**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**

**імені ІГОРЯ СІКОРСЬКОГО»**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ФАКУЛЬТЕТ БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ

(повна назва інституту/факультету)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_кафедра БІОМЕДИЧНОЇ КІБЕРНЕТИКИ

(повна назва кафедри)

| **ДОМАШНЯ КОНТРОЛЬНА РОБОТА** | |
| --- | --- |
|  | |
| з дисципліни (кредитного модуля) | **Основи програмування** |

| спеціальність | ***122 Комп’ютерні науки*** |
| --- | --- |
| спеціалізація | ***Комп’ютерні технології в біології та медицині*** |

| **На тему** |  |
| --- | --- |
|  | |
| (тема індивідуального завдання) | |
| **Варіант №** | *Якщо заплановано в додатку до РНП* |

**Виконав студент \_1\_\_-го курсу гр. БС-03**

**ЗАТУЛОВСЬКИЙ ГЕОРГІЙ АНДРІЙОВИЧ**

***Засвідчую, що у роботі немає запозичень з праць інших авторів без відповідних посилань.***

**Студент (-ка) \_\_\_\_\_\_\_\_\_\_\_\_\_**

**Перевірив ст. викл. Олена ВДОВИЧЕНКО**

**ст. викл. Галина КОРНІЄНКР**

**Бали за роботу студента відповідно до РСО за**

| Функціональність, Алгоритм, Розрахунок (6- 9 балів) |  |
| --- | --- |
| Оформлення (2-3 балів) |  |
| Захист (2-3 балів) |  |
| ВСЬОГО (6-15балів) |  |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(підпис викладачів)

**Київ – 202\_\_р.**

**Національний технічний університет України**

**«Київський політехнічний інститут імені Ігоря Сікорського»**

| Інститут (факультет) | | БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ |
| --- | --- | --- |
| (повна назва) | | |
| Кафедра | БІОМЕДИЧНОЇ КІБЕРНЕТИКИ | |
| (повна назва) | | |

**ЗАВДАННЯ**

**на домашню контрольну роботу студенту**

| **ЗАТУЛОВСЬКИЙ ГЕОРГІЙ АНДРІЙОВИЧ** | |
| --- | --- |
| (прізвище, ім’я, по батькові) | |
| 1. Тема роботи (варіант) | *гра “Сапер”* |
|  | |
|  | |

| 2. Термін подання студентом роботи | | | ***14-21 грудня 2020 року*** |
| --- | --- | --- | --- |
| 3. Вихідні дані до роботи | | *гра “Сапер”* | |
| *Cередовище IDLE* | | | |
|  | | | |
| 4. Зміст роботи | *Практика роботи в середовищі IDLE для гри* | | |
| *гра повинна мати 3 рівні складності;:”Новачок” (поле 9х9, кількість мін – 10),* | | | |
| *”Любитель”(поле 16х16, кількість мін - 40),”Професіонал”( поле 16х30,* | | | |
| *кількість мін - 90) Забезпечити можливість зберігати гру і продовжити* | | | |
| *збережену гру та вивід списку з 10 найкращих результатів* | | | |

5. Дата видачі завдання  ***19-23 жовтня 2020 р.***

**Календарний план**

| № з/п | Назва етапів виконання  курсової роботи | Термін виконання  етапів роботи | Примітка |
| --- | --- | --- | --- |
| 1 | Отримати завдання на КР | 19-23 жовтня 2020р.. |  |
|  | Оформлення розділу з | До 5 грудня 2020р. |  |
|  | Оформлення розділу з | До 7 грудня 2020р. |  |
|  | Оформлення розділу з | До 9 грудня 2020р. |  |
|  | Оформлення ДКР | До 12 грудня 2020р. |  |
|  | Подання в електронному вигляді ДКР на перевірку | До 14 грудня 2020р. |  |
|  | Подання пакету документів по ДКР до захисту | До 15 грудня 2020р. |  |
|  | Захист ДКР | 28-30 грудня 2020р. |  |

| Студент |  |  |  | Г. А. ЗАТУЛОВСЬКИЙ |
| --- | --- | --- | --- | --- |
|  |  | (підпис) |  | (ініціали, прізвище) |
| Керівник роботи |  |  |  | Олена ВДОВИЧЕНКО |
|  |  | (підпис) |  | (ініціали, прізвище) |

**ЗМІСТ**

[ВСТУП 4](#_1fob9te)

[РОЗДІЛ 1 ТЕОРЕТИЧНА ЧАСТИНА](#_3znysh7)

[1.1 Основна література 5](#_2et92p0)

[Висновки до розділу 1 12](#_3dy6vkm)

[РОЗДІЛ 2 АНАЛІТИЧНА ЧАСТИНА](#_1t3h5sf)

[2.1 Актуальність 13](#_4d34og8)

[2.2 Підходи вирішення завдання 16](#_2s8eyo1)

[Висновок до розділу 2 19](#_17dp8vu)

[РОЗДІЛ 3 ПРАКТИЧНА ЧАСТИНА](#_3rdcrjn)

[3.1 Лістинг програми 20](#_26in1rg)

[3.2 Результати 30](#_lnxbz9)

[Висновок до розділу 3 35](#_35nkun2)

[ЗАГАЛЬНІ ВИСНОВКИ 36](#_1ksv4uv)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 37](#_44sinio)

**ВСТУП**

Метою завдання є реалізування гри «Сапер».

**Правила гри:**

Прямокутне ігрове поле поділене на суміжні комірки (квадрати), деякі з яких «заміновані»; кількість «замінованих» комірок відомо. Гравець відкриває комірки, намагаючись не відкрити комірку з міною. Відкривши комірку з міною, він програє. Метою гри є відкриття всіх комірок, які не містять міни.

**Вимоги до програми:**

Міни повинні розставлятися після першого ходу. Якщо під відкритою коміркою міни немає, то в ній повинно з’являтися число, що показує, скільки комірок, що є сусідами щойно відкритої, «заміновано». Якщо під сусідніми комірками теж немає мін, то повинна відкритися деяка «не замінована» область до комірок, в яких є цифри.

Гравець повинен мати можливість позначати «заміновані» комірки (щоб випадково не відкрити їх) двома кольорами або типами значків: «ймовірно є міна», і «міна є точно» (в класичному «Сапери» це знак питання і прапорець, відповідно). Гравцю необхідно показувати загальну кількість мін і кількість знайдених мін.

Гра повинна мати 3 рівні складності: «Новачок» (поле 9x9, кількість мін - 10), «Любитель» (поле 16x16, кількість мін - 40) і «Професіонал» (поле 16x30, кількість

мін - 90).

Забезпечити можливість зберігати гру і продовжити збережену гру. Для того, щоб гравець не міг дізнатися правильне розташування мін, бажано зберігати ігрове поле в двійковому форматі в бінарному файлі.

Реалізувати збереження результатів. Для цього необхідно реалізувати аутентифікацію гравця (запит/перевірку імені користувача і пароля перед початком гри). Для кожного гравця зберігати до 10 результатів ігор. Забезпечити можливість перегляду результатів окремого гравця і загального рейтингу гравців.

За даними вітчизняної та зарубіжної науково-технічної літератури, можно зробити висновок, що найкраще використати для виконання поставленої задачі метод програмування “Зверху вниз”.

Програмування «зверху вниз», або низхідне програмування – це методика розробки програм, при якій розробка починається з визначення цілей вирішення проблеми, після чого йде послідовна деталізація, що закінчується детальною програмою.[[5]](#_3j2qqm3)

Для виконання цього завдання треба вибрати мову програмування і системи (середовища) програмування, визначитися з методами вирішення задачі, використовуючи різноманітні способи реалізування програми, визначити всі недоліки та переваги рішень, створити план послідовності виконання поставлених задач й поступово зв’язувати їх між собою у повноцінну програму.

**РОЗДІЛ 1 ТЕОРЕТИЧНА ЧАСТИНА**

* 1. **Основна література**

Розроблено безліч ігор, які допоможуть поглибити свої пізнання в тій чи іншій області, навчать діяти в різних ситуаціях, сприяють формуванню посидючості, зосередженості, уважності, тренують здатність до логічного мислення, просторової уяви, планування і вмінню вчитися на помилках.

Предметом дослідження є програмна реалізація логічної гри «Сапер». «Сапер» - нескладна гра, яка, тим не менш, розвиває увагу, пам'ять і логіку. Основна її мета - знайти і позначити прапорцями міни. Потрібно бути дуже обережним: як тільки ви відкриваєте на клітку, під якою захований вибухонебезпечний сюрприз, гра закінчена. Сапер помиляється тільки один раз. [[2]](#_2jxsxqh)

Для досягнення мети дослідження поставимо перед собою наступні завдання:

* Здійснити вибір мови та середовища програмування.
* Розробити програмну реалізацію логічної гри " Сапер».
* Скласти пояснювальну записку для опису функціональних можливостей розробленого програмного додатка.

**1.1.1. Вибір середовища програмування**

Розглядаючи переваги та недоліки існуючих мов програмування, можна зробити висновок, що найкращим вибором для виконання цієї задачи є мова Python.

