Nombre mystère [lp06] - Exercice résolu

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Université Haute-Alsace

Version 17 mai 2018

Table des matières

C++ - Nombre mystère (Solution)

Mots-Clés Structures répétitives ■
Requis Structures de base, Structures conditionnelles ■
Difficulté • ○ ○ (30 min) ■

Objectif
Cet exercice permet à un joueur de deviner un entier secret tiré par le meneur de jeu.

1 Nombre mystère / pgmystere

1.1 Énoncé

Le « Nombre mystère » consiste à demander, à un meneur de jeu, un entier compris entre 0 et 100 (nombre mystère), puis à demander au joueur (qui n'a pas vu le nombre) de le deviner en faisant une suite de propositions. A chaque proposition du joueur, l'algorithme répond par « Trop petit » ou « Trop grand ». Lorsque le joueur a trouvé le nombre mystère, l'algorithme annonce le nombre de propositions qu'il a dû faire pour trouver. Le meneur de jeu ainsi que le joueur sont sensés donner des nombres compris entre 0 et 100, mais aucun contrôle n'est fait.

Objectif
Écrire un algorithme du jeu.

Remarque
Cet exercice donne l'occasion d'analyser le même problème de deux manières différentes, toutes deux correctes, mais conduisant à l'utilisation de deux répétitives différentes.

1.2 Nombre mystère (1)

Les variables
Le choix des variables est guidé par l'énoncé. Il en faut une pour le nombre mystère (`mystere`), une qui recevra les entiers proposés successivement par le joueur (`njoueur`), et une pour compter les essais (`nessais`). Toutes ces variables doivent être de type entier.

Déroulement de l'algorithme
Le déroulement de cet algorithme doit être une répétition de séquences comprenant la demande d'un entier, l'analyse de cet entier et l'affichage de la réponse adéquate. Face à un problème qui nécessite manifestement une répétitive, il faut se poser plusieurs questions :

- Comment obtient-on l'élément courant ?
- De quoi est fait le traitement qui s'applique à l'élément courant ?
- Comment s'arrête la répétition ?
- Le traitement est-il forcément exécuté une fois ?
- Comment faut-il initialiser le traitement ?

Essayons de répondre à ces questions pour le problème posé :

- L'élément courant est la valeur que le joueur propose à chaque tentative : il est représenté par la variable `njoueur`. Il est obtenu par une saisie auprès du joueur.
- Le traitement de l'élément courant consiste à incrémenter le nombre d'essais et à comparer la valeur proposée avec le nombre mystère, d'où trois cas :
  - `njoueur = mystere` : c'est gagné. Arrêt de la répétition

— `njoueur < mystere` : c'est trop petit

— `njoueur > mystere` : c'est trop grand

Modèle de l'algorithme

La répétition s'arrête dès que le nombre proposé est égal au nombre mystère. Il faut que cette dernière valeur, la bonne, soit comptée dans le nombre d'essais. Il y a forcément un premier essai qui sera comparé au nombre mystère et qui recevra un message. Nous sommes donc poussés à choisir l'instruction `Répéter` qui s'exécute au moins une fois. L'algorithme suivra le modèle :

```
Répéter
|  saisir l'élément courant
|  traiter l'élément courant
Jusquà ce que la valeur soit la dernière
```

Initialisations

Pour que la répétitive puisse s'effectuer correctement, les variables `mystere` et `nessais` doivent avoir une valeur avant le début de la répétitive. C'est pourquoi celle-ci doit être précédée d'une phase d'initialisation qui les initialisent.

Écrivez un programme qui traduit cette analyse.
Le meneur de jeu sera la machine ce qui permettra de masquer le nombre mystère.

Testez.

Validez votre programme avec la solution.

Solution C++ @[pgmystere1.cpp]

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

int main()
{
  srand(time(0));
  int mystere = rand()%100 + 1;
  int nessais = 0;
  int njoueur = -1;
  while (njoueur != mystere)
  {
    cout<<"Votre entier [1..100]? ";
    cin>>njoueur;
    ++nessais;
    if (njoueur == mystere)
    {
      cout<<"==> Gagne"<<endl;
    }
```

