Jeu du pendu [ka07] - Exercice

Karine Zampieri, Stéphane Rivière, Béatrice Amerein-Soltner

Unisciel algoprog Version 21 mai 2018

Table des matières

C++ - Jeu du pendu (Solution)

Mots-Clés Jeux de logique, Classes abstraites ■
Requis Classes, Pointeurs, Héritage, Polymorphisme d'inclusion ■
Fichiers Pendaison ■
Difficulté ●●○ (2 h) ■

Objectif
Cet exercice réalise une version orientée objet polymorphique du jeu du Pendu.

1 Présentation du jeu

La version officielle du « Jeu du Pendu » se joue à plusieurs personnes. Vous supposerez ici que le jeu se joue à deux.

L'un des joueurs commence par proposer un mot à deviner. Supposez que l'autre joueurs ne voit pas ce mot. Il n'en connait au départ que la longueur. Ce deuxième joueur propose ensuite une lettre. Si cette lettre est présente dans le mot, toutes ses occurrences sont indiquées. Sinon, ce joueur « progresse » sur le chemin de la pendaison. Au bout d'un certain nombre `NMAX` étapes, il a perdu : il est « pendu ».

La partie se termine lorsque chacun des deux joueurs a proposé à l'autre un mot à deviner. Chaque joueur qui parvient à en pendre un autre gagne un point. Chaque joueur qui trouve un mot (c.-à-d. propose la dernière lettre à trouver) gagne un point. A la fin de la partie, gagne celui qui est encore en vie et a le plus de points.

Au terme de la pendaison (`NMAX`-ème étape), le pendu pourrait ressembler à un petit pendu (ici 12 essais) :

```
. ______
  |/  |
  |  _0_
  |   I
  |  / \
_/|\_____  #e= 12
```

On peut choisir ici un affichage totalement différent. Le but n'étant pas de perdre de temps à essayer de faire un joli dessin (ce qui compte c'est que chaque joueur voit clairement à chaque tour de jeu à quel stade il en est), vous pouvez utiliser la classe `PendaisonC1` fournie en téléchargement qui permet l'affichage du pendu.

Plusieurs joueurs

Le jeu du pendu pouvant se jouer à plusieurs joueurs (un qui propose le mot et les autres essayant à tour de rôle de le deviner), la solution proposée ne fait pas d'hypothèse sur le nombre de joueurs.

2 Algorithmique, Programmation

2.1 Classe Joueur

Créez une classe de base `Joueur` représentant les joueurs.
Incluez :

- Le nom (chaîne de caractères) du joueur.
- Le numéro de l'étape (entier positif) atteinte par le joueur.
- Le nombre de points (entier positif) gagné par le joueur.

Écrivez un constructeur à un paramètre (le nom du joueur) initialisant les attributs.

Écrivez une méthode `incrPoints` qui incrémente le nombre de points du joueur.

Écrivez une méthode `incrEtape` qui incrémente le nombre d'étapes du joueur.

Écrivez une méthode abstraite `proposerMot` qui permet au joueur de proposer un mot (à faire deviner à l'autre).

Écrivez une méthode abstraite `proposerLettre` qui permet au joueur de proposer une lettre (lorsqu'il essaie de deviner un mot).

Validez votre classe et vos méthodes avec la solution.