**Переваги Python:**

* Наявність великої кількості модулів, що підключаються до програми, які забезпечують різноманітні додаткові можливості (*Tkinter, random, math, sys* та інші).
* Завдяки простоті та гнучкості мови Python можна знайти багато підходів для виконання поставленого завдання.
* Читабельність. Читається майже як нативна англійська мова. Коли я вперше побачив код на мові Python, велика частина його була зрозуміла просто з імен ключових слів та структури програми. Так, відступи там обов’язкові (це вже може трактуватися як плюсом так і мінусом). Простота та прямота коду, легко зрозуміти що звідки береться.
* Інтерпретована та з динамічною типізацією. Тобто менше коду для визначення змінних. Це дозволяє студентові зосередитись на логіці програми, розв’язуванні задачі, а не так на нюансах мови та нутрощах комп’ютера (пам’яті, процесів, і т.д).

**Недоліки Python:**

* Порівняно невисока швидкість виконання Python-програми, що обумовлено її інтерпретованістю. Проте переваги значно більші у програмах не дуже критичних щодо швидкості виконання.

Існує два способи виконання інтерпретатора: командний режим і режим виконання програм із файлів.

У стандартний комплект Пітона входить інтегроване середовище IDLE, в якому редагувати програми буде набагато легше, ніж в звичайному текстовому редакторі. IDLE має вбудовану систему відлагодження, що дозволяє запускати програму порядково, що полегшує процес пошуку помилок.[[1]](#_1y810tw)

**1.1.2. Способи реалізації завдань**

**1.1.2.1 Генерування випадкових чисел.**

Міни повинні розставлятися після першого ходу. Для цієї задачи треба імпортувати модуль random, який надає можливість отримувати випадкові числа.

В сучасних умовах для отримання випадкових чисел використовують різноманітні генератори, які поділяються на апаратні та програмні.

В апаратних генераторах джерелом випадкових чисел є шум в електронних приладах. Проте, використання апаратних генераторів вимагає наявності спеціального обладнання. У зв’язку з цим більш зручним вважається застосування програмних генераторів випадкових чисел.

Програмний генератор випадкових чисел являє собою програму, яка генерує послідовність чисел за деяким алгоритмом. Завдяки алгоритму така послідовність чисел цілком детермінована, тобто не може бути цілком випадковою. Її називають послідовністю псевдовипадкових чисел.

В Python є вбудований модуль, який дозволяє генерувати псевдовипадкові числа.

Підключити модуль можна за допомогою інструкції import.

1. Функція **random()**

Щоб отримати випадкове дійсне число, або число з плаваючою точкою, слід

використовувати функцію random() з однойменного модуля random мови Python.

Вона не приймає ніяких аргументів і повертає число від 0 до 1, не включаючи 1, або

на мові математики [0, 1).

2.) Функція **randint()**

Функція randint(a,b) генерує псевдовипадкові цілі числа. Вона завжди приймає

тільки два аргументи - межі целочисленного діапазону, з якого вибирається

будь-яке число. Обидві межі включаються в діапазон, тобто на мові

математики відрізок описується як [a; b].

3.) Функція **randrabge()**

Якщо в randrange() передається три аргументи, то перші два - це межі діапазону,

як у випадку з двома аргументами, а третій - так званий крок.

4.) Функція **random.choice(x)**

Повертає випадковий елемент з непорожньої послідовності

(списку або рядка) x.

5.) Функція **random.shuffle(x)**

Випадковим чином “перемішує” елементи послідовності

(списку або рядка) x, зберігаючи результат в x.

**1.1.2.2 Одновимірний та двовимірний масив**

Для створення ігрового поля, яке поділене на суміжні комірки (квадрати) слід вибрати тип масиву котрий ми будемо використовувати для вирішення нашої задачі.

**Одновимірний масив (список)**

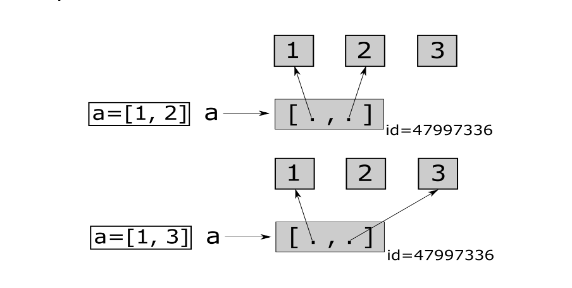
Список – спосіб доступу до даних, коли є одне ім’я і багато даних. При його створенні в пам’яті резервується область,яку умовно можна назвати деяким контейнером, в якому зберігаються посилання на інші елементи даних в пам’яті.На відміну від таких типів даних, як число чи рядок, вміст контейнера можна змінювати (рис. 1.1).

Рисунок 1.1 Знаходження елементів списку в пам’яті

Методи списків:

* append(item): додає елемент item до кінця списку
* insert(index, item): додає елемент item до списку за індексом index
* remove(item): видаляє елемент item. Видаляється тільки перше входження

елемента. Якщо елемент не знайдено, генерує виключення ValueError

* clear(): видалення всіх елементів зі списку
* index(item): повертає індекс елемента item. Якщо елемент не знайдено, генерує

виключення ValueError

* pop([index]): видаляє та повертає елемент за індексом index. Якщо індекс не

переданий, то видаляє останній елемент

* count(item): повертає кількість входжень елементів item до списку
* sort([key]): сортує елементи. За замовчуванням сортує по зростанню. Але за

допомогою параметра key ми можемо передати функцію сортування

* reverse(): переставляє всі елементи в зворотному порядку

Функції списків:

* len(list): повертає довжину списку
* sorted(list, [key]): повертає відсортований список
* reversed(list): повертає обернений список
* min(list): повертає найменший елемент списку
* max(list): повертає найбільший елемент списку

**Двовимірний масив**

Двовимірний масив — це таблиця з рядків і стовпчиків. У таблицях перший номер вказує на рядок, а другий — на положення елемента в рядку. Усі рядки таблиці мають однакову довжину.У мові програмування Python двовимірний масив можна представити у вигляді списку рядків, кожен елемент якого є в свою чергу списком.

У мові програмування Python двовимірний масив можна представити у

вигляді списку рядків, кожен елемент якого є в свою чергу списком.

Приклад. Створення числової таблиці з двох рядків і трьох стовпців, над

якими здійснюються різні дії (рис. 1.2)

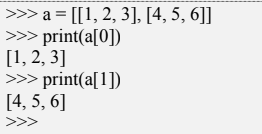


Рисунок 1.2 Виведення елементів двовимірного масиву

Перший рядок списку a[0] є списком з чисел [1, 2, 3]. Тобто a[0][0]=1,значення a[0][1]=2, a[0][2]=3, a[1][0]=4, a[1][1]=5, a[1][2]=6.Для обробки і виведення списку, як правило, використовують двавкладених цикли. Перший цикл перебирає номер рядка, другий цикл проходить за елементами всередині рядка.

**1.1.2.3 Файли**

Для реалізування функції збереження та загрузки гри треба пов’язати програму з файлами, що знаходяться у довільному доступі.

Активна програма працює з даними, які зберігаються в запам’ятовуючому пристрої з довільним доступом (Random Access Memory (RAM)). RAM – дуже швидка пам’ять, але вона дорога і вимагає постійного живлення; якщо живлення пропаде, то всі дані, які в ній зберігаються, будуть втрачені. Жорсткі диски повільніші оперативної пам’яті, але вони більш місткі, коштують дешевше і можуть зберігати дані навіть після того, як хтось вимкне живлення. Тому багато зусиль при створенні комп’ютерних систем направлено на пошук кращого співвідношення між зберіганням даних на диску і в оперативній пам’яті.

Найпростіший приклад стійкого сховища – це файл, що являє собою послідовність байтів, яка зберігається під ім’ям файлу. Можна зчитувати дані з файлу в пам’ять і записувати дані з пам’яті в файл. Python дозволяє робити це досить легко.

Перед тим як щось записати в файл або зчитати з нього, необхідно відкрити його. Щоб відкрити файл, програма повинна викликати функцію open(), передавши їй ім’я зовнішнього файлу і режим роботи:

**fileobj = open (filename, mode)**

де

* fileobj– це об’єкт файлу, що повертається функцією open ();
* filename - це рядок, що представляє собою ім’я файлу;
* mode - це рядок, що вказує на тип файлу і дії, які необхідно над ним

зробити.

Перша літера рядка mode вказує на операцію:

* r - означає читання;
* w - означає запис. Якщо файлу не існує, він буде створений. Якщо файл

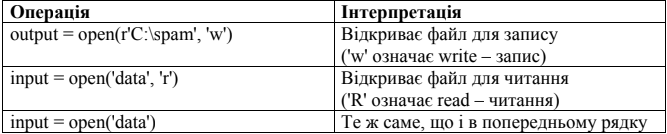
існує, він буде перезаписаний;

* x - означає запис, але тільки якщо файлу ще не існує;
* a - означає додавання даних в кінець файлу, якщо він існує.

Друга літера рядка mode вказує на тип файлу:

* t - (або нічого) означає, що файл текстовий;
* b - означає, що файл бінарний.

