кожен пункт складеного плану. Результати тестування детально опишемо.

## План тестування

1. Тестування правильності введених значень.
   1. Тестування при введенні некоректних символів в поле гри.
   2. Тестування при введенні комбінацій клавіш в поле гри.
2. Тестування збереження даних гри:
   1. Тестування збереження початкової розкладки гри.
   2. Тестування збереження даних, які користувач вводив додатково.
   3. Тестування збереження підказок.
3. Тестування коректної роботи виведення підказок.
   1. Тестування логічності підказок.
   2. Тестування виводу підказок при повному некоректному заповненні поля гри.
   3. Тестування виводу підказок при повному коректному заповненні поля гри.
4. Тестування коректності генерації різних рівнів складності.
   1. Перевірка коректності генерації рівня складності «easy».
   2. Перевірка коректності генерації рівня складності «medium».
   3. Перевірка коректності генерації рівня складності «hard».

## Приклади тестування

Результати тестувань наведено у таблицях 5.1 – 5.11

Таблиця 5.1 ‑ Приклад роботи програми при введенні некоректних символів.

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість введення некоректних даних |
| Початковий стан програми | Відкрите вікно гри програми |
| Вхідні дані | s |
| Схема проведення тесту | Введення некоректних даних в поле матриці гри |
| Очікуваний результат | Клітинка не заповниться цими даними |
| Стан програми після проведення випробувань | Як і очікувано |

Таблиця 5.2 ‑ Приклад роботи програми при введенні комбінації клавіш.

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість введення комбінацій клавіш |
| Початковий стан програми | Відкрите вікно гри програми |
| Вхідні дані | Ctrl+Z |
| Схема проведення тесту | Введення комбінацій клавіш в поле матриці гри |
| Очікуваний результат | Клітинка не заповниться цими даними і програма не закриється |
| Стан програми після проведення випробувань | Як і очікувано |

Таблиця 5.3 ‑ Приклад роботи програми при збереженні початкових даних гри.

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість збереження початкової розкладки гри |
| Початковий стан програми | Відкрите вікно гри програми |
| Вхідні дані | Згенерована розкладка гри |
| Схема проведення тесту | Натискання на кнопку «Save» |
| Очікуваний результат | Після виходу з гри зберігатиметься початкова розкладка |

Продовження таблиці 5.3

|  |  |
| --- | --- |
| Стан програми після проведення випробувань | Як і очікувано |

Таблиця 5.4 ‑ Приклад роботи програми при збереженні додатково введених даних гри.

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість коректного збереження додатково введених даних користувачем |
| Початковий стан програми | Відкрите вікно гри програми |
| Вхідні дані | Згенерована розкладка гри і додатково введені дані |
| Схема проведення тесту | Натискання на кнопку «Save» попередньо ввівши додаткові дані |
| Очікуваний результат | Після виходу з гри зберігатиметься початкова збережена розкладка та зберігатимуться нові введені значення |
| Стан програми після проведення випробувань | Як і очікувано |

Таблиця 5.5 ‑ Приклад роботи програми при збереженні розкладки підказок.

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість збереження позицій наданих підказок та їх кількість |
| Початковий стан програми | Відкрите вікно гри програми |
| Вхідні дані | Згенерована розкладка гри і використані підказки |
| Схема проведення тесту | Натискання на кнопку «Save» попередньо використавши n-ну кількість підказок |

Продовження таблиці 5.5

|  |  |
| --- | --- |
| Очікуваний результат | Після виходу з гри зберігатимуться позиції використаних підказок, їх кількість |
| Стан програми після проведення випробувань | Як і очікувано |

Таблиця 5.6 ‑ Приклад роботи програми на логічність підказок, які виводяться на екран.

|  |  |
| --- | --- |
| Мета тесту | Перевірити логічність підказок |
| Початковий стан програми | Відкрите вікно гри програми |
| Вхідні дані | Згенерована розкладка гри і використані всі три підказки |
| Схема проведення тесту | Натискання на кнопку «Save» одразу після генерування розкладки гри, далі використання всіх 3 підказок, вихід з програми, продовження початої гри та знову використання 3 підказок |
| Очікуваний результат | Підказки будуть на одних і тих самих позиціях |
| Стан програми після проведення випробувань | Як і очікувано |

Таблиця 5.7 – Приклад виводу підказок при повному некоректному заповненні поля гри.

|  |  |
| --- | --- |
| Мета тесту | Перевірити логічність виводу підказок при повністю некоректно заповненому полю гри |
| Початковий стан програми | Відкрите вікно гри програми |
| Вхідні дані | Згенерована розкладка гри, повністю заповнене поле гри одиницями |

Продовження таблиці 5.7

|  |  |
| --- | --- |
| Схема проведення тесту | Натискання на кнопку «Hint» |
| Очікуваний результат | Підказки будуть з’являтися у логічному порядку, некоректні дані будуть замінені на коректні, якщо вони підпадають під логіку роботи підказки |
| Стан програми після проведення випробувань | Як і очікувано |

Таблиця 5.8 – Приклад виводу підказок при повному коректному заповненні поля гри.

|  |  |
| --- | --- |
| Мета тесту | Перевірити логічність виводу підказок при повністю коректно заповненому полю гри |
| Початковий стан програми | Відкрите вікно гри програми |
| Вхідні дані | Згенерована розкладка гри, повністю заповнене поле гри коректними даними |
| Схема проведення тесту | Натискання на кнопку «Hint» |
| Очікуваний результат | Програма буде переводити користувача на вікно фіналу гри |
| Стан програми після проведення випробувань | Як і очікувано |

Таблиця 5.9 – Приклад роботи алгоритму генерації рівня складності «easy».

|  |  |
| --- | --- |
| Мета тесту | Перевірити коректність генерації рівня складності «easy» |
| Початковий стан програми | Відкрите вікно гри програми |

Продовження таблиці 5.9

|  |  |
| --- | --- |
| Вхідні дані | Згенерована розкладка гри рівня «easy» |
| Схема проведення тесту | Вирішення головоломки та підрахунок вільних клітинок |
| Очікуваний результат | Коректна генерація головоломки та кількість вільних клітинок = 36 |
| Стан програми після проведення випробувань | Як і очікувано |

Таблиця 5.10 – Приклад роботи алгоритму генерації рівня складності «medium».

|  |  |
| --- | --- |
| Мета тесту | Перевірити коректність генерації рівня складності «medium» |
| Початковий стан програми | Відкрите вікно гри програми |
| Вхідні дані | Згенерована розкладка гри рівня «medium» |
| Схема проведення тесту | Вирішення головоломки та підрахунок вільних клітинок |
| Очікуваний результат | Коректна генерація головоломки та кількість вільних клітинок = 42 |
| Стан програми після проведення випробувань | Як і очікувано |

Таблиця 5.11 – Приклад роботи алгоритму генерації рівня складності «hard».

|  |  |
| --- | --- |
| Мета тесту | Перевірити коректність генерації рівня складності «hard» |
| Початковий стан програми | Відкрите вікно гри програми |
| Вхідні дані | Згенерована розкладка гри рівня «hard» |

Продовження таблиці 5.11

|  |  |
| --- | --- |
| Схема проведення тесту | Вирішення головоломки та підрахунок вільних клітинок |
| Очікуваний результат | Коректна генерація головоломки та кількість вільних клітинок = 48 |
| Стан програми після проведення випробувань | Як і очікувано |

# **Інструкція користувача**

## Робота з програмою

Після запуску виконавчого файлу, відкривається головне вікно програми(рисунок 6.1).



Рисунок 6.1 – Головне вікно програми

Натиснувши на кнопку з назвою «Quit», відбувається повне закриття програми.