Solution C++ @[Joueur.hpp] @[Joueur.cpp]

```cpp
#ifndef JPJOUEUR_CLASS
#define JPJOUEUR_CLASS
#include <iostream>
#include <string>
using namespace std;

/**
  Represente des joueurs au "Jeu du Pendu"
  super-classe abstraite
*/
class Joueur
{
  public:
    Joueur(const string& n);
    virtual ~Joueur();
    const string& getNom() const;
    int getPoints() const;
    int getEtape() const;
    void incrPoints();
    void incrEtape();
    void reset();

    // Permet au joueur de proposer un mot
    virtual string proposerMot() const = 0;
    // Permet au joueur de proposer une lettre
    virtual char proposerLettre() const = 0;

  private:
    string m_nom; // nom du joueur
    int m_npoints; // # de points cumules
    int m_etape; // etape
};
#include "Joueur.cpp"
#endif
```

```
/**
  Constructeur normal
*/
Joueur::Joueur(const string& n)
: m_nom(n), m_npoints(0), m_etape(0)
{}

/**
  Destructeur virtuel
*/
Joueur::~Joueur()
{}

/**
  Accesseur le nom du joueur
*/
const string& Joueur::getNom() const
{
  return m_nom;
}

/**
  Accesseur du nombre de points du joueur
*/
int Joueur::getPoints() const
{
  return m_npoints;
}

/**
  Accesseur de l'etape atteinte par le joueur
*/
int Joueur::getEtape() const
{
  return m_etape;
}

/**
  Modifieur du nombre de points du joueur
*/
void Joueur::incrPoints()
{
  ++m_npoints;
}

/**
  Modifieur du numero etape
*/
void Joueur::incrEtape()
{
  ++m_etape;
}

/**
  Reinitialise le # d'etapes a zero avant
  de recommencer une partie
*/
void Joueur::reset()
```

```
{
  m_etape = 0;
}
```

2.2 Classe Pendu

Créez une classe `JeuPendu` permettant de jouer une partie de « pendu ». Incluez parmi ses attributs :

- Le mot à-deviner (chaîne de caractères).
- La tentative (chaîne de caractères) (c.-à-d. le mot tenté) de l'adversaire.
- Un vecteur de pointeurs sur `Joueur`.

Écrivez un constructeur prenant en paramètre un vecteur de pointeurs sur `Joueur`.

Écrivez une méthode interne `initADeviner(k)` qui permet au joueur numéro `k` de proposer un mot à deviner.

Aide simple
Ce mot sert à initialiser le mot à-deviner et est généré grâce à la méthode `proposerMot` de la classe `Joueur`.

Écrivez une méthode interne `initTentative` qui initialise la tentative avec des blancs soulignés (`'_'`). Exemple : Si à-deviner vaut « test », la tentative vaudra « `____` ».

Écrivez une méthode interne `estDansMot(c)| qui teste et renvoie \lstinline`Vrai| si une lettre `c` (caractère) se trouve dans le mot à-deviner.

Écrivez une méthode interne `inclureDansMot(c)` qui remplace le blanc souligné correspondant dans la tentative par la lettre qui se trouve dans le mot à-deviner.

Soit la classe `Pendaison` qui affiche la pendaison d'ordre `n`. Chaque ligne du tableau interne correspond à l'affichage devant se faire à l'étape d'une pendaison.

C++ @[Pendaison.hpp]

Écrivez une méthode interne `devinerMot(k)` qui, étant donné l'indice du joueur `k` ayant proposé le mot à-deviner, permet à l'autre d'essayer de deviner le mot caché. La méthode fait réessayer le joueur tant que le mot n'est pas deviné ou qu'il n'a pas été pendu.

Aide simple

Utilisez la méthode `Joueur::proposerLettre` pour permettre au joueur qui doit deviner le mot de proposer une lettre. A chaque étape, c'est soit le mot partiellement deviné qui est affiché (succès), soit la progression de la figure du pendu (échec). Pour réaliser l'affichage de la pendaison d'ordre `n|`, appelez la méthode `Pendaison::afficher(n)`.

Écrivez une méthode `jouer` qui permet à chaque joueur de proposer à l'autre un mot à deviner.

Écrivez une méthode `afficher` qui affiche le résultat de la partie.

Validez votre classe et vos méthodes avec la solution.