Крім того, функція може приймати третій необов’язковий аргумент, керуючий буферизацією виведених даних, – значення нуль означає, що вихід на інформація не буде буферизованою (тобто вона буде записуватися у зовнішній файл відразу ж, в момент виклику методу запису). Ім’я зовнішнього файлу може включати платформо залежні префікси абсолютного або відносного шляху до файлу. Якщо шлях до файлу не вказано, передбачається, що він знаходиться в поточному робочому каталозі (тобто в каталозі, де був запущений сценарій).Після відкриття файлу можна викликати функції для читання або запису даних. По завершенню роботи, файл потрібно закрити.Існує кілька різновидів методів читання і запису, а в табл. 1.2.1 перераховані лише найбільш часто використовувані з них:

Таблиця 1.2.1

**1.1.2.4 Функції**

Завдяки функціям ми зможемо реалізувати програму методом “Зверху вниз” та створити послідовний цикл виконання програми.

Функція в Python є основою при написанні програм. До сих пір ми використовували вбудовані функції і функції з модулів, які входять в комплект його постачання. Але міць структурних мов програмування полягає в тому, що ми можемо створювати свої власні функції, при чому робиться це досить просто. У структурному програмуванні функція являє собою іменовану послідовність виразів, що виконують необхідну операцію.

Крім складання складних математичних виразів Python дозволяє передавати результати виконання функцій в якості аргументів інших функцій без використання додаткових змінних (рис 1.3)

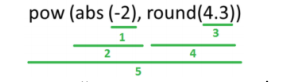


Рисунок 1.3 Приклад виклику та обчислення виразів

На малюнку представлений приклад виклику і порядок обчислення

виразів. У цьому прикладі на місці числових об’єктів (-2, 4.3) можуть перебувати виклики функцій або їх комбінації, тому вони теж обчислюються.

Функція int () повертає цілочисельний об’єкт, побудований з числа або

рядка, або 0, якщо аргументи не передані.

Функція float () повертає число з плаваючою точкою, побудоване з числа

або рядка.

Функція в Python – об’єкт, який приймає аргументи і повертає значення. Оголошення функції описує її прототип (іноді кажуть "сигнатура"). Зазвичай функція визначається за допомогою інструкції def, що має такий синтаксис:

**def ІМ'Я\_ФУНКЦІЇ(СПИСОК\_ПАРАМЕТРІВ):**

**ПОСЛІДОВНІСТЬ\_ВИРАЗІВ**

Правила вибору імен функцій повністю аналогічні правилам вибору імен змінних. Список параметрів визначає набір значень, які можуть бути передані функції в якості вихідних даних – параметри перераховуються через кому. Перший рядок визначення зазвичай називається заголовком функції, позначених ключовим словом def (від англ. «Define» – «визначити»). Заголовок функції в Python завершується двокрапкою. Після нього може слідувати будь-яка кількість виразів, але вони повинні бути записані зі зміщенням щодо початку рядка.

**Параметри та аргументи функції**

Часто функція використовується для обробки даних, отриманих із зовнішнього для неї середовища (з основної програми). Дані передаються функції при її виклику в дужках і називаються аргументами. Однак, щоб функція могла "взяти" передані їй дані, необхідно при її створенні описати параметри (в дужках після імені функції), що представляють собою змінні.

Параметрами функції називають список змінних, яким присвоюються при виклику цієї функції значення, що передаються. А самі передані значення називають аргументами. У своєму тілі функція оперує параметрами, але не аргументами.

Коли функція викликається, конкретні аргументи підставляються замість параметрів-змінних. У більшості випадків кількість аргументів і параметрів має збігатися (хоча можна запрограмувати змінну кількість прийнятих аргументів). В якості аргументів можуть виступати як безпосередньо значення, так і змінні, що посилаються на них (рис 1.4).[[3]](#_z337ya)

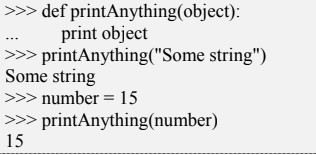


Рисунок 1.4 Приклад роботи функції, яка приймає аргумент

**Висновки до розділу 1**

Ми висвітлити стан питання, якому присвячена робота, навели стисле роз’яснення, поставленої задачі в ДКР та дали об’єктивну оцінку стану поставленої задачі, навести існуючі переваги і недоліки у вирішенні задачі, проаналізувати можливі підходи до її вирішення, узагальнити результати розв’язку задачі в порівнянні цього та суміжних питань, визначити можливі шляхи вирішення поставленого завдання та викласти загальну методику і основні методи вирішення поставленої задачі, що будуть використані в другому та третьому розділах.

**РОЗДІЛ 2 АНАЛІТИЧНА ЧАСТИНА**

**2.1 Актуальність**

Гра "Сапер" - це проста, але в той же час дуже цікава гра-головоломка. На початку гри відкривається поле, розділене на рівні клітини, на якому заховані міни. Коли гравець клацає мишкою по довільній клітці на полі, то в цій клітці з'являється цифра, що показує скільки хв заховано по сусідству. Аналізуючи ці цифри можна зрозуміти, де заховані наступні міни. Потім на клітці (в якій гравець передбачає, що захована міна) ставиться прапорець для наочного позначення міни. Таким чином, завдання гравця-знайти всі міни, заховані на ігровому полі, і при цьому постаратися не підірватися на них. [5]

**Особливості гри “Сапера”**

* Розвиває логічне мислення
* Розвиває швидкість мислення
* Вчить швидко приймати рішення

**Локальні міркування: нуль сусідніх мін**

В оригінальній грі використовується один автоматичний механізм: коли гравець відкриває клітку, поруч з якою немає мін, движок гри відкриває всі сусідні клітини. Це нічим не загрожує грі, тому можна спокійно дозволити комп'ютеру це зробити, а сама ситуація відразу ж зрозуміла гравцеві і ніяк не заважає геймплею.

Таке міркування абсолютно локально: для прийняття рішення про наступну дію враховується інформація тільки однієї клітини.

Складно придумати ситуацію, в якій гра стала б гірше без цієї автоматичної допомоги. Спробуйте зіграти в таку гру, щоб отримати уявлення про те, як вона проходить без автоматичного відкриття клітин [в оригіналі статті всі приклади інтерактивні]:

**Локальные рассуждения с учетом окружения**

Новому гравцеві нескладно буде зрозуміти, що якщо кількість сусідніх мін, тобто число, показане в клітці, дорівнює кількості невідкритих сусідніх клітин, то всі ці клітини повинні бути мінами, тому на них треба поставити прапорці. Аналогічно, коли кількість сусідніх мін дорівнює кількості сусідніх прапорів, то залишилися невідкритими сусідні клітини повинні бути порожні.

У цих правилах враховується одна клітина, а також стан сусідніх (відкриті/поставлений прапорець).

Реалізація цих правил вручну може бути захоплюючою. Якщо додати таймер, то гравець починає вчитися застосовувати їх швидко і точно. Це перетворює "Сапера" в гру на реакцію.

У подібної автоматизації є цікавий побічний ефект-установка прапорця може миттєво мати фатальні наслідки.

В іншому у нас можуть виникнути ситуації, які можна розбити на три категорії:

Ігри, повністю дозволяються застосуванням автоматичних правил Складні ситуації, що вимагають для міркувань більшої кількості клітин cтану гри, в яких немає логічного шляху вперед — гравцеві залишається тільки вибирати випадково, можливо з урахуванням ймовірностей.

Ситуація 1 здається красивою, але не особливо цікавою, якщо буде виникати занадто часто. Чи будуть такі ігри краще без автоматичного рішення? Може бути і ні; такі ігри дуже прості навіть при вирішенні вручну, і гравцеві не особливо цікаво грати в ігри, в яких немає труднощів. Хоча, зрозуміло, в грі на реакцію складність є завжди: потрібно діяти якомога швидше.

Дуже привабливою мені здається Ситуація 2. Ми більше зосереджуємося на вирішенні логічних умов, менше витрачаючи час на точне прицілювання і натискання правильних кнопок. Це робить "Сапера" більше схожим на активну головоломку.

Ситуація 3 повністю руйнує всю захопливість. Втім, я чув, що деяким людям подобається грати в ігри з випадковістю.

**Повне рішення: глобальні міркування**

Для алгоритмічного виявлення всіх необхідних для стану гри логічних умов нам потрібно виконати вичерпний пошук всіх станів гри. Доведено, що Minesweeper-це NP-повне завдання. Нижче вказаний невеликий, але цікавий і наочний приклад стану гри, що має всього одне логічне рішення, але для його знаходження необхідно враховувати стан гри цілком:

Чи можливо виконати пошук по всьому простору станів гри? Скільки всього існує варіантів станів s?

Даний:

w = ширина поля

h = висота поля

k = кількість хв

n = w · h

Тоді кількість можливих станів s дорівнює (рис 2.1).

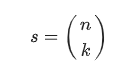


Рисунок 2.1 кількість можливих станів

Для стандартних рівнів "новачок", "любитель" і» професіонал " це дає нам таку кількість можливих станів (рис 2.2).