Натиснувши на кнопку з назвою «Play», відкривається сторінка з вибором складності генеруємої задачі(рисунок 6.2). Якщо користувач натисне на кнопку з назвою «Back», відбудеться повернення до стартового меню(рисунок 6.1).



Рисунок 6.2 – Вибір необхідної складності гри

Натиснувши одну з кнопок з назвами: «Easy mode», «Medium mode», «Hard mode», генерується початкова розкладка головоломки з вибраною складністю, де «Easy mode» – легка складність, «Medium mode» – середня складність та «Hard mode» – тяжка складність(рисунок 6.3). Натиснувши на кнопку з назвою «Back» користувач повернеться до початкового меню(рисунок 6.1).

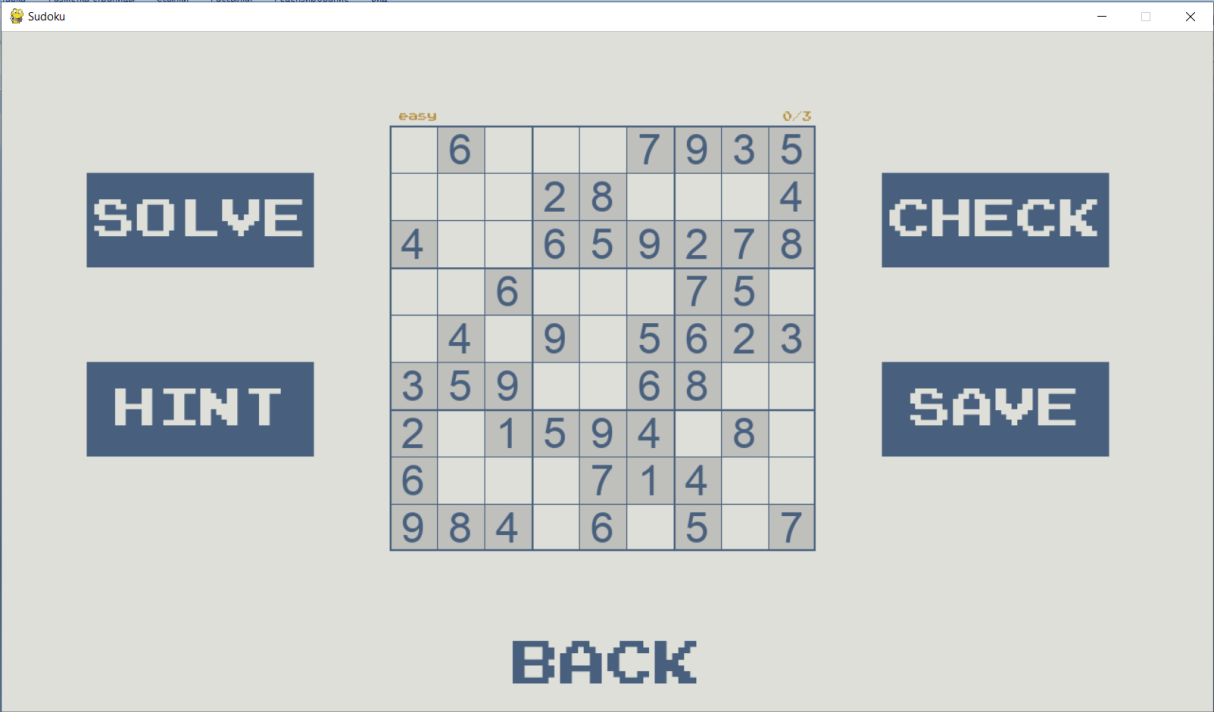


Рисунок 6.3 – Вікно поля гри головоломки

Шляхом натиску на вільні клітинки і введенням числа з клавіатури необхідно виставляти числа, що є розв’язком до головоломки, де клавіша нуля стирає введені користувачем дані. Якщо натиснути кнопку з назвою «Hint», то в одній із незаповнених або неправильно заповнених клітинок головоломки з’явиться число зеленого кольору, яке є правильною відповіддю для даної клітинки (Рисунок 6.8). У користувача є можливість використати до 3 підказок, число використаних можна побачити над правим кутком основного поля гри(рисунок 6.4).

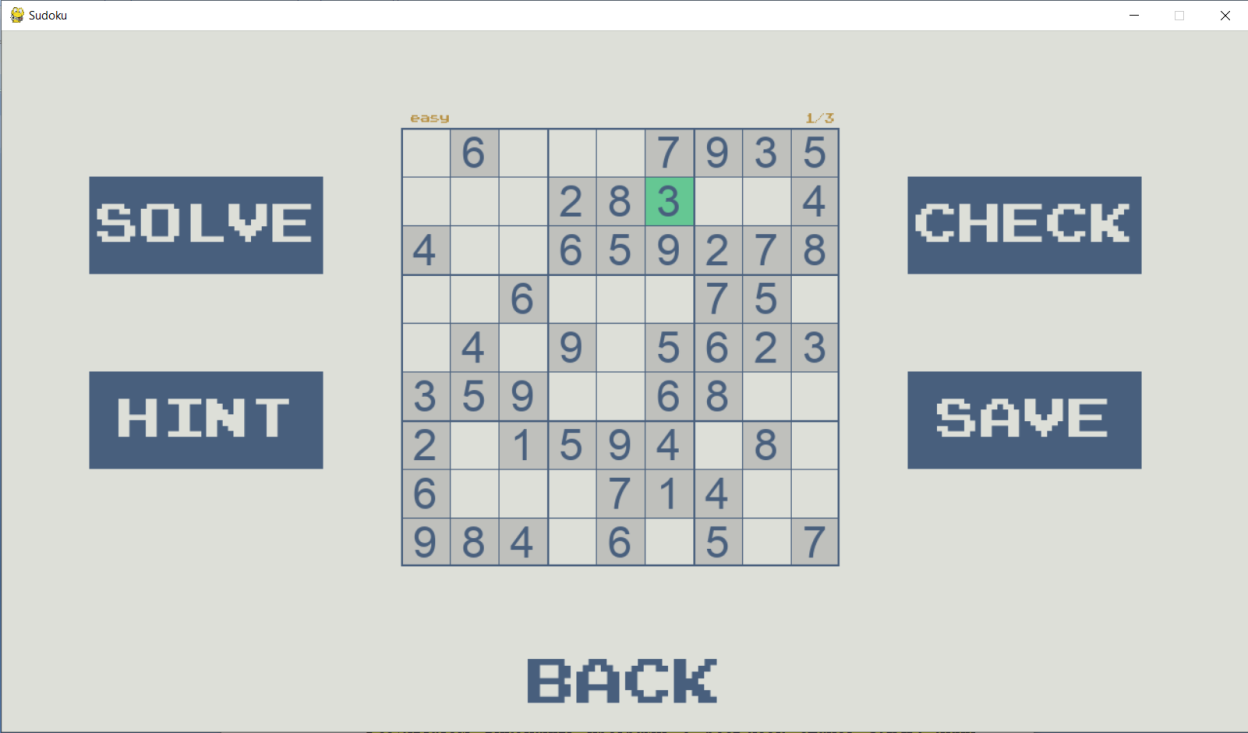


Рисунок 6.4 – Приклад використання підказки

Натиснувши на кнопку з назвою «Save», користувач може зберегти свій прогрес. При використанні кнопок з назвами «Solve» та «Check» на повністю заповненому полі збережений прогрес видалиться, бо користувач отримає свій результат.

Якщо користувач повернеться в стартове меню після зберігання, у нього з’явиться нова кнопка з назвою «Resume». За допомогою неї він зможе повернутися до раніше початої гри(рисунок 6.5). Поле гри буде виглядати стандартно(рисунок 6.3).