```
    else if (njoueur < mystere)
    {
      cout<<"==> Trop petit"<<endl;
    }
    else
    {
      cout<<"==> Trop grand"<<endl;
    }
  }
  cout<<"Trouve en "<<nessais<<" essai(s)"<<endl;
}
```

1.3 Nombre mystère (2)

Autre analyse

Proposons une deuxième analyse, un peu différente, qui considère que l'élément courant traité dans la répétitive concerne uniquement les **mauvaises** tentatives. Le traitement consiste à incrémenter le nombre de mauvaises tentatives et à comparer la mauvaise valeur au nombre mystère pour annoncer si elle est trop petite ou trop grande. La répétitive s'arrête dès que le nombre proposé est égal au nombre mystère. Si le joueur trouve du premier coup, il n'y a aucune mauvaise valeur à traiter, donc aucune entrée dans la boucle. Il n'est donc plus possible d'utiliser une instruction `Répéter` : il faut utiliser une instruction `TantQue`. Reste à la construire correctement. Si on entre dans la boucle, c'est que la valeur saisie lors de la tentative est mauvaise, il faut donc traiter cette valeur avant de saisir la valeur suivante. De ce fait, les phases de traitement et d'acquisition de valeur doivent être écrites dans un ordre différent de celui de la répétitive `Répéter`.

La phase d'initialisation, qui précède la répétitive, doit compter la première tentative. La variable `nessais` sera donc initialisée à 1 et non à 0.

Écrivez un programme qui traduit cette deuxième analyse.
Ici aussi l'entier caché sera tiré par la machine.

Testez.

Validez votre programme avec la solution.

Solution C++ @[pgmystere2.cpp]

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

int main()
{
  srand(time(0));
  int mystere = rand()%100 + 1;
  int nessais = 0;
```

```
  int njoueur;
  cout<<"Votre entier [1..100]? ";
  cin>>njoueur;
  ++nessais;
  while (njoueur != mystere)
  {
    if (njoueur < mystere)
    {
      cout<<"==> Trop petit"<<endl;
    }
    else
    {
      cout<<"==> Trop grand"<<endl;
    }
    cout<<"Votre entier [1..100]? ";
    cin>>njoueur;
    ++nessais;
  }
  cout<<"Trouve en "<<nessais<<" essai(s)"<<endl;
}
```

1.4 Nombre mystère avec aide stratégique

Cette fois on veut aider le joueur à répondre intelligemment : la première fois, on lui précise qu'il doit donner un nombre compris entre 1 et 100, au tour suivant l'une des deux bornes doit être modifiée en fonction de sa réponse précédente. Par exemple, si le nombre mystère est 20 et que le joueur donne comme première proposition, qui est trop grand, au tour suivant l'algorithme doit lui demander de saisir un entier entre 1 et 35 et non entre 1 et 100.

Modifications

La modification concerne l'affichage des bornes à respecter par le joueur. Elle peut se faire indifféremment sur l'une ou l'autre des versions. Les bornes ne sont plus 1 et 100, mais vont changer : il faut donc déclarer des variables supplémentaires entières `binf` et `bsup`, pour ranger les valeurs des bornes. Ces deux variables seront initialisées à entre 1 et 100, puis l'une ou l'autre changera à chaque tour. Si le joueur donne une valeur trop petite, la nouvelle borne inférieure prendra la valeur $+1$; si sa valeur est trop grande, la nouvelle borne supérieure prendra la valeur -1.

Écrivez un programme qui traduit cette analyse.
Ici aussi l'entier caché sera tiré par la machine.

Testez.

Validez votre programme avec la solution.

Solution C++ @[pgmystere3.cpp]

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

int main()
{
  srand(time(0));
  int binf = 1;
  int bsup = 100;
  int mystere = rand()%bsup + binf;
  int nessais = 0;
  int njoueur = -1;
  while (njoueur != mystere)
  {
    cout<<"Votre entier ["<<binf<<".."<<bsup<<"]? ";
    cin>>njoueur;
    ++nessais;
    if (njoueur == mystere)
    {
      cout<<"==> Gagne"<<endl;
    }
    else if (njoueur < mystere)
    {
      cout<<"==> Trop petit"<<endl;
      binf = njoueur + 1;
    }
    else
    {
      cout<<"==> Trop grand"<<endl;
      bsup = njoueur - 1;
    }
  }
  cout<<"Trouve en "<<nessais<<" essai(s)"<<endl;
}
```

2 Que retenir de cet exercice?

Lorsque l'on traite un problème où le nombre de répétitions ne peut pas être connu à priori, on est le plus souvent amené à choisir l'un des deux modèles suivants :

- Le traitement répétitif s'arrête après le traitement d'un élément particulier qui doit être traité comme les autres :

```
Répéter
|  obtenir l'élément courant
|  traiter l'élément courant
Jusqu'à ce que l'élément courant soit le dernier
```

- Le traitement répétitif s'arrête après l'acquisition d'un élément particulier (*sentinelle*) qui ne doit pas être traité comme les autres :

```
obtenir le premier élément
TantQue l'élément n'est pas la sentinelle Faire
```

```
|  traiter l'élément courant
|  obtenir l'élément courant
FinTantQue
```

3 Références générales

Comprend [Tartier-AL1 :c5 :ex14] ■