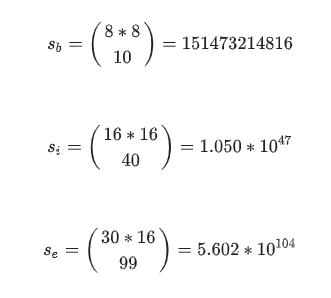


Рисунок 2.2 кількість можливих станів при різних

рівнів складності

Ми розуміємо, що наївний підхід тут абсолютно не підходить. Давайте подивимося, як би міг виглядати наївний алгоритм, і дізнаємося, чи можна його оптимізувати в щось працююче.

**Наївний алгоритм**

Завдання алгоритму-знайти всі необхідні для заданого стану поля логічні умови. Було б складно реалізувати це ретельним продумуванням рішень; комп'ютер набагато краще справляється з швидким виконанням купи дурних дій.

Що ми можемо зробити «дурного»: згенерувати всі можливі перестановки позицій Мін для всіх, що залишилися мін, якщо така перестановка відповідає всім відкритим числам, то вона буде правильним рішенням гри. Потім ми вивчаємо всі можливі перестановки, знаходимо всі можливі рішення, але як і раніше не знаємо, яке з них є вірним.

Якщо у всіх можливих рішеннях є щось спільне, або серед відкритих клітин, або серед клітин, позначених як міни, то ми розуміємо, що це загальне має бути частиною вірного рішення для поточного поля. І справді: неможливо створити вірного рішення, що не має таких співпадаючих елементів, інакше б ми їх виявили.

Таким чином ми можемо знайти всі необхідні для поточного стану поля логічні умови.

**Клітини з обмеженнями і без обмежень**

Наведений вище алгоритм має очевидну проблему: кількість станів, які потрібно досліджувати. Але не всі клітини однакові. Невідкриті клітини, що знаходяться поруч з числом, очевидно обмежені цим числом. Ми назвемо ці клітини обмеженими. Решта клітини ми назвемо необмеженими.

Якщо ми реалізуємо наведений вище алгоритм, але будемо виконувати пошук тільки в просторі станів обмежених клітин, і будемо повертатися назад, як тільки порушимо обмеження, то в багатьох іграх зможемо вирішити всі логічні умови за розумний проміжок часу:

У разі необмежених клітин ми ніяк не можемо дізнатися, де розташовані міни, і логічно відразу знаємо про це. Це означає, що можна виключити їх з обчислення і розглядати тільки розташування мін поруч з відкритими числами.

Однак ми знаємо, що деяка кількість хв може потрапити в безліч необмежених клітин; якщо є 6 хв і 4 обмежених клітини, то в обмежених клітинах може бути максимум 4 міни, тобто не менше 2 хв повинно знаходитися в необмежених клітинах. За аналогічною логікою ми іноді можемо визначити, що всі необмежені клітини повинні бути порожніми або всі містити міни.

У наведеному нижче випадку ми знаємо позиції всіх мін, тому AI повинен бути здатний зрозуміти, що залишилися осередки не зайняті.

**2.2 Підходи вирішення завдання**

**Версія з випадковістю**

Якщо ми автоматично будемо запускати глобальний солвер, то отримаємо оптимізовану по випадковості версію " Сапера»:

Можна розділити гри в цій версії на три категорії:

* Ігри, в яких гравець робить довільний вибір і виграє.
* Ігри, в яких гравець робить довільний вибір і програє.
* Ігри, в яких робота AI вимагає багато часу, і гравець насправді може використовувати міркування.

Очевидно, що це гра з випадковостями. У чому ж привабливість таких ігор? З точки зору логіки показана вище гра схожа з такою:

Але яка з ігор з випадковостями краще? Схоже, що сенс інших ігор з випадковостями полягає в існуванні складної зв'язку між діями гравця і перемогою / програшем. Для витягування номерів лотереї використовуються складні машини, які спеціально не поспішають у виборі номера і роблять з показу цього номера ціле шоу.

Можливо, велике поле, яке вирішується автоматично, є досить хорошою грою

з випадковостями, враховуючи те, що гравець спостерігає за поступовим відкриванням всіх клітин.

У наступному випадку ми не знаємо позицій всіх мін, але можемо зрозуміти, що залишилася міну потрібно помістити в одну з двох клітин зліва вгорі. Це означає, що залишилася в правому нижньому кутку клітина вільна.

**Детермінована версія**

Тепер у нас є штучний інтелект, здатний визначати всі логічні кроки з заданого стану гри. Іноді він не зможе знайти логічних кроків. У таких ситуаціях гравцеві доводиться вгадувати і він може програти, якщо йому не пощастить.

Що якщо ми додамо ще одне правило? Коли у гри немає логічного шляху вперед, то ми можемо попросити про допомогу. Якщо штучний інтелект погоджується, що гравець не може нічого вдіяти, то приходить йому на допомогу. В іншому випадку гравець негайно програє. Це може бути цікавим. Якою може бути така допомога? Можливо, потрібно відкрити одну клітку, незалежно від наявності в ній міни.

Таким чином, ми повністю позбулися ситуацій, в яких можна було програти випадково.

Однак тут є один виняток: як і раніше існує ймовірність вироджених ситуацій, в яких глобальний солвер не може закінчити обчислення за розумний проміжок часу. На жаль, це неминучий результат того, що завдання» Сапера " NP-повна.

**Версія за типом інтерфейсу**

Ці верcії відрізняються за типом відображення інтерфейсу:

консольний або віконний додаток. Зазвичай программа автоматично виводить програму у консольному варіанті, а для створення віконного додатку треба імпортувати окремі бібліотеки (наприклад модуль Tkinter).

Консольні додатки головним чином використовуються для виконання вторинних або фонових завдань, таких як генерація коду, компіляція пошукового індексу, відправлення повідомлень електронної пошти і т.д. Yii надає інструмент для розробки консольних додатків, дотримуючись об'єктно-орієнтованого підходу. Він дозволяє консольному додатку отримати доступ до ресурсів, які використовує основний веб-додаток (наприклад, до бази даних).

Tkinter-це пакет для Python, призначений для роботи з бібліотекою Tk. Бібліотека Tk містить компоненти графічного інтерфейсу користувача (graphical user interface-GUI), написані на мові програмування Tcl.

Під графічним інтерфейсом користувача (GUI) маються на увазі всі ті вікна, кнопки, текстові поля для введення, скролери, списки, радіокнопки, прапорці та ін., які ви бачите на екрані, відкриваючи ту чи іншу програму. Через них ви взаємодієте з програмою і керуєте нею. Всі ці елементи інтерфейсу будемо називати віджетами (widgets).

В даний час майже всі програми, які створюються для кінцевого користувача, мають GUI. Рідкісні програми, що мають на увазі взаємодію з людиною, залишаються консольними. У попередніх двох курсах ми писали тільки консольні програми.

Існує безліч бібліотек GUI, серед яких Tk не найпопулярніший інструмент, хоча з його допомогою написано чимало проектів. Він був обраний для Python за замовчуванням. Інсталяційний файл інтерпретатора пітона зазвичай вже включає пакет tkinter у складі стандартної бібліотеки.

Tkinter можна охарактеризувати як перекладач з мови Python на мову Tcl. Ви пишете програму на Python, а код модуля tkinter переводить ваші інструкції на мову Tcl, який розуміє бібліотека Tk.

Додатки з графічним інтерфейсом користувача подієво-орієнтовані. Ви вже повинні мати уявлення про структурне і бажано об'єктно-орієнтоване програмування. Подієво-орієнтоване орієнтоване на події. Тобто та чи інша частина програмного коду починає виконуватися лише тоді, коли трапляється та чи інша подія.

Подієво-орієнтоване програмування базується на об'єктно-орієнтованому та структурному. Навіть якщо ми не створюємо власних класів і об'єктів, то все-одно ними користуємося. Всі віджети-об'єкти, породжені вбудованими класами.

Події бувають різними. Спрацював часовий фактор, хтось клікнув мишкою або натиснув Enter, почав вводити текст, переключив радіокнопки, прокрутив сторінку вниз і т.д. коли трапляється щось подібне, то, якщо був створений відповідний обробник, відбувається спрацьовування певної частини програми, що призводить до якого-небудь результату. [[6]](#_4i7ojhp)

**Висновок до розділу 2**

Реалізувавши та аналізувавши повний солвер «Сапера», ми змогли створити різновид гри, позбавленої від її головної проблеми: тепер неможливо програти через те, що доводиться вибирати випадково, коли вже майже вирішив все поле. Ця версія відрізняється від оригінальної гри тільки в ті моменти, коли потрібно вгадувати випадковим чином, тому можу припустити, що вона набагато цікавіше, ніж оригінальна гра.

Крім того, ми розробили варіант гри, автоматично вирішальний прості локальні правила. Чи варто використовувати таку допомогу-залежить тільки від вас. Вона зміщує фокус гри з механічного клацання до більш головоломного ігрового процесу. При цьому необов'язково користуватися удосконаленням геймплея, яке забезпечує кнопка «попросити допомоги».