Якщо користувач натискає кнопку з назвою «Check» з неповністю заповненим полем гри, всі незаповнені клітинки зафарбуються червоним кольором(рисунок 6.6). Якщо ж поле заповнене повністю, користувач побачить фінальне вікно гри, де базуючись на коректності заповнення поля гри, він отримає повідомлення про перемогу(рисунок 6.7) або повідомлення про поразку(рисунок 6.8).

Text

Description automatically generated

Рисунок 6.5 – Створення кнопки “Resume”

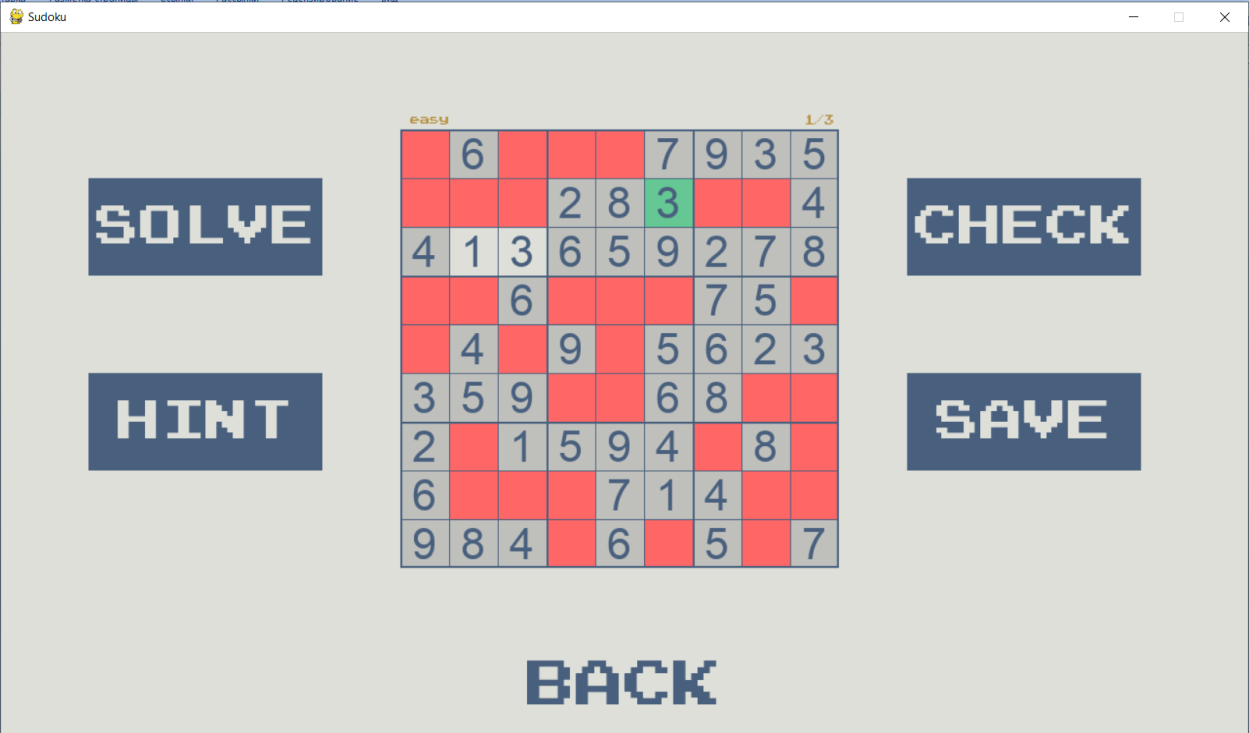


Рисунок 6.6 – Показ незаповнених клітинок



Рисунок 6.7 – Повідомлення про перемогу

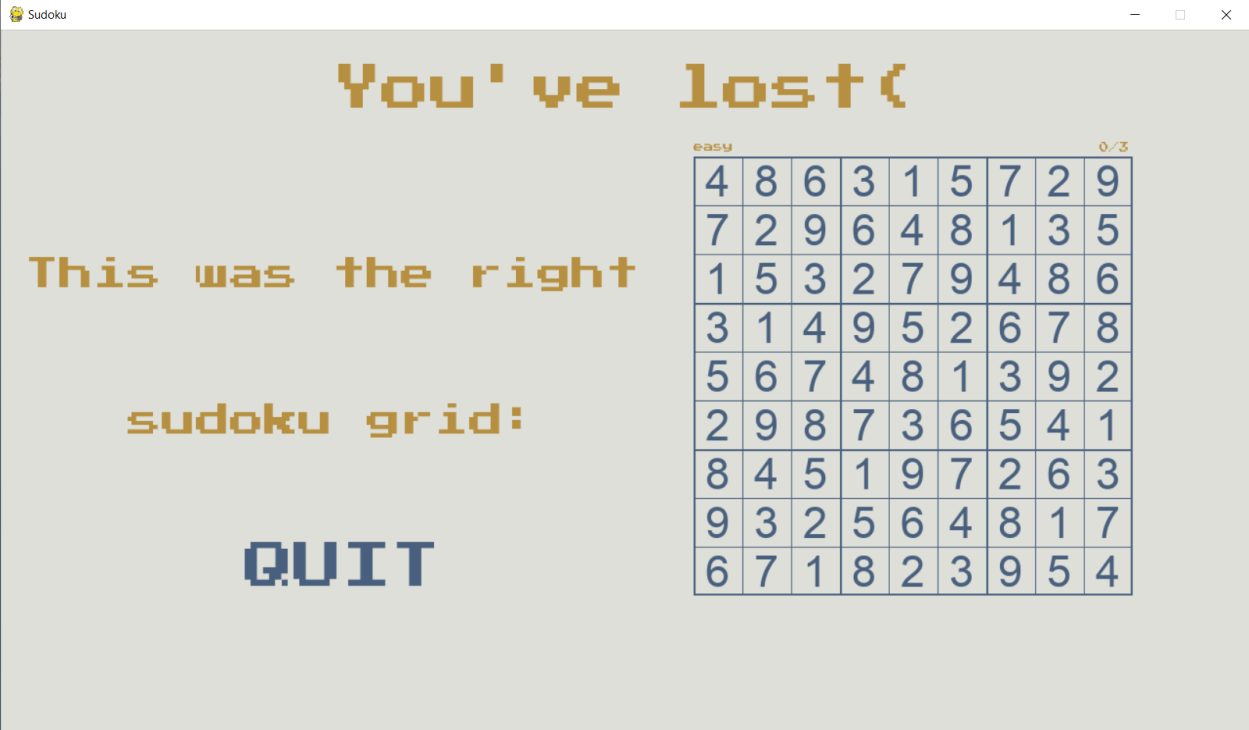


Рисунок 6.8 – Повідомлення про програш

На фінальному вікні гри, користувач може натиснути на кнопку з назвою «Quit», яка перенесе його на стартове вікно(рисунок 6.1).

Якщо користувач у вікні гри натисне на кнопку з назвою «Solve», він отримає автоматично вирішену головоломку(рисунок 6.9).



Рисунок 6.9 – Автоматично вирішена головоломка

Якщо користувач натисне на кнопку з назвою «Back», він повернеться до стартового вікна програми(рисунок 6.1).

## Формат вхідних та вихідних даних

Користувачем на вхід програми подається складність генерованого поля головоломки.

Результатом виконання програми є згенерована розв’язна говоломка та розв’зок до заданої головоломки.

## Системні вимоги

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 (з останніми обновленнями) | Windows 10  (з останніми обновленнями) |
| Процесор | Intel® Core® i5-4690K  3.50 GHz або AMD FX-8300 3.3 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 | 1280х720 | 1920х1080 або краще |
| Прилади введення | Клавіатура, комп’ютерна миша | |
| Додаткове програмне забезпечення | Python 3.8 і вище | |

# **Висновки**

По закінченню даної курсової можна підбити підсумки виконаної роботи.

