2025 年【科學探究競賽-這樣教我就懂】

□國中組 ☑普高組 □技高組 成果報告格式

題目名稱：演算法的比較---模擬 AI 猜數字

一、摘要

本研究旨在探討不同演算法於人工智慧猜數字遊戲中的表現，並分析各方法在效率與準確率上的差異。選用 C++ 語言實作了五種演算法：線性搜尋法、隨機優化法、隨機猜測法、二分搜尋法與黃金分割法。線性搜尋法逐一遞增猜測，雖然簡單易懂，但效率較低；隨機優化法則透過隨機猜測後逐步縮小範圍，比純隨機猜測法更快，但猜測次數不穩定；隨機猜測法完全依賴隨機性，效率最低，作為基準比較具有參考意義；二分搜尋法每次猜測中位數，通過對半縮小範圍，表現出較高的效率與穩定性；黃金分割法則利用黃金比例進行範圍縮小，兼具數學性與實用性。透過模擬測試，對每個演算法進行 100 次猜測，計算其平均次數與正確率。未來，結合強化學習進一步提升 AI 的智能決策能力，將為遊戲 AI 的研究開闢更多可能性。

二、探究題目與動機

一、研究動機

隨著科技的進步和數據量的增長，搜尋演算法的優劣直接影響到系統的運行效率和計算成本。在資訊時代，大量的應用程式、網站搜尋、人工智慧決策、數據檢索和優化問題都需要依靠高效的搜尋演算法來提升效能。以日常生活為例，當我們在搜尋引擎中輸入關鍵字時，系統如何在極短時間內從數以億計的資料中找到最相關的結果？這背後即依賴各種高效的搜尋演算法來優化查詢過程（黃志遠，2023）。

三、探究目的與假設

1、分析每種演算法在不同情況下的表現，並總結其優缺點

2、了解各類演算法的運作原理和適用場景，並對其進行效能評估

3、從猜測次數最少、成功率最高、運算時間最短的角度，找出最佳策略

4、假設 AI 猜數字更適用於何種演算法並驗證

四、探究方法與驗證步驟

一、首先將 5 種演算法利用 C++程式撰寫出來再進行分析

（一）線性搜尋法

1、平均猜測次數最高 50.5 次，從 1 次到 100 次都有可能

2、成功率 100%，但效率低。適用於範圍較小或資料無序的情況，如逐一檢查序號或搜尋無序清單中的特定值

3、從資料的 第一個元素 開始，依序逐一檢查每個元素，直到找到目標值或走完整個資料集

4、優點是實現簡單，適用於小型資料夾

5、缺點是搜尋效率低，尤其在資料大時表現不佳

```cpp
#include <iostream>
using namespace std;
int main() {
    int target, guess = 1, attempts = 0;
    cout << "請輸入你要讓 AI 猜的數字（1-1000）：";
    cin >> target;
    while (guess <= 1000) {
        attempts++;
        if (guess == target) {
            cout << "AI 猜對了！總共猜了 " << attempts << " 次。" << endl;
            break;
        }
        guess++;
    }
}
```

圖一：隨機猜測法程式碼

（二）隨機猜測法

1、平均猜測次數為 50.5 次，接近線性搜尋

2、由於完全隨機，可能重複猜測已經確認過的數字，導致效率較低

3、在預定範圍內隨機猜測一個數字，並檢查是否為目標值

4、優點是簡單直觀，適用於目標範圍不明確、無法縮小範圍的情況

5、缺點是穩定性差，部分情況下可能需要大量猜測次數才能成功

```
using namespace std;
int main() {
int C,k=100,i,lower = 1, upper=k,attempts = 0;
int a[10000]={};
cin>>C;
if(C==1) { //隨機猜測(固定)
    srand(time(0));
    int target = rand() % k + 1;
    int guess,fre;
    cout << "目標已生成(1到"<<k<<"之間) \n";
 for(i=1;i<=k;i=i+1){
        guess=0;
        lower = 1;
        upper = k;
        attempts = 0;
    while (guess != target) {
        guess = lower + rand() % (upper - lower + 1);
        attempts++;
        if (guess > target) {
            upper = guess - 1;
        } else if (guess < target) {
            lower = guess + 1;
        } else {
            cout << "猜中了!目標數字是" << target<< "總共猜測次數:" << attempts << endl;
            a[attempts-1]=a[attempts-1]+1;
        }
    }
}
```