**РОЗДІЛ 3 ПРАКТИЧНА ЧАСТИНА**

**3.1 Лістинг програми**

Із всіх існуючих модулей для створення нашої програми пригодиться лише

модуль random.

Першим ділом створюємо початкове головне меню, з якого гравець зможе подивитись на глобальний та локальний список 10 найкращих результатів, продовжити та почати нову гру (рис 3.1).

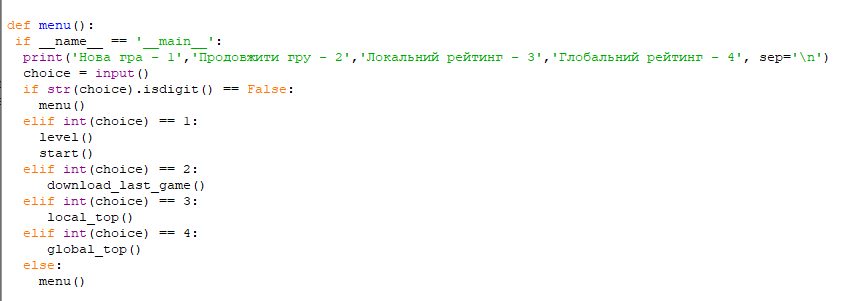
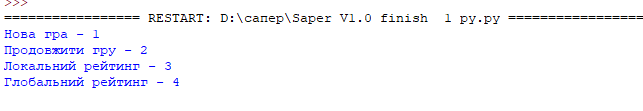
 

Рисунок 3.1 Створення початкового меню

Створюємо функцію для вибору складності гри та створення имя і паролю.

Так як розміри поля, кількість мін, ім'я та пароль не змінюються протягом всієї гри, для робимо їх глобальними змінними (рис 3.2).

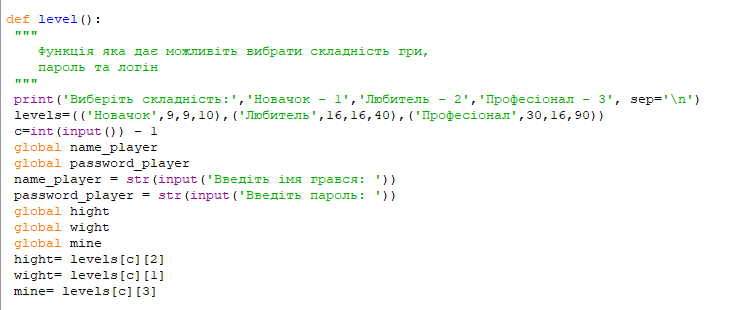


Рисунок 3.2 Створення функції для ведення параметрів: лістинг програми

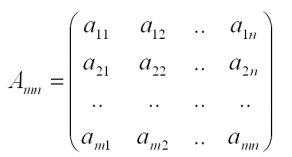


Рисунок 3.2.1 Приклад позначення елементів в матриці

Функція, яка створює пусте поле (без мін), визиває функцію і отримує параметри першої координати поля визиває функції створення ігрового поля (передає в нього перші координати та пусте поле для мін), функцію перевірки сусідніх комірок (передає перші координати та функції у якій буде відбуватися сам процес гри (рис 3.3).

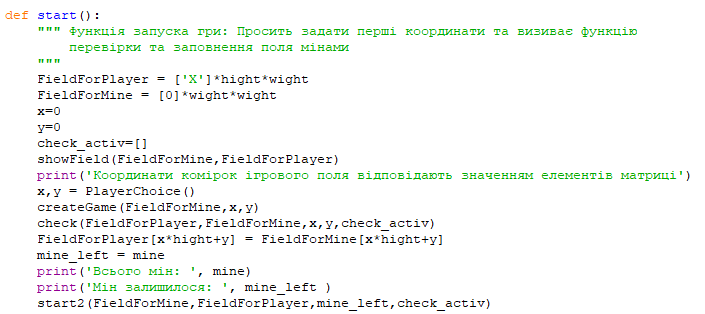


Рис 3.3 Створення функції завдання першої координати

Функція вибору, у якій гравець задає координати, після чого йде перевірка на те, чи виходить координата за межі ігрового поля (рис. 3.4). Задання координат відбувається методом значень елементів матриць (рис. 3.2.1) перше число відповідає рядкам, друге стовпця.Повертає значення координат вписаних гравцем.

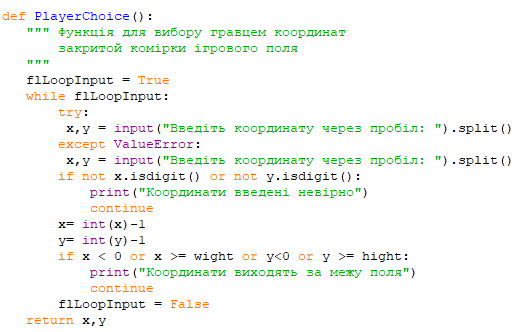


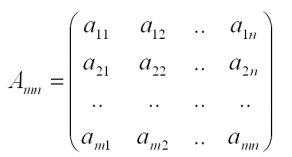
Рисунок 3.4 Завдання та перевірки координат

Рисунок 3.2.1 Приклад позначення елементів в матриці

Функція створення ігрового поля (рис 3.6), яка створює ігрове поле залежно від кількості всіх мін, ширини та висоти поля, та першої координати. Розставляє випадковим чином міни по полю.

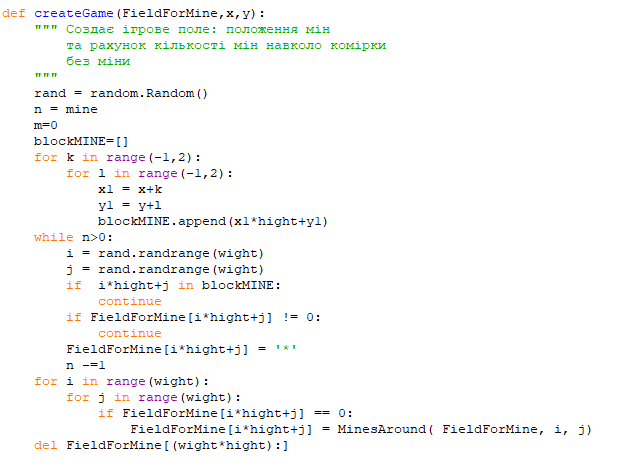


Рисунок 3.6 Створення повноцінного ігрового поля.

Функція, яка перевіряє вміст сусідніх комірок мін навколо комірки та повертає їхню кількість (рис 3.7).

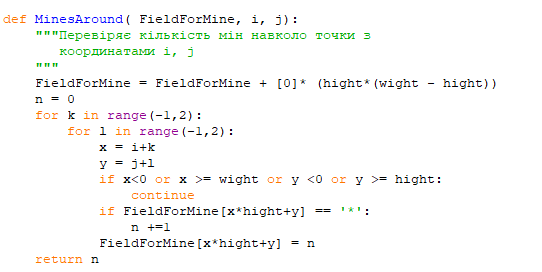


Рисунок 3.7 Функція рахування мін навколо комірки

Основна функція у якій буде відбуватися сама гра. В залежності від того що гравець хоче зробити буде виповнитися певні дії : відкриття комірок, поставлення та прибирання флагу (позначка ‘F’) та позначки «ймовірно є міна» ( позначка ‘?’), збереження гри (визиває функцію збереження гри) та виходу в головне меню (визиває функцію головного меню). (рис. 3.8). Отримає поле грався, закрите поле з мінами, кількість мін що залишилося та список координат перевірених комірок. .

Рисунок 3.8 Основна функція у якій відбувається гра.

Функція, яка показує стан ігрового поля в даний момент (рис. 3.9).Отримує поле

грався, закрите поле з мінами.Та відповідно до розмірів поля (ширини та висоти), показує гравцю ігрове поле.

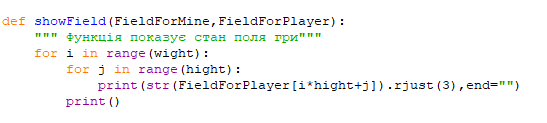


Рисунок 3.9 Відображення поля для гравця

Функція збереження гри, сохранняє імя; пароль гравця; кількість мін; кількість мін, що залишилося; висоту; ширину і записує «login.txt»; ігрове поле для гравця у файл під назвою «save\_for\_player.txt»; закрите ігрове поле з полеженнями мін, яке зберігається у двійковому код у файл «save\_mine.txt», (рис. 3.9).Отримує поле грався, закрите поле з мінами, список координат перевіриних комірок.