Я досягла поставленної переді мною задачі, а саме написання алгоритму для генерації та рішення головоломки «Судоку», також створила комфортний та лаконічний дизайн для швидкої роботи з розробленими раніше алгоритмами. Також створене програмне забезпечення дотримується усіх вимог схваленних у технічному завданні.

Під час виконання даної курсової роботи я покращила свої знання у сфері використання мови Python та середовища розробки Pycharm. Отримала практичні навички створення програмного забезпечення, дотримуючись парадигм об’єктно-орієнтованого програмування, та навчилася на практиці розробці алгоритмів. Цей проект спонукав мене до покращення моїх навичок як розробника програмного забезпечення.

Під час створення цієї записки я детально описала кожен з зазначених розділів. Перший з них «Постановка задачі» описує вхідні та вихідні дані нашої програми та саме завдання. Другий розділ з назвою «Теоритичні відомості» несе в собі детально розписані правила гри головоломки «Судоку» та основні засади рішення його, також в ньому описані базові функції розробляємої програми. Третій за номером розділ «Опис алгоритмів» повністю описує використаний нами алгоритм, використовуючи псевдокод. «Опис програмного забезпечення», який в цій записці посідає четверту за порядком позицію, містить в собі мануально створену діаграму класів та таблицю методів, як стандартних, так і користувацьких, з їх описом. П’ятий за порядком розділ «План тестування» повністю описує процес тестування нашого програмного забезпечення, власне алгоритму та графічного інтерфейсу, з належними таблицями та повним його описом. Останній розділ «Інструкція користувача» детально розписує процес використання створеного програмного забезпечення гравцем та допомагає йому дізнатися про всі його можливості.

# **Посилання**

1. Правила гри та методи вирішення «Судоку»[Електронний ресурс] –

URL: <https://sudokutable.com/howtoplay/> (дата звернення: 10.04.2022).

1. Як вирішувати «Судоку» - алгоритми та стратегії»[Електронний ресурс] – URL: [https://peskiadmin.ru/](https://peskiadmin.ru/uk/kak-reshayutsya-sudoku-kak-reshat-sudoku-algoritmy-i-strategii.html) (дата звернення 20.04.2022).
2. Василенко С.Л. Числова гармонія «Судоку» – Науковий електронний ресурс «SciTecLibrary» – С. 1-8 – URL: [https://docplayer.com/](https://docplayer.com/32441088-Chislovaya-garmoniya-sudoku.html) (дата звернення 31.04.2022).

Додаток А Технічне завдання

КИЇВСЬКИЙПОЛІТЕХНІЧНИЙІНСТИТУТім. І. Сікорського

Кафедра

інформатики та програмної інженерії

Затвердив

Керівник:

Головченко Максим Миколайович

«24» лютого 2022 р.

Виконавець:

Студент Лисенко Анастасія Олегівна

«24» лютого 2022 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання курсової роботи

на тему: «Класичне судоку»

з дисципліни:

«Основи програмування»

Київ 2022

* 1. *Мета*: Метою курсової роботи є розробка ефективного програмного забезпечення у вигляді гри «Судоку» з можливістю вибору рівня складності гри.
  2. *Дата початку роботи*: «12» лютого 2022р.
  3. *Дата закінчення роботи*: «12» червня 2022 р.
  4. *Вимоги до програмного забезпечення*.

1. Функціональні вимоги:

* Можливість генерації задачі, яку можна вирішити, на ігровому полі розміром 9х9.
* Можливість вибору рівня складності гри(easy, medium, hard).
* Можливість розв’язання задачі автоматично за допомогою кнопки на полі графічного інтерфейсу.
* Можливість користувача заповнювати клітинки ігрового поля самостійно з клавіатури.
* Можливість користувача змінювати та видаляти значення клітинок за допомогою клавіш клавіатури та мишки.
* Можливість виводити підказки щодо заповнення клітинок за допомогою кнопки на полі графічного інтерфейсу.
* Можливість перевірки правильності рішення задачі та виведення результату на екран після повного заповнення поля гри.
* Можливість розпочати нову гру.
* Можливість продовжити почату раніше гру.

1. Нефункціональні вимоги:

* 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 06.04.2022 р.)
2. Об'єктно-орієнтоване проектування архітектури програмної системи (до 20.04.2022р.)
3. Розробка програмного забезпечення (до 28.05.2022р.)
4. Тестування розробленої програми (до 1.06.2022р.)
5. Розробка пояснювальної записки (до 11.06.2022 р.).
6. Захист курсової роботи (до 17.06.2022 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-13 1 курсу*

*Лисенко А. О.*

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

(Вид носія даних)

*Електронний носій*

(Найменування програми (документа))

*Тексти програмного коду програмного забезпечення вирішення та генерації головоломки «Судоку»*

**КОД**

**button.py**

# клас, який відповідає за створення кнопки  
**class Button:**  
 def \_\_init\_\_(self, pos, text\_input, font, base\_color, hovering\_color):  
 self.x\_pos = pos[0]  
 self.y\_pos = pos[1]  
 self.font = font  
 self.base\_color, self.hovering\_color = base\_color, hovering\_color  
 self.text\_input = text\_input  
 self.text = self.font.render(self.text\_input, True, self.base\_color)  
 self.text\_rect = self.text.get\_rect(center=(self.x\_pos, self.y\_pos))  
  
 # оновити кнопку на екрані  
 **def update**(self, screen):  
 screen.blit(self.text, self.text\_rect)  
  
 # перевірити чи курсор мишки над кнопкою  
 **def checkForInput**(self, position):  
 if position[0] in range(self.text\_rect.left, self.text\_rect.right) and position[1] in range(self.text\_rect.top, self.text\_rect.bottom):  
 return True  
 return False  
  
 # змінити колір кнопки при наведенні на неї  
 **def changeColor**(self, position):  
 if self.checkForInput(position):  
 self.text = self.font.render(self.text\_input, True, self.hovering\_color)  
 else:  
 self.text = self.font.render(self.text\_input, True, self.base\_color)

**grid.py**

import pygame, random  
from constants import BLUISH, BEIGE, GOLDEN, SHADE  
  
# клас, який відповідає за все, що відбувається на полі гри  
**class Grid**:  
 def \_\_init\_\_(self, SCREEN, numbers, x, y, hint\_amount=0):  
 self.SCREEN = SCREEN  
 self.x = x  
 self.y = y  
 self.height = 450  
 self.width = 450  
 self.step = 50  
 self.lockedCells = []  
 self.startEmptyCells = []  
 self.solvedGrid = numbers[1]  
 self.unsolvedGrid = numbers[0]  
 self.hint\_amount = hint\_amount  
 self.notCompletedCells = []  
 self.\_\_lock()  
 self.files = self.fileInput()  
  
 # намалювати порожнє поле гри  
 **def draw\_grid**(self):  
 pygame.draw.rect(self.SCREEN, BLUISH, pygame.Rect(self.x, self.y, self.height, self.width), 2)  
  
 for x in range(self.x + self.step, self.x + self.width - self.step + 1, self.step):  
 if x == self.x + 3 \* self.step or x == self.x + self.width - 3 \* self.step:  
 pygame.draw.line(self.SCREEN, BLUISH, (x, self.y), (x, self.y + self.height - 1), 2)  
 else:  
 pygame.draw.line(self.SCREEN, BLUISH, (x, self.y), (x, self.y + self.height - 1), 1)  
 for y in range(self.y + self.step, self.y + self.height - self.step + 1, self.step):  
 if y == self.y + 3 \* self.step or y == self.y + self.height - 3 \* self.step:  
 pygame.draw.line(self.SCREEN, BLUISH, (self.x, y), (self.x + self.width - 1, y), 2)  
 else:  
 pygame.draw.line(self.SCREEN, BLUISH, (self.x, y), (self.x + self.width - 1, y), 1)  
  