Solution C++ @[JeuPendu.hpp] @[JeuPendu.cpp]

```
#ifndef JPPENDU_CLASS
#define JPPENDU_CLASS
#include <iostream>
#include <cstdlib>
#include <string>
#include <vector>
using namespace std;
#include "Pendaison.hpp"
#include "Joueur.hpp"

/**
  Represente des parties du jeu du pendu
*/
class JeuPendu
{
  public:
    JeuPendu(const vector<Joueur*>& j);
    void afficher() const;
    void jouer();

  protected:
    bool estDansMot(char c) const;
    void inclureDansMot(char c);
    void initADeviner(unsigned k);
    void initTentative();
    bool devinerMot(unsigned k);

  private:
    const vector<Joueur*> &m_joueurs;
    string m_adeviner; // mot cache
    string m_tentative; // mot tente
};
#include "JeuPendu.cpp"
#endif
```

```
/**
  Constructeur normal
*/
```

```
JeuPendu::JeuPendu(const vector<Joueur*>& j)
: m_joueurs(j)
{}

/**
  Affiche les resultats d'une partie
*/
void JeuPendu::afficher() const
{
  cout<<endl<<"La partie est finie. Les resultats:"<<endl;
  for (unsigned j = 0; j < m_joueurs.size(); ++j)
  {
    Joueur &joueur = *m_joueurs[j];
    cout<<"Joueur "
        <<joueur.getNom()
        <<" ("<<(j+1)<<") ==> "
        <<(!Pendaison::estPendu(joueur.getEtape())
             ? "EN VIE"
             : "PENDU...")
        <<" (points: "
        <<joueur.getPoints()
        <<")"<<endl;
  }
  cout<<endl;
}

/**
  Boucle principale d'une partie
*/
void JeuPendu::jouer()
{
  // Reinitialise les status des joueurs
  for (unsigned k = 0; k < m_joueurs.size(); ++k)
  {
    m_joueurs[k]->reset();
  }

  // Fait jouer chacun des joueurs à tour de rôle
  for (unsigned k = 0; k < m_joueurs.size(); ++k)
  {
    // Initialise le mot à deviner par le joueur k
    initADeviner(k);

    // Initialise la tentative
    initTentative();

    // Les joueurs autres que k sont sollicites pour
    // proposer des lettres
    bool b = devinerMot(k);
    if (not b)
    {
      cout<<"ADeviner etait: "<<m_adeviner<<endl;
    }
  }
}

/**
  Teste si un caractère est present dans le mot à deviner,
```

```
  et s'il n'a pas déjà ete propose
*/
bool JeuPendu::estDansMot(char c) const
{
  return (m_adeviner.find(c) != string::npos)
      and (m_tentative.find(c) == string::npos);
}

/**
  Inclut la lettre dans la tentative
*/
void JeuPendu::inclureDansMot(char c)
{
  for (unsigned j = 0; j < m_tentative.size(); ++j)
  {
    if (c == m_adeviner[j])
    {
      m_tentative[j] = c;
    }
  }
}

/**
  Permet au joueur d'initialiser le mot à deviner
*/
void JeuPendu::initADeviner(unsigned k)
{
  cout<<endl<<"Au joueur "
      <<m_joueurs[k]->getNom()
      <<" ("
      <<(k+1)
      <<") de proposer un mot:"<<endl;
  m_adeviner = m_joueurs[k]->proposerMot();
}

/**
  Initialise l'attribut <tentative> avec des '_'
*/
void JeuPendu::initTentative()
{
  m_tentative = string(m_adeviner.size(), '_');
}

/**
  Permet aux joueurs (a l'exception du joueur k) devant deviner
  le mot (propose par le joueur k) de faire leurs propositions
  a tour de role.
  @return Vrai si le mot a ete devine, Faux sinon
  (ce qui permet d'afficher ce qu'il fallait trouver).
  STRATEGIE Si la lettre proposee par un joueur est dans le
  mot, la tentative est affichee en la completant avec la
  lettre aux endroits appropries. Sinon l'affichage du pendu
  progresse d'une etape.
*/
bool JeuPendu::devinerMot(unsigned k)
{
  // Tantqu'un joueur a reussi a jouer et que le mot n'est
  // pas devine: on continue de jouer
```

```
for(;;)
{
  bool ajouer = false;
  for (unsigned j = 0; j < m_joueurs.size(); ++j)
  {
    // Le joueur k a propose le mot: on le saute
    if (j == k)
    {
      continue;
    }

    // Reference sur le joueur j
    Joueur &joueur = *m_joueurs[j];

    // Si le joueur j ne peut pas jouer: passe au suivant
    if (Pendaison::estPendu(joueur.getEtape()))
    {
      continue;
    }

    // Ici au moins un joueur a reussi a jouer
    ajouer = true;

    // Affiche la tentative actuelle, le status du joueur j,
    // et demande sa lettre proposee
    cout<<endl<<"TENTATIVE: "<<m_tentative<<endl;
    cout<<"  Joueur "<<joueur.getNom()<<" ("<<(j+1)<<") lettre? ";
    char lettre = joueur.proposerLettre();

    // Actualise le score du joueur
    if (estDansMot(lettre))
    {
      inclureDansMot(lettre);
      cout<<"Bravo: "<<m_tentative<<endl;
      if (m_tentative == m_adeviner)
      {
        joueur.incrPoints();
        cout<<"==> GAGNE !"<<endl;
        return true;
      }
    }
    else
    {
      joueur.incrEtape();
      cout<<"Pas de chance:"<<endl;
      Pendaison::afficher(joueur.getEtape());
      if (Pendaison::estPendu(joueur.getEtape()))
      {
        cout<<"==> PERDU !"<<endl;
        m_joueurs[k]->incrPoints();
        return false;
      }
    }
  }

  // Si aucun des joueurs ne peut plus jouer: le joueur k
  // a gagne tous les points: on sort
  if (not ajouer)
```

```
    {
      return false;
    }
  }
}
```