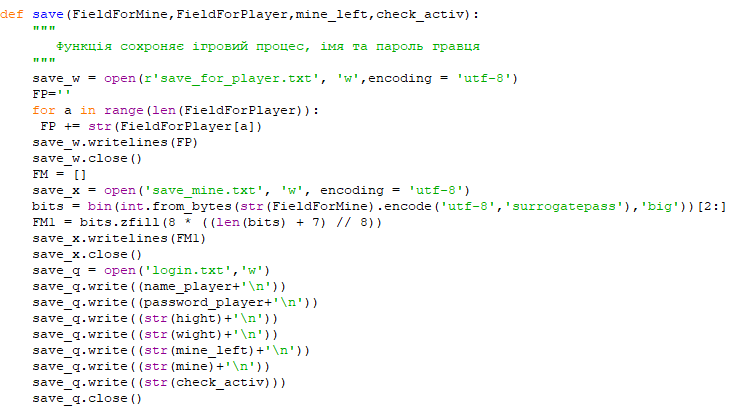


Рисунок 3.9 Функція збереження даних

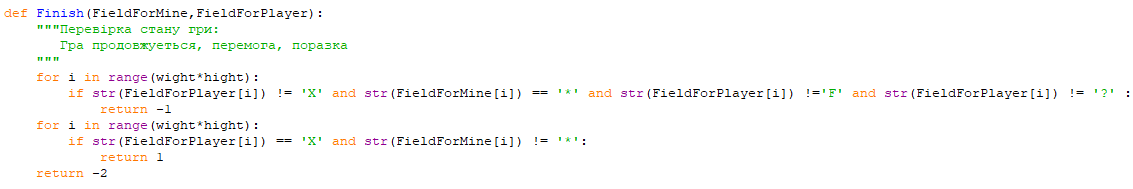
Функція, яка перевіряє стан гри. Якщо ця функція повертає -1, то гравець програл, якщо -2 гравець переміг, а коли 1, тоді гра продовжається (рис.3.10). Приймає ігрове поле гравця та закрите поле з мінами. 

Рисунок 3.10 Перевірка стану гри

Функція, яка відкриває всі пусті комірки, тобто комірки, навколо яких немає мін

(повинні мати число 0 ) та додає їхні координати до списку таких самих комірок

(рис. 3.11).

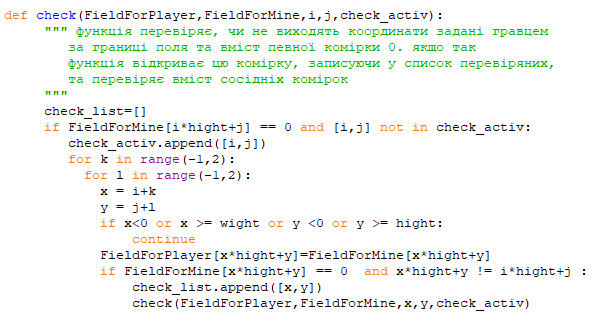


Рисунок 3.11 Функція відкриття пустих комірок

Функція розрахунку кінцевого результату та їхнього збереження за ім’ям.

Якщо гравець переміг, то за перемогу гравець отримує балів розміром кількості всіх комірок помножених на 2, за розмінування всіх мін та за всі пусті комірки. Якщо гравець програв, рахує кількість правильних розтавлених флажків (один правильний

флажок дорівнює 5 балів) та рахує всі відкриті гравцем комірки (1 відкрита комірка дорівнює 1 бал), додає результат ( ім'я, кількість очків ) до інших результатів у файл під назвою «top.txt» (рис.3.12).

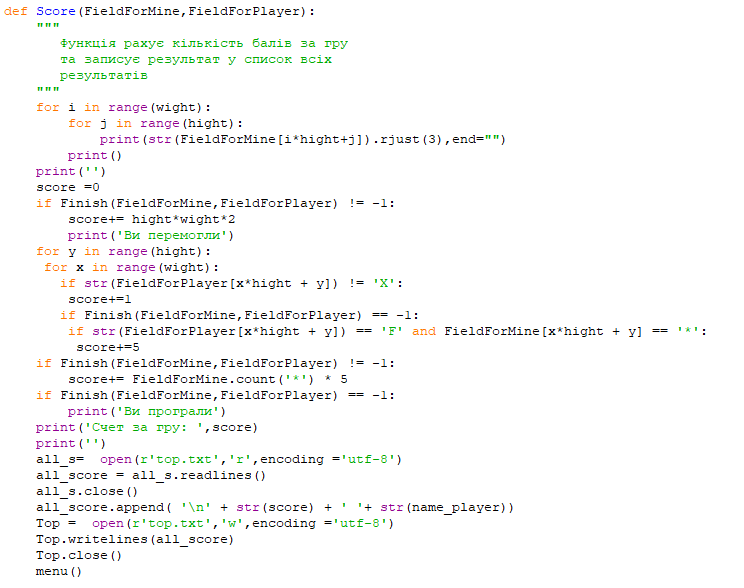


Рисунок 3.12 Функція розрахунку кінцевого результату

Функції, яка показує глобальну (рис. 3.13) та локальну (рис. 3.14) статистику гравців із 10 найкращих результатів. При виборі локального списку гравець повинен вписати ім’я гравця, якого він хоче подивитись. Відкриває та читає файл «top.txt» і відповідно до вимоги (глобальний чи локальний список) виповняє дії, закінчуючи виходом у головне меню.

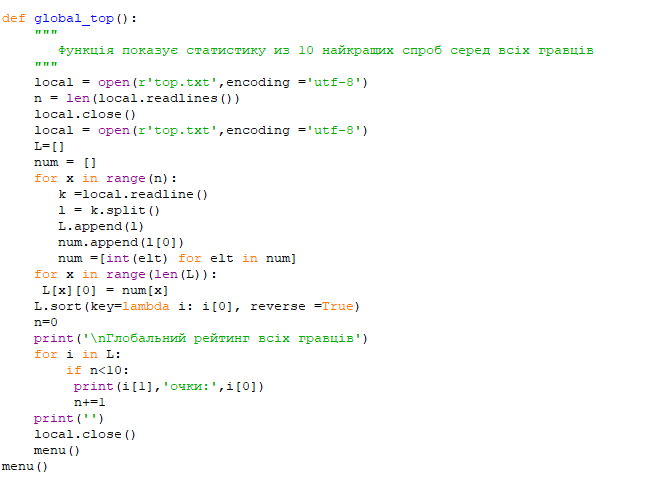


Рисунок 3.13 Функція 10 найкращих результатів серед всіх гравців

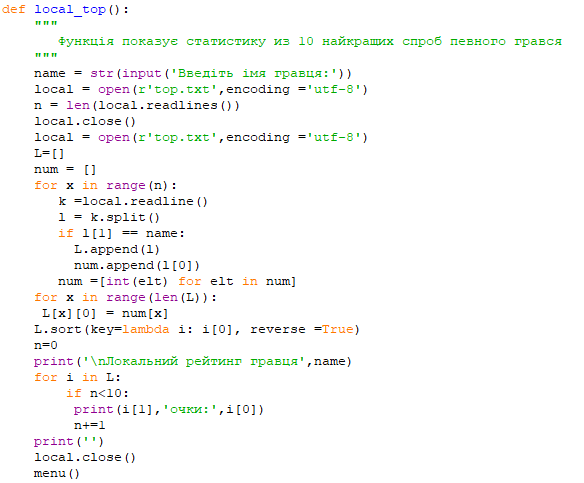


Рисунок 3.14 Функція 10 найкращих результатів певного гравця.

Функція продовження гри. Приймає та перевіряє тотожність імені та паролю.

Якщо гравець хоче повернутися в головне меню, то він повинен замість імені та паролю вписати число «1», повертає висоту та ширину ігрового поля; кількість всіх мін; кількість мін, що залишилося розмінувати; список чистих комірок; поле для гравця та розшифровує закрите від гравця поле з розташуванням мін (рис. 3.15).Читає файл «login.txt», та перевіряє ім’я та пароль, якщо ім’я та пароль були введені правильно, починає читати файли під назвами «save\_mine.txt», «save\_for\_player.txt», повернути дані та визвати функцію у якій буде відбуватися сама гра. Якщо неправильно – введення ім’я та пароль повторюється поки гравець не ввиде правильний ім’я та пароль, або не впише ‘1’, після чого гравець повернеться до головного меню.