 # перевірити чи знаходиться курсор мишки над полем гри  
 **def isOnTheGrid**(self, MOUSE\_POS):  
 if MOUSE\_POS[0] in range(self.x, self.x + self.width) and MOUSE\_POS[1] in range(self.y, self.y + self.height):  
 return [(MOUSE\_POS[0] - self.x) // self.step, (MOUSE\_POS[1] - self.y) // self.step]  
 return False  
  
 # виділяти сірим кольором вибрану клітинку  
 **def highlightCells**(self, SCREEN, position, color):  
 pygame.draw.rect(SCREEN, color,  
 ((position[0] \* self.step) + self.x, (position[1] \* self.step) + self.y, self.step, self.step))  
  
 # стоврити колекцію початково заповнених клітинок та початково незаповнених  
 **def \_\_lock**(self):  
 for yidx, row in enumerate(self.unsolvedGrid):  
 for xidx, num in enumerate(row):  
 if num != 0:  
 self.lockedCells.append([xidx, yidx])  
 else:  
 self.startEmptyCells.append([xidx, yidx])  
  
 # запис у файл початкового поля та матриці рішень  
 **def fileInput**(self):  
 code = self.boardToCode(self.unsolvedGrid)  
 cd = self.boardToCode(self.solvedGrid)  
 return code, cd  
  
 # перетворення матриці у стрічку значень кожного елементу  
 **def boardToCode**(self, brd):  
 code = ''  
 for row in range(9):  
 for col in range(9):  
 code += str(brd[row][col])  
 return code  
  
 # створення лісту поточно незаповненних клітинок  
 **def notComplettedCells**(self, code):  
 self.notCompletedCells = []  
 for yidx, row in enumerate(code):  
 for xidx, num in enumerate(row):  
 if num == 0:  
 self.notCompletedCells.append([xidx, yidx])  
 return self.notCompletedCells  
  
 # функція перевірки на коректність деякого значення на деякій позиції  
 **def checkSpace**(self, num, space, board):  
 if not board[space[0]][space[1]] == 0:  
 return None  
  
 for col in board[space[0]]:  
 if col == num:  
 return None  
  
 for row in range(len(board)):  
 if board[row][space[1]] == num:  
 return None  
  
 \_internalBoxRow = space[0] // 3  
 \_internalBoxCol = space[1] // 3  
  
 for i in range(3):  
 for j in range(3):  
 if board[i + (\_internalBoxRow \* 3)][j + (\_internalBoxCol \* 3)] == num:  
 return None  
  
 return num  
  
 # функція логічного знаходження найбільш вакантної позиції для виводу підказки  
 **def solve**(self, board):  
 correctEmptyCells = []  
 correct\_board = []  
 for i in range(9):  
 lst = []  
 for j in range(9):  
 if board[i][j] == self.solvedGrid[i][j]:  
 lst.append(board[i][j])  
 else:  
 correctEmptyCells.append([j, i])  
 lst.append(0)  
 correct\_board.append(lst)  
 if correct\_board == self.solvedGrid:  
 return None  
 num\_amn = []  
 for i in range(len(correctEmptyCells)):  
 \_row, \_col = correctEmptyCells[i]  
 for n in range(1, 10):  
 if self.checkSpace(n, (\_col, \_row), correct\_board):  
 num\_amn.append(n)  
 if len(num\_amn) > 1 or len(num\_amn) == 0:  
 num\_amn = []  
 elif len(num\_amn) == 1:  
 board[\_col][\_row] = num\_amn[0]  
 self.hint\_amount += 1  
 correctEmptyCells.remove([\_row, \_col])  
 return [\_row, \_col]  
 '''elif len(num\_amn) == 0 and len(self.startEmptyCells) > 0:  
 rand = random.choice(self.startEmptyCells)  
 self.hint\_amount += 1  
 return rand'''  
  
 # зафарбування початково даних значень сірим кольором  
 **def shadeCells**(self, SCREEN):  
 for cell in self.lockedCells:  
 pygame.draw.rect(SCREEN, SHADE,  
 (cell[0] \* self.step + self.x, cell[1] \* self.step + self.y, self.step, self.step))  
  
 # зафарбування деяких клітинок деяким кольором  
 **def colorCells**(self, SCREEN, color, solvedCells):  
 for cell in solvedCells:  
 pygame.draw.rect(SCREEN, color,  
 (cell[0] \* self.step + self.x, cell[1] \* self.step + self.y, self.step, self.step))  
  
 # виділення позиції виводу підказки зеленим кольором  
 **def hint**(self, SCREEN, position, code):  
 for pos in position:  
 code[pos[1]][pos[0]] = self.solvedGrid[pos[1]][pos[0]]  
 self.unsolvedGrid[pos[1]][pos[0]] = self.solvedGrid[pos[1]][pos[0]]  
 pygame.draw.rect(SCREEN, "#65C793",  
 (pos[0] \* self.step + self.x, pos[1] \* self.step + self.y, self.step, self.step))  
  
 # ініціалізація розміру та шрифту цифр  
 **def \_\_numbers**(self, SCREEN, number, position):  
 fnt = pygame.font.Font("assets/arial.ttf", 45)  
 num = fnt.render(number, True, BLUISH)  
 SCREEN.blit(num, position)  
  
 # вивід на екран чисел  
 **def drawNumbers**(self, SCREEN, code):  
 for yidx, row in enumerate(code):  
 for xidx, num in enumerate(row):  
 if num != 0:  
 position = [xidx \* self.step + self.x + self.step // 4, yidx \* self.step + self.y]  
 self.\_\_numbers(SCREEN, str(num), position)

**board.py**

import random, copy

# клас, який відповідає за головний алгоритм гри та її логіку  
**class Board:**  
 def \_\_init\_\_(self, code=None):  
 self.\_\_resetBoard()  
  
 # перетворює матрицю значень на стрічку коду  
 **def boardToCode**(self, input\_board=None):  
 if input\_board:  
 \_code = ''.join([str(i) for j in input\_board for i in j])  
 return \_code  
 else:  
 self.code = ''.join([str(i) for j in self.board for i in j])  
 return self.code  
  
 # знаходить позицію першго нульового елементу матриці  
 **def findEmptyBox**(self):  
 for row in range(len(self.board)):  
 for col in range(len(self.board[0])):  
 if self.board[row][col] == 0:  
 return row, col  
  
 return False  
  
 # перевіряє чи деяке число(від 1 до 9) підходить на деяку позицію матриці  
 **def checkBox**(self, num, position):  
 if not self.board[position[0]][position[1]] == 0: # перевіряє чи клітинка вже заповнена  
 return False  
  
 for col in self.board[position[0]]: # перевіряє чи є це деяке число в рядку  
 if col == num:  
 return False  
  
 for row in range(len(self.board)): # перевіряє чи є це деяке число в колонці  
 if self.board[row][position[1]] == num:  
 return False  
  
 \_internalBoxRow = position[0] // 3  
 \_internalBoxCol = position[1] // 3  
  
 for i in range(3): # перевіряє чи є це деяке число в меншому квадраті 3х3  
 for j in range(3):  
 if self.board[i + (\_internalBoxRow \* 3)][j + (\_internalBoxCol \* 3)] == num:  
 return False  
  
 return True  
  
 # функція рекурсивного вирішення  
 **def solve**(self):  
 \_spacesAvailable = self.findEmptyBox()  
  
 if not \_spacesAvailable:  
 return True  
 else:  
 row, col = \_spacesAvailable  
 for n in range(1, 10):  
 if self.checkBox(n, (row, col)):  
 self.board[row][col] = n  
  
 if self.solve():  
 return self.board  
  
 self.board[row][col] = 0  
  
 return False  
  
 # функція генерації початкового поля гри та матриці рішень до нього  
 **def createBoards**(self, difficulty):  
 self.board, \_fullBoard = self.createStartBoard(self.\_\_createRandomFullBoard(), difficulty)  
 return self.board, \_fullBoard  
  