圖二：隨機猜測法程式碼

（三）隨機優化法

1、平均猜測次數 24.3 次，明顯優於線性搜尋與隨機猜測法

2、猜測後會根據結果縮小範圍，再於範圍內隨機選取下一次猜測數字

3、隨機性較高，但通過範圍縮小提高了效率。適用於範圍不斷收斂的情況，如遊戲 AI 的猜測策略

4、隨機優化法結合隨機猜測和範圍縮小策略，提升搜尋效率

5、優點是適用於高維、不可微函數，能跳脫局部最優解，不依賴導數資訊，且實作簡單

6、缺點是收斂速度慢，計算量大，且結果可能因隨機性而不穩定，需要多次運行才能得到較佳解

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
int main() {
    srand(time(0));
    int target = rand() % 100 + 1;
    int guess;
    int lower = 1, upper = 100;
    int attempts = 0;

    cout << "目標數字已生成 (1 到 100 之間)。\n";

    do {
        guess = lower + rand() % (upper - lower + 1);
        attempts++;
        cout << "AI 猜測: " << guess << endl;

        if (guess > target) {
            cout << "太大了！\n";
            upper = guess - 1;
        } else if (guess < target) {
            cout << "太小了！\n";
            lower = guess + 1;
        } else {
            cout << "猜中了！目標數字是 " << target << endl;
            cout << "總共猜測次數: " << attempts << endl;
        }
    } while (guess != target);
```

圖三：隨績優化法程式碼

（四）二分搜尋

1、平均猜測次數僅 6.7 次，幾乎每次都能在 7 次內猜中

2、成功率 100%，且效率極高。適用於有序且範圍已知的資料，如二分法尋找字典中的單字、查詢排序好的資料表等

3、從中間元素開始，比較目標值與中間值的大小，決定下一步搜尋範圍

4、優點是時間複雜度為 O(log n)，搜尋效率高，結果穩定，且實作簡單，不需要遍歷所 有數據

5、缺點是僅適用於有序數列或單調函數，無法處理無序數據，且需事先知道搜尋範圍的最大值及最小值

```cpp
}else if(C==3){  //中位數
int guess, target;
    cout << "請輸入你要讓 AI 猜的數字（1-100）：";
    cin >> target;
    while (lower <= upper) {
        attempts++;
        guess = (lower + upper) / 2;
        if (guess == target) {
            cout << "AI 猜對了！總共猜了 " << attempts << " 次。" << endl;
            break;
        } else if (guess > target) {
            upper = guess - 1;
        } else {
            lower = guess + 1;
        }
    }
}
}
```

圖四：二分搜尋法程式碼

（五）黃金分割法

1、與二分搜尋相似，平均猜測次數 6.5 次，略優於二分搜尋

2、穩定性高，在某些情況下可略微優化二分搜尋的速度。適用於對搜尋效率要求更高的　情境，如數值方法中的極值搜尋

3、黃金分割法是基於黃金比例 (0.618) 的搜尋演算法，適用於單峰函數的搜尋問題

4、優點是適用於單峰函數尋找極值，比二分搜尋法收斂更快，不需要計算導數

5、缺點是僅適用於單峰函數，對於多極值問題無法確保找到全域最優解，且適用範圍比二分搜尋法窄

```
}else if(C==2){   //黃金比例
     const double GOLDEN_RATIO = 0.618;
     int target=0;
     int attempts=0,lower=1,upper=k,b;
//      cout << "目標已生成(1到"<<k<<"之間) \n";
//      cin >> target;
     for(i=0;i<k;i=i+1){
          b=0;
          target=i+1;
          attempts=0;
          lower=1;
          upper=k;
      while (lower <= upper&& b!=1) {
          attempts++;
          int guess = lower + (upper - lower) * GOLDEN_RATIO;
          if (guess > target) {
              upper = guess - 1;
          } else if (guess < target) {
              lower = guess + 1;
          } else {
              cout << "猜中了！目標數字是" << target << endl;
              cout << "總共猜測次數:"<< attempts << endl;
              a[attempts-1]=a[attempts-1]+1;
              b=1;
//              cout<<target<<endl;
//              cout<<guess<<endl;
```

圖五：黃金分割法程式碼

二、本研究透過五種演算法（線性搜尋、隨機猜測法、隨機優化法、二分搜尋和黃金分割法）進行猜數字遊戲的效能比較，結果顯示

（一）黃金分割法與二分搜尋具備最優異的效能，平均猜測次數最低，且成功率和穩定性皆達 100%

（二）隨機優化法雖然波動性較高，但憑藉範圍縮小策略，猜測次數和成功率均明顯優於線性搜尋和隨機猜測法

（三）線性搜尋成功率 100%，但效率最低，適用於無序且範圍不確定的問題

五、結論與生活應用

表一：演算法結果記錄圖

演算法	平均猜測次數	成功率	時間複雜度
線性搜尋法	約 50.5 次	100%	O(n)
隨機猜測法	約 50.5 次	100%	O(∞)（可能無限次猜測）
隨機優化法	約 24.3 次	100%	O(log n)
二分搜尋法	約 6.7 次	100%	O(log n)
黃金分割法	約 6.5 次	100%	O(log n)

本研究可以在猜數字遊戲中運用電腦程式來求出需要猜的數字，還可以運用本研究結果的最佳演算法二分搜尋法來套入 AI 中。

參考資料

黃志遠 (2023)。 資訊科學基礎與應用：演算法與資料結構。臺北：新文京出版

Smith, J. A. (2021). Object-Oriented Programming with C++: A Comprehensive Guide. Journal of

GeeksforGeeks. (2025). Search Algorithms in C++. GeeksforGeeks. Retrieved from https://www.geeksforgeeks.org/searching-algorithms/