2.3 Classe Humain

Ce problème définit la classe du joueur humain et réalise le jeu du pendu de deux joueurs humains.

Créez une classe (concrète) `Humain` dérivée de la classe `Joueur` qui permet de modéliser les joueurs humains.

Écrivez la méthode abstraite `proposerMot` qui se contentera de lire une chaîne depuis le terminal et de la renvoyer.

Écrivez la méthode abstraite `proposerLettre` qui se contentera de lire un caractère depuis le terminal et de le renvoyer.

Validez votre classe et vos méthodes avec la solution.

Solution C++ @[Humain.hpp] @[Humain.cpp]

```
#ifndef JPHUMAIN_CLASS
#define JPHUMAIN_CLASS
#include <iostream>
#include <string>
using namespace std;
#include "Joueur.hpp"

/**
  Represente des joueurs "Humain"
*/
class Humain : public Joueur
{
  typedef Joueur super;

  public:
    Humain(const string& n);
    virtual string proposerMot() const;
    virtual char proposerLettre() const;
};
#include "Humain.cpp"
#endif
```

```
/**
  Constructeur normal
*/
```

```
Humain::Humain(const string& n)
: super(n)
{}

/**
  Propose un mot a faire deviner
*/
string Humain::proposerMot() const
{
  cout<<"(Les autres joueurs ne regardent pas)"<<endl;
  cout<<"Mot propose? ";
  string s;
  cin>>s;
  return s;
}

/**
  Propose une lettre
*/
char Humain::proposerLettre() const
{
  char c;
  cin>>c;
  return c;
}
```

Écrivez une procédure `jeu_contreHumain` qui permet à deux joueurs humains de s'affronter.

Testez.

Validez votre procédure avec la solution.

Solution C++ @[pgjpendu.cpp]

```
void jeu_contreHumain()
{
  // Instancie les joueurs
  Humain j1("Humain1");
  Humain j2("Humain2");

  vector<Joueur*> joueurs;
  joueurs.push_back(&j1);
  joueurs.push_back(&j2);

  // Instancie un jeu du Pendu
  JeuPendu jeu(joueurs);
  // Effectue les parties
  cout<<"Une autre partie (o/n)? ";
  char c;
  cin>>c;
  while (tolower(c) == 'o')
  {
    jeu.jouer();
```

```
    jeu.afficher();
    cout<<"Une autre partie (o/n)? ";
    cin>>c;
  }
}
```

2.4 Classe Machine

On souhaite maintenant faire en sorte qu'un des joueurs puisse être l'ordinateur. Si c'est le cas, les méthodes abstraites de la classe `Joueur` ne se contenteront plus de lire le mot ou la lettre mais les génèreront automatiquement. Afin de générer des mots valides, un joueur machine dispose d'un dictionnaire de mots (tableau de chaînes de caractères) qui sera chargé au lancement du programme et qui sera fourni au constructeur.