 # видаляє з повністю заповненої таблиці деяку кількість значень клітинок, базуючись на вибраній складності  
 **def createStartBoard**(self, fullBoard, difficulty):  
 self.board = copy.deepcopy(fullBoard)  
  
 if difficulty == "easy":  
 \_squares\_to\_remove = 36  
 elif difficulty == "medium":  
 \_squares\_to\_remove = 42  
 elif difficulty == "hard":  
 \_squares\_to\_remove = 48  
 else:  
 return  
  
 \_counter = 0  
 while \_counter < 4:  
 \_rRow = random.randint(0, 2)  
 \_rCol = random.randint(0, 2)  
 if self.board[\_rRow][\_rCol] != 0:  
 self.board[\_rRow][\_rCol] = 0  
 \_counter += 1  
  
 \_counter = 0  
 while \_counter < 4:  
 \_rRow = random.randint(3, 5)  
 \_rCol = random.randint(3, 5)  
 if self.board[\_rRow][\_rCol] != 0:  
 self.board[\_rRow][\_rCol] = 0  
 \_counter += 1  
  
 \_counter = 0  
 while \_counter < 4:  
 \_rRow = random.randint(6, 8)  
 \_rCol = random.randint(6, 8)  
 if self.board[\_rRow][\_rCol] != 0:  
 self.board[\_rRow][\_rCol] = 0  
 \_counter += 1  
  
 \_squares\_to\_remove -= 12  
 \_counter = 0  
 while \_counter < \_squares\_to\_remove:  
 \_row = random.randint(0, 8)  
 \_col = random.randint(0, 8)  
  
 if self.board[\_row][\_col] != 0:  
 n = self.board[\_row][\_col]  
 self.board[\_row][\_col] = 0  
  
 if len(self.findNumberOfSolutions()) != 1:  
 self.board[\_row][\_col] = n  
 continue  
  
 \_counter += 1  
  
 return self.board, fullBoard  
  
 # генерує повністю коректно заповену таблицю значень поля гри  
 **def \_\_createRandomFullBoard**(self):  
 self.\_\_resetBoard()  
  
 \_l = list(range(1, 10))  
 for row in range(3):  
 for col in range(3):  
 \_num = random.choice(\_l)  
 self.board[row][col] = \_num  
 \_l.remove(\_num)  
  
 \_l = list(range(1, 10))  
 for row in range(3, 6):  
 for col in range(3, 6):  
 \_num = random.choice(\_l)  
 self.board[row][col] = \_num  
 \_l.remove(\_num)  
  
 \_l = list(range(1, 10))  
 for row in range(6, 9):  
 for col in range(6, 9):  
 \_num = random.choice(\_l)  
 self.board[row][col] = \_num  
 \_l.remove(\_num)  
 return self.\_\_finishCreating()  
  
 # функція рекурсивного догенерування повністю заповненої таблиці  
 **def \_\_finishCreating**(self):  
 for row in range(len(self.board)):  
 for col in range(len(self.board[row])):  
 if self.board[row][col] == 0:  
 \_num = random.randint(1, 9)  
  
 if self.checkBox(\_num, (row, col)):  
 self.board[row][col] = \_num  
  
 if self.solve():  
 self.\_\_finishCreating()  
 return self.board  
  
 self.board[row][col] = 0  
  
  
 return False  
  
 # знаходить кількість рішень для деякої напівпорожньої матриці та повертає множину цих значень  
 **def findNumberOfSolutions**(self):  
 \_z = 0  
 \_list\_of\_solutions = []  
  
 for row in range(len(self.board)):  
 for col in range(len(self.board[row])):  
 if self.board[row][col] == 0:  
 \_z += 1  
  
 for i in range(1, \_z + 1):  
 \_board\_copy = copy.deepcopy(self)  
  
 \_row, \_col = self.\_\_findEmptyBoxToFindNumberOfSolutions(\_board\_copy.board, i)  
 \_board\_copy\_solution = \_board\_copy.\_\_solveToFindNumberOfSolutions(\_row, \_col)  
  
 \_list\_of\_solutions.append(self.boardToCode(input\_board=\_board\_copy\_solution))  
 if len(list(set(\_list\_of\_solutions))) > 1:  
 break  
  
 return list(set(\_list\_of\_solutions))  
  
 # знаходить деякий за номером нульовий елемент матриці  
 **def \_\_findEmptyBoxToFindNumberOfSolutions**(self, board, num):  
 \_k = 1  
 for row in range(len(board)):  
 for col in range(len(board[row])):  
 if board[row][col] == 0:  
 if \_k == num:  
 return (row, col)  
  
 \_k += 1  
  
 return False  
  
 # вирішує деякую напівпорожню матрицю задля знаходження кількості її розв'язків  
 **def \_\_solveToFindNumberOfSolutions**(self, row, col):  
 for n in range(1, 10):  
 if self.checkBox(n, (row, col)):  
 self.board[row][col] = n  
  
 if self.solve():  
 return self.board  
  
 self.board[row][col] = 0  
  
 return False  
  
 # повністю обнуляє матрицю  
 **def \_\_resetBoard**(self):  
 self.board = [[0 for \_ in range(9)] for \_ in range(9)]  
 return self.board

**screen.py**

from board import \*  
from button import \*  
from grid import \*  
import os, sys  
vector = pygame.math.Vector2  
  
  
# клас, який відповідає за графічний інтерфейс та з'єднує логіку гри з інтерфейсом  
**class Screen**:  
 def \_\_init\_\_(self, screen):  
 self.SCREEN = screen  
 self.running = True  
 self.selected = None  
 self.hint\_cells = []  
 self.notCompletedCells = []  
 self.x = 410  
 self.y = 100  
 self.hint\_amn = 0  
  
 @staticmethod  
 **def \_\_get\_font**(size): # ініціалізація шрифту за деяким розміром  
 return pygame.font.Font("assets/font.ttf", size)  
  
 # вивід деякого тексту деякого розміру на деяку позицію  
 **def \_\_print\_text**(self, text, pos, size, color=GOLDEN):  
 text = Screen.\_\_get\_font(int(size)).render(text, True, color)  
 rect = text.get\_rect(center=pos)  
 self.SCREEN.blit(text, rect)  
  
 # екран автоматичного вирішення судоку  
 **def solve\_display**(self, mode, game\_matrix):  
 self.grid = Grid(self.SCREEN, game\_matrix, self.x, self.y + 40)  
 while self.running:  
 mouse\_pos = pygame.mouse.get\_pos()  
 SOLVE\_BACK = Button(pos=(640, 670), text\_input="BACK", font=Screen.\_\_get\_font(50), base\_color=BLUISH, hovering\_color=GOLDEN)  
 SOLVE\_BACK.changeColor(mouse\_pos)  
 self.SCREEN.fill(BEIGE)  
 SOLVE\_BACK.update(self.SCREEN)  
 self.grid.drawNumbers(self.SCREEN, game\_matrix[1])  
 self.grid.draw\_grid()  
 self.\_\_print\_text("Solved sudoku:", (640, 60), 50)  
 self.\_\_print\_text(mode, (440, 130), 10)  
 self.\_\_print\_text(str(self.hint\_amn) + "/3", (840, 130), 10)  
  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 sys.exit()  
 if event.type == pygame.MOUSEBUTTONDOWN:  
 if SOLVE\_BACK.checkForInput(mouse\_pos):  
 Screen.main\_menu(self)  
  
 pygame.display.update()  
  