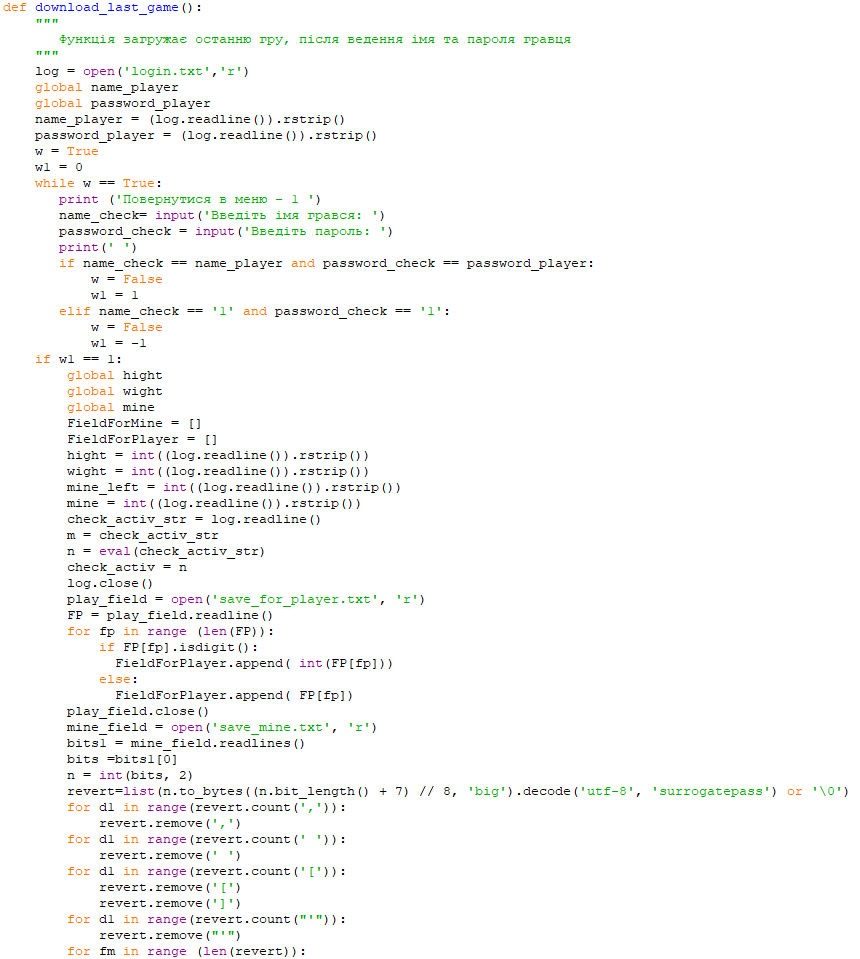
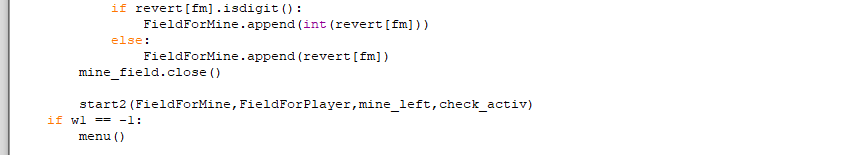


Рисунок 3.15 Функція продовження гри

Продовження рисунку 3.15



**3.2 Результати**

Після заходження в програму перше що ви побачите це головне меню гри саперу (Рис. 3.16). Тут можно вибрати одну із чотирьох дій: почати нову гру, продовжити попередню гру, показати локальний та глобальний список результатів. При введенні чисел від 1 до 4 виповнюється відповідна дія, в інших випадках гра перезапустить меню.

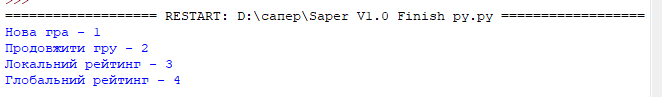


Рисунок 3.16 Головне меню

При виборі дії “Нова гра” гравцю буде запропоновано 3 складності гри: «Новачок» (поле 9x9, кількість мін - 10), «Любитель» (поле 16x16, кількість мін - 40) і «Професіонал» (поле 16x30, кількість мін - 90). Після вибору складності гравець повинен ввести ім’я та пароль за яким він буде авторизовуватися, якщо захоче продовжити гру та для просмотра результатів. Далі гравець вписує початкові координати відповідно як через пробіл (Рис. 3.17).

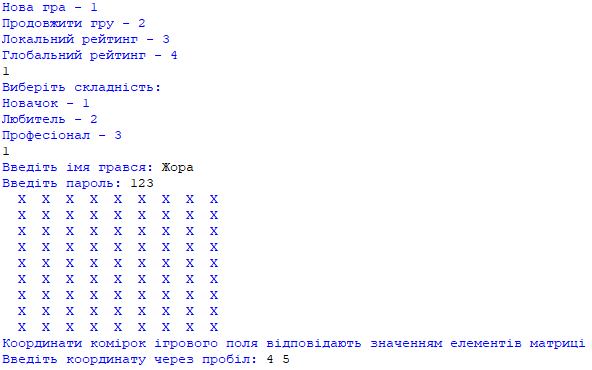


Рисунок 3.17 Вибір складності, введення першої координати, ім’я та паролю

Ігрове меню. Показує кількість всіх мін та мін, що залишилось; ігрове поле і можливі дії. Кожна дія має свою команду визову, якщо гравець введе команду яка не відповідає ніякій дії, ігрове меню перезапуститься (Рис. 3.18).

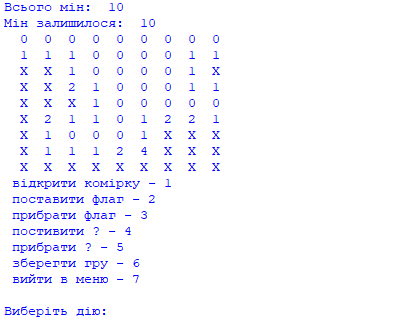


Рисунок 3.18 Ігрове меню

Закінчення гри. Гра має два випадки за якими вона закінчується: 1 - коли гравець відкриває комірку з міною (програш), 2 – коли він відкриває всі комірки які були не заміновані. Кожен з цих випадків має свій спосіб зчитування очків. За першим випадком, коли гравець програє, рахується кількість відкритих комірок (1 комірка = 1 очко), та кількість правильно поставлених флажків, тобто там де насправді стоїть міні (1 такий флажок = 5 очкам).При перемозі додає кількість балів відповідно розмірам ігрового поля, кількістю всіх мін помножених на 5 та відкритих комірок (Рис. 3.19).

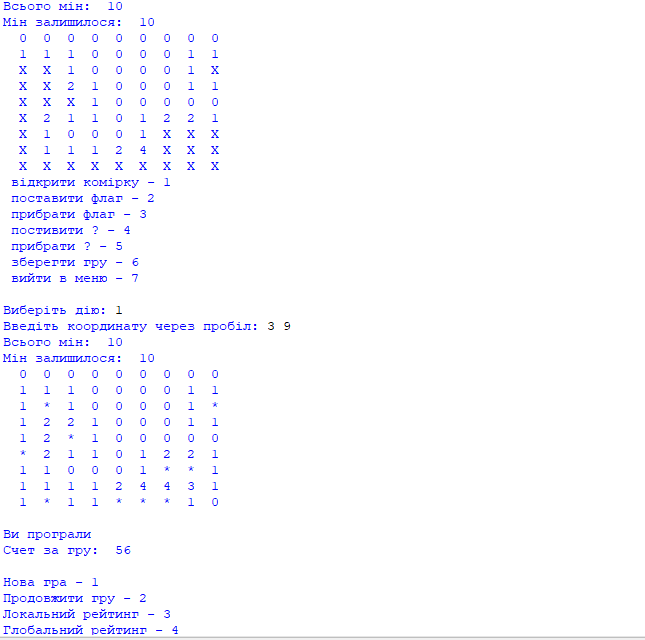


Рисунок 3.19.1 Закінчення гри- програш

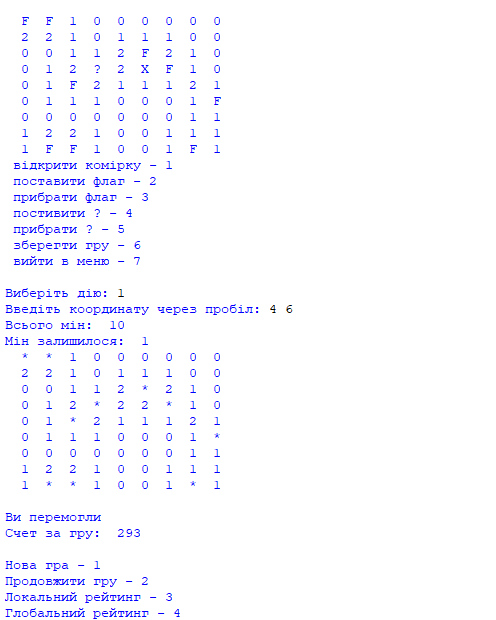


Рисунок 3.19.2 Закінчення гри- перемога

При виклику локального рейтингу просить надати ім’я гравця для якого виводить рейтинг із 10 його найкращих результатів (Рис. 3.20). При виклику глобальний рейтингу виводить рейтинг 10 найкращих результатів серед всіх гравців (Рис. 3.21).