 # екран перевірки правильності рішення(фінальний екран)  
 **def finish\_display**(self, mode, game\_matrix):  
 self.grid = Grid(self.SCREEN, game\_matrix, self.x + 300, self.y + 30)  
 while self.running:  
 self.SCREEN.fill(BEIGE)  
 mouse\_pos = pygame.mouse.get\_pos()  
 FINISH\_DISPLAY\_BACK = Button(pos=(350, 550), text\_input="QUIT", font=Screen.\_\_get\_font(50), base\_color=BLUISH, hovering\_color=GOLDEN)  
 FINISH\_DISPLAY\_BACK.changeColor(mouse\_pos)  
 FINISH\_DISPLAY\_BACK.update(self.SCREEN)  
  
 self.grid.drawNumbers(self.SCREEN, game\_matrix[1])  
 self.grid.draw\_grid()  
 if game\_matrix[0] == game\_matrix[1]:  
 self.\_\_print\_text("You've won!", (640, 60), 50)  
 self.\_\_print\_text("Congratulations!", (340, 250), 35)  
 self.\_\_print\_text("You used " + str(self.hint\_amn) + "/3 hints", (340, 400), 35)  
 self.\_\_print\_text(mode, (730, 120), 10)  
 else:  
 self.\_\_print\_text("You've lost(", (640, 60), 50)  
 self.\_\_print\_text(mode, (730, 120), 10)  
 self.\_\_print\_text(str(self.hint\_amn) + "/3", (1140, 120), 10)  
 self.\_\_print\_text("This was the right", (340, 250), 35)  
 self.\_\_print\_text("sudoku grid:", (340, 400), 35)  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 sys.exit()  
 if event.type == pygame.MOUSEBUTTONDOWN:  
 if FINISH\_DISPLAY\_BACK.checkForInput(mouse\_pos):  
 Screen.main\_menu(self)  
 pygame.display.update()  
  
 # екран гри на деякій складності  
 **def mode\_play**(self, mode, file\_insides):  
 solve\_pressed = 0  
 self.hint\_cells = []  
 if not file\_insides:  
 self.hint\_amn = 0  
 brd = Board()  
 game\_matrix = brd.createBoards(mode)  
 self.grid = Grid(self.SCREEN, game\_matrix, self.x, self.y)  
 with open("game\_started.txt", "w") as file:  
 file.write(self.grid.boardToCode(game\_matrix[0]) + '\n' + self.grid.boardToCode(game\_matrix[1]))  
 else:  
 game\_matrix = self.\_\_empty\_code\_list()  
 with open("game\_saved.txt", "r") as file:  
 lst = file.readlines()  
 game\_matrix[0] = self.\_\_codeToBoard(lst[0])  
 game\_matrix[1] = self.\_\_codeToBoard(lst[1])  
 cd = self.\_\_codeToBoard(lst[2])  
 self.hint\_amn = int(lst[4])  
 if len(lst) <= 5:  
 self.hint\_cells = []  
 else:  
 self.hint\_cells = self.\_\_code\_to\_hint\_pos(lst[5])  
 self.grid = Grid(self.SCREEN, game\_matrix, self.x, self.y, self.hint\_amn)  
 game\_matrix[0] = cd  
 self.notCompletedCells = []  
 while self.running:  
 play\_mouse\_pos = pygame.mouse.get\_pos()  
 PLAY\_BACK = Button(pos=(640, 670), text\_input="BACK", font=Screen.\_\_get\_font(50), base\_color=BLUISH, hovering\_color=GOLDEN)  
 PLAY\_BACK.changeColor(play\_mouse\_pos)  
 self.SCREEN.fill(BEIGE)  
 PLAY\_BACK.update(self.SCREEN)  
 self.grid.shadeCells(self.SCREEN)  
 self.grid.colorCells(self.SCREEN, '#FF6666', self.notCompletedCells)  
 self.grid.hint(self.SCREEN, self.hint\_cells, game\_matrix[0])  
 if self.selected:  
 self.grid.highlightCells(self.SCREEN, self.selected, SHADE)  
 self.grid.drawNumbers(self.SCREEN, game\_matrix[0])  
 pygame.draw.rect(self.SCREEN, BLUISH, pygame.Rect(90, 150, 240, 100))  
 SOLVE\_BUTTON = Button(pos=(210, 200), text\_input="SOLVE", font=Screen.\_\_get\_font(45), base\_color=BEIGE, hovering\_color="White")  
  
 pygame.draw.rect(self.SCREEN, BLUISH, pygame.Rect(90, 350, 240, 100))  
 HINT\_BUTTON = Button(pos=(210, 400), text\_input="HINT", font=Screen.\_\_get\_font(45), base\_color=BEIGE, hovering\_color="White")  
  
 pygame.draw.rect(self.SCREEN, BLUISH, pygame.Rect(930, 150, 240, 100))  
 CHECK\_BUTTON = Button(pos=(1050, 200), text\_input="CHECK", font=Screen.\_\_get\_font(45), base\_color=BEIGE, hovering\_color="White")  
  
 pygame.draw.rect(self.SCREEN, BLUISH, pygame.Rect(930, 350, 240, 100))  
 SAVE\_BUTTON = Button(pos=(1050, 400), text\_input="SAVE", font=Screen.\_\_get\_font(45), base\_color=BEIGE, hovering\_color="White")  
  
 for button in [SOLVE\_BUTTON, HINT\_BUTTON, CHECK\_BUTTON, SAVE\_BUTTON]:  
 button.changeColor(play\_mouse\_pos)  
 button.update(self.SCREEN)  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 sys.exit()  
 if event.type == pygame.MOUSEBUTTONDOWN:  
 if PLAY\_BACK.checkForInput(play\_mouse\_pos):  
 Screen.main\_menu(self)  
 if event.type == pygame.MOUSEBUTTONDOWN:  
 if SOLVE\_BUTTON.checkForInput(play\_mouse\_pos):  
 if file\_insides is True or solve\_pressed>0:  
 with open("game\_saved.txt", "w") as file:  
 file.truncate()  
 Screen.solve\_display(self, mode, game\_matrix)  
 if event.type == pygame.MOUSEBUTTONDOWN:  
 selected = self.grid.isOnTheGrid(play\_mouse\_pos)  
 if selected:  
 self.selected = selected  
 else:  
 self.selected = None  
 if event.type == pygame.MOUSEBUTTONDOWN:  
 if HINT\_BUTTON.checkForInput(play\_mouse\_pos) and self.grid.hint\_amount < 3:  
 temp\_hint = self.grid.solve(game\_matrix[0])  
 if temp\_hint is None:  
 if file\_insides is True or solve\_pressed > 0:  
 with open("game\_saved.txt", "w") as file:  
 file.truncate()  
 Screen.finish\_display(self, mode, game\_matrix)  
 else:  
 self.hint\_cells.append(temp\_hint)  
 self.hint\_amn = self.grid.hint\_amount  
 if event.type == pygame.MOUSEBUTTONDOWN:  
 if CHECK\_BUTTON.checkForInput(play\_mouse\_pos):  
 if len(self.grid.notComplettedCells(game\_matrix[0])) != 0:  
 self.notCompletedCells = self.grid.notComplettedCells(game\_matrix[0])  
 else:  
 if file\_insides is True or solve\_pressed > 0:  
 with open("game\_saved.txt", "w") as file:  
 file.truncate()  
 Screen.finish\_display(self, mode, game\_matrix)  
 if event.type == pygame.MOUSEBUTTONDOWN:  
 if SAVE\_BUTTON.checkForInput(play\_mouse\_pos):  
 solve\_pressed+=1  
 code, cd = self.grid.files  
 with open("game\_saved.txt", "w") as file:  
 file.write(code + '\n' + cd + '\n' + self.grid.boardToCode(game\_matrix[0]) + '\n' + mode + '\n' + str(self.hint\_amn) + '\n' + self.\_\_hint\_pos\_code())  
 if event.type == pygame.KEYDOWN:  
 if self.selected is not None and self.selected not in self.grid.lockedCells and self.selected not in self.hint\_cells:  
 if event.unicode.isdigit():  
 game\_matrix[0][self.selected[1]][self.selected[0]] = int(event.unicode)  
  
 self.grid.draw\_grid()  
 self.\_\_print\_text(mode, (440, 90), 10)  
 self.\_\_print\_text(str(self.hint\_amn) + "/3", (840, 90), 10)  
 pygame.display.update()  
  
 # перетворення стрічки значень позицій підказок на ліст позицій  
 **def \_\_code\_to\_hint\_pos**(self, code):  
 hint\_pos = []  
 for i in range(len(code)//2):  
 lst = [0, 0]  
 for j in range(2):  
 lst[j] = int(code[0])  
 code = code[1:]  
 hint\_pos.append(lst)  
 return hint\_pos  
  
 # функція перетворення позицій підказок на стрічку значень позицій підказок  
 **def \_\_hint\_pos\_code**(self):  
 code = ''  
 for i in range(0, len(self.hint\_cells)):  
 code += str(self.hint\_cells[i][0])  
 code += str(self.hint\_cells[i][1])  
 return code  
  
 # екран вибору складності гри  
 **def play**(self):  
 while self.running:  
 play\_mouse\_pos = pygame.mouse.get\_pos()  
 self.SCREEN.fill(BEIGE)  
 self.\_\_print\_text("SELECT YOUR MODE", (640, 60), 60)  
 PLAY\_BACK = Button(pos=(640, 670), text\_input="BACK", font=Screen.\_\_get\_font(50), base\_color=BLUISH, hovering\_color=GOLDEN)  
 PLAY\_BACK.changeColor(play\_mouse\_pos)  
 PLAY\_BACK.update(self.SCREEN)  
 PLAY\_EASY = Button(pos=(640, 200), text\_input="EASY MODE", font=Screen.\_\_get\_font(65), base\_color=BLUISH, hovering\_color=GOLDEN)  
 PLAY\_EASY.changeColor(play\_mouse\_pos)  
 PLAY\_EASY.update(self.SCREEN)  
 PLAY\_MEDIUM = Button(pos=(640, 350), text\_input="MEDIUM MODE", font=Screen.\_\_get\_font(65), base\_color=BLUISH, hovering\_color=GOLDEN)  
 PLAY\_MEDIUM.changeColor(play\_mouse\_pos)  
 PLAY\_MEDIUM.update(self.SCREEN)  
 PLAY\_HARD = Button(pos=(640, 500), text\_input="HARD MODE", font=Screen.\_\_get\_font(65), base\_color=BLUISH, hovering\_color=GOLDEN)  
 PLAY\_HARD.changeColor(play\_mouse\_pos)  
 PLAY\_HARD.update(self.SCREEN)  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 sys.exit()  
 if event.type == pygame.MOUSEBUTTONDOWN:  
 if PLAY\_BACK.checkForInput(play\_mouse\_pos):  
 Screen.main\_menu(self)  
 if PLAY\_EASY.checkForInput(play\_mouse\_pos):  
 mode = "easy"  
 Screen.mode\_play(self, mode, False)  
 if PLAY\_MEDIUM.checkForInput(play\_mouse\_pos):  
 mode = "medium"  
 Screen.mode\_play(self, mode, False)  
 if PLAY\_HARD.checkForInput(play\_mouse\_pos):  
 mode = "hard"  
 Screen.mode\_play(self, mode, False)  
 pygame.display.update()  
  
 # екран головного меню  
 **def main\_menu**(self):  
 while self.running:  
 self.SCREEN.fill(BEIGE)  
 menu\_mouse\_pos = pygame.mouse.get\_pos()  
 self.\_\_print\_text("SUDOKU", (640, 160), 150)  
 if os.stat("game\_saved.txt").st\_size == 0:  
 pygame.draw.rect(self.SCREEN, BLUISH, pygame.Rect(370, 310, 520, 170))  
 pygame.draw.rect(self.SCREEN, BLUISH, pygame.Rect(460, 500, 350, 100))  
 MAIN\_PLAY\_BUTTON = Button(pos=(640, 400), text\_input="PLAY", font=Screen.\_\_get\_font(120), base\_color=BEIGE, hovering\_color="White")  
 MAIN\_QUIT\_BUTTON = Button(pos=(640, 550), text\_input="QUIT", font=Screen.\_\_get\_font(75), base\_color=BEIGE, hovering\_color="White")  
  
 for button in [MAIN\_PLAY\_BUTTON, MAIN\_QUIT\_BUTTON]:  
 button.changeColor(menu\_mouse\_pos)  
 button.update(self.SCREEN)  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 sys.exit()  
 if event.type == pygame.MOUSEBUTTONDOWN:  
 if MAIN\_PLAY\_BUTTON.checkForInput(menu\_mouse\_pos):  
 Screen.play(self)  
 if MAIN\_QUIT\_BUTTON.checkForInput(menu\_mouse\_pos):  
 pygame.quit()  
 sys.exit()  
 else:  
 pygame.draw.rect(self.SCREEN, BLUISH, pygame.Rect(370, 260, 520, 170))  
 pygame.draw.rect(self.SCREEN, BLUISH, pygame.Rect(460, 600, 350, 100))  
 pygame.draw.rect(self.SCREEN, BLUISH, pygame.Rect(380, 460, 500, 110))  
 MAIN\_PLAY\_BUTTON = Button(pos=(640, 350), text\_input="PLAY", font=Screen.\_\_get\_font(120), base\_color=BEIGE, hovering\_color="White")  
 MAIN\_RESUME\_BUTTON = Button(pos=(640, 520), text\_input="RESUME", font=Screen.\_\_get\_font(80), base\_color=BEIGE, hovering\_color="White")  
 MAIN\_QUIT\_BUTTON = Button(pos=(640, 650), text\_input="QUIT", font=Screen.\_\_get\_font(75), base\_color=BEIGE, hovering\_color="White")  
 for button in [MAIN\_PLAY\_BUTTON, MAIN\_RESUME\_BUTTON, MAIN\_QUIT\_BUTTON]:  
 button.changeColor(menu\_mouse\_pos)  
 button.update(self.SCREEN)  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 sys.exit()  
 if event.type == pygame.MOUSEBUTTONDOWN:  
 if MAIN\_RESUME\_BUTTON.checkForInput(menu\_mouse\_pos):  
 with open('game\_saved.txt', "r") as file:  
 mode = str(file.readlines()[3].strip('\n'))  
 Screen.mode\_play(self, mode, True)  
 if event.type == pygame.MOUSEBUTTONDOWN:  
 if MAIN\_PLAY\_BUTTON.checkForInput(menu\_mouse\_pos):  
 Screen.play(self)  
 if MAIN\_QUIT\_BUTTON.checkForInput(menu\_mouse\_pos):  
 pygame.quit()  
 sys.exit()  
 pygame.display.update()  
  
 # перетворення стрічки значень елементів поля гри на матрицю поля гри  
 **def \_\_codeToBoard**(self, code):  
 board = [[0 for \_ in range(9)] for \_ in range(9)]  
 for row in range(9):  
 for col in range(9):  
 board[row][col] = int(code[0])  
 code = code[1:]  
 return board

# функція створення ліста з двох порожніх матриць розміром 9х9  
 **def \_\_empty\_code\_list**(self):  
 board = [[0 for \_ in range(9)] for \_ in range(9)]  
 game\_matrix = []  
 for row in range(2):  
 game\_matrix.append(board)  
 return game\_matrix

**main.py**

from screen import \*  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 pygame.init()  
 SCREEN = pygame.display.set\_mode((1280, 720))  
 pygame.display.set\_caption("Sudoku")  
 menu = Screen(SCREEN)  
 menu.main\_menu()