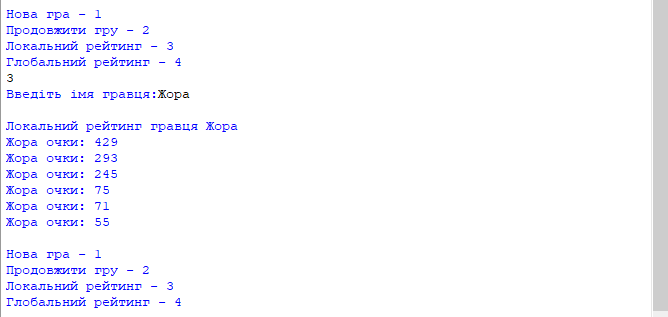


Рисунок 3.20 Локальний рейтинг

Глобальний рейтинг (Рис. 3.21):

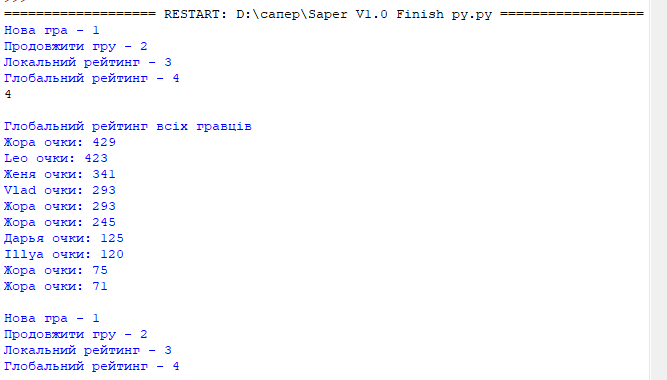


Рисунок 3.21 Глобальний рейтинг

Для збереження даних використовується чотири файли: «login.txt» в нього зберігається ім’я та пароль гравця; кількість всіх мін; мін, що залишилося; розміри ігрового поля(ширина, висота) та список відкритих комірок навколо яких немає мін (Рис. 3.22.1). «save\_for\_player.txt» ігрове поле, яке гравець бачить під час усієї гри (Рис. 3.22.2). «save\_mine.txt» закрите ігрове поле з розтушуванням мін, зберігається у двійковому коді (Рис 3.22.3). «top.txt» містить історію всіх результатів всіх закінчиних ігор (Рис 3.22.4).

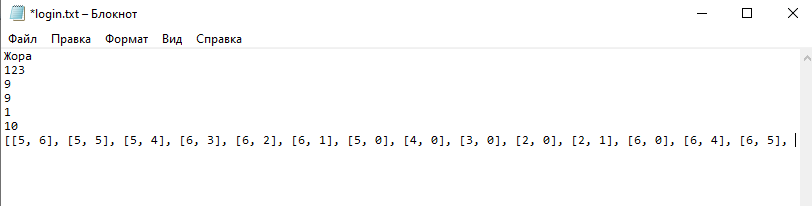


Рисунок 3.22.1 Збереження даних. Дані останньої гри.

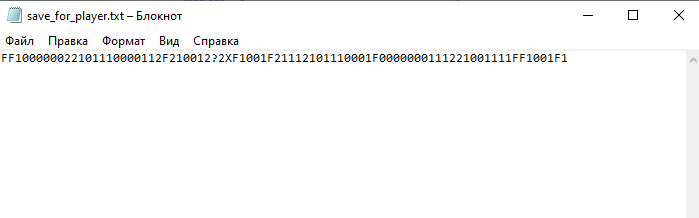


Рисунок 3.22.2 Збереження даних. Дані поля гравця

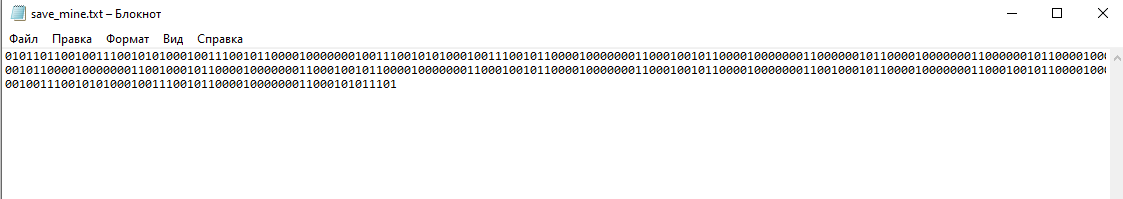


Рисунок 3.22.3 Збереження даних. Зашифровані дані поля у якому

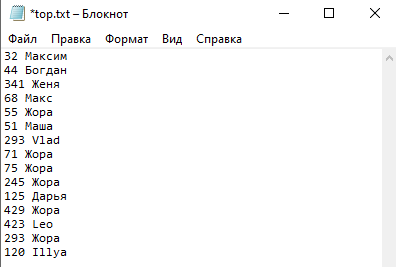


Рисунок 3.22.4 Збереження даних. Список всіх результатів

Продовження гри відбувається, якщо гравець вибрав в меню дію «Продовжити гру» та ввел правильно ім’я та пароль попередньої збереженої гри. Якщо гравець вводить неправильно ім’я або пароль, програма повторює перевірку. Для виходу у головне меню треба вписати «1» у двох перевірках (рис.3.23).

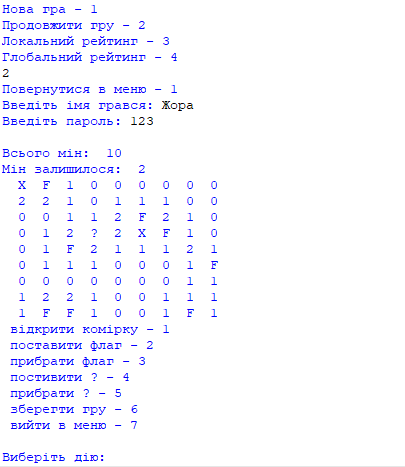


Рисунок 3.23 Продовження гри

**Висновок до розділу 3**

Ми розробили і обґрунтовували один із можливих варіантів створення програми «Cапер», виконали всі поставлені задачі: збереження гри; продовження гри; добавили можливість позначати «заміновані» комірки (щоб випадково не відкрити їх). А саме позначення комірки «ймовірно є міна» і «міна є точно»; можливість перегляду десять найкращих результатів окремого гравця й серед всіх гравців; відкриття пустих сусідніх комірок; систему очок; вибору складності гри і можливості створення ім’я та пароля. Також розглянули різні методи для вирішення кожної з поставлених задач, використовуючи різні підходи до вирішення завдань.

**ЗАГАЛЬНІ ВИСНОВКИ**

В даній роботі ми поставили завдання розробити популярну логічну гру «Сапер».

1. Призначенням гри «Сапер» є розвиток логічного мислення, Розвиток пам'яті і логіки. Після занять з логічними іграми розвивається не просто мислення, а і його швидкість, людина починає все швидше вирішувати проблеми і швидше знаходити правильні рішення на поставлені завдання. Гра "Сапер" допомагає зосередитися, а також відволіктися від усього іншого і дратівливого, граючи в неї, ми розвиваємо пам'ять і швидкість мислення.
2. Основною загальнотеоретичною проблемою під час виконання поставленої задачі був вибір мови та середовища програмування, розробити програмну реалізацію логічної гри «Сапер» відповідно поставленим завданням та скласти опис функціональних можливостей розробленої програми.
3. Реалізовуємо гру «Сапер» за допомогою метода програмування «зверху вниз», створюючи основні функції та після визначення і створення плану послідовності виконання поставлених задач з різними методами виконання кожного завдання, поступово описувати кожну функцію.
4. Для виконання цього завдання треба вибрати мову програмування і системи (середовища) програмування, визначитися з методами вирішення задачі, використовуючи різноманітні способи реалізування програми, визначити всі недоліки та переваги рішень, створити план послідовності виконання поставлених задач й поступово логічно пов’язувати їх між собою.
5. У результаті в нас повинна вийти повноцінна гра «Сапер», яка, крім основних своїх функцій, може виводити десять найкращих результатів окремого гравця і серед всіх гравців, виповняти збереження гри та завантаження останньої сесії.
6. Однією з проблем під час створення гри «Сапер» було реалізація відкриття та перевірка сусідніх комірок на знаходження в них мін. Я усунув цю проблему завдяки одновимірному масиву та формули для нього, яка перевіряє кожну сусідню комірку.

**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

**Базова література**

1. Доусон М. Программируем на Python. — СПб.: Питер, 2014. — 416 с.: ил. —

С. 277-307.

2. Лутц М. Изучаем Python. — М.: Символ-Плюс, 2011. [Електронний ресурс] /

Лутц М. - Режим доступу: Google клас. - Основи програмування. - ОСНОВИ

ПРОГРАМУВАННЯ НА PYTHON. - Лутц программирование на Питон.

<https://classroom.google.com/u/1/w/MTYwMzIyNzgwMjU3/t/all>

3. Яковенко А. В. Методичні вказівки до виконання комп’ютерних

практикумів з дисципліни «Основи програмування». Основи програмування

мовою Python. [Електронний ресурс] / Яковенко А. В. -Режим доступу: Google

клас. - Основи програмування.- ОСНОВИ ПРОГРАМУВАННЯ НА

PYTHON.-

Методичні основи програмування. -

<https://classroom.google.com/u/1/w/MTYwMzIyNzgwMjU3/t/all>

**Додаткова література**

4. Коэльё Л. П, Ричерт В. М. Построение сиситем машиного обучения на языке Phyton ДМК Пресс, (2016) – 297 с.

5. Прохорёнок Н. А. Python3. Самое необходимое / Н. А. Прохорёнок, В. А.Дронов. — СПб.:"БХВ-Петербург", 2016. — 461 с.

.

**Інформаційні джерела**

6.  [Hovland M. H.](https://magnushoff.com/) Solving Minesweeper, 2015 [Електронний ресурс] / ]. – Режим доступу:

[https://magnushoff.com/articles/minesweeper](https://magnushoff.com/articles/minesweeper/)/

7. Python 3 для начинающих. Карта сайта [Електронний ресурс]. – Режим доступу:

<https://pythonworld.ru/karta-sajta>