Créez une classe (concrète) `Machine` dérivée de classe `Joueur` qui contient un vecteur de mots (chaîne de caractères).

Fournissez un constructeur permettant de mémoriser le dictionnaire.

Écrivez la méthode abstraite `proposerMot` de la façon suivante :

1. Générez aléatoirement un entier $u \in [1..n]$ (où n désigne la taille du dictionnaire).
2. Retournez le u-ème mot du dictionnaire.

Outil C++

La fonction `rand()`, définie dans la bibliothèque `<random>`, renvoie un entier pseudo-aléatoire positif ou nul. Utilisez le modulo pour projeter l'entier dans l'intervalle souhaité.

Outil C++

L'initialisation du générateur de nombres pseudo-aléatoires (ici avec l'horloge système) s'effectuera dans le **programme principal** avec l'instruction :

```
srand(time(NULL));
```

La procédure `srand` est définie dans la bibliothèque `<random>` et la fonction `time` dans la bibliothèque `<ctime>`.

Écrivez la méthode pure `proposerLettre` de la façon suivante :

1. Générez aléatoirement un entier $p \in [0..25]$.
2. Retournez la p-ème lettre correspondante.

Outil C++

La p-ème lettre minuscule est donné par l'expression :

```
char(int('a')+p)
```

Validez votre classe et vos méthodes avec la solution.

Solution C++ @[Machine.hpp] @[Machine.cpp]

```
#ifndef JPMACHINE_CLASS
#define JPMACHINE_CLASS
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <string>
#include <vector>
using namespace std;

/**
  Represente des joueurs "Ordinateur"
*/
#include "Joueur.hpp"
class Machine : public Joueur
{
  typedef Joueur super;

  public:
    Machine(const string& n, const vector<string>& d);
    virtual string proposerMot() const;
    virtual char proposerLettre() const;

  private:
    const vector<string> &m_dict; // dictionnaire
};
#include "Machine.cpp"
#endif
```

```
/**
  Constructeur normal
  Initialise le generateur de nombres aleatoires
*/
Machine::Machine(const string& n, const vector<string>& d)
: super(n), m_dict(d)
{
  srand(time(0));
}

/**
  Propose un mot a faire deviner: Tire un nombre de la
  taille du dictionnnaire et retourne le mot correspondant
*/
string Machine::proposerMot() const
{
  string s = m_dict[ rand() % m_dict.size() ];
  cout<<"Mot propose "<<s<<endl;
  return s;
}

/**
  Propose une lettre.
  Genère aleatoirement une lettre de l'alphabet.
*/
```

```
char Machine::proposerLettre() const
{
  // On pourrait pu memoriser un tableau de booleens
  // des lettres deja proposees afin d'eviter de proposer
  // plusieurs fois la meme lettre... on admettra que le
  // generateur est un "bon" generateur uniforme
  char c = char(int('a') + rand() % 26);
  cout<<c<<endl;
  return c;
}
```

Écrivez une procédure `jeu_contreMachine` qui permet à un joueur humain d'affronter un joueur machine.

Testez.

Validez votre procédure avec la solution.

Solution C++ @[pgjpendu.cpp]

```
void jeu_contreMachine()
{
  // Tente l'ouverture du dictionnaire de mots
  ifstream is;
  if ( not UtilsFH::demanderNom(is) )
  {
    cerr<<"=> j'abandonne..."<<endl;
    return;
  }

  // Charge le dictionnaire
  istream_iterator<string> beg(is);
  istream_iterator<string> eos;
  vector<string> dico(beg, eos);
  is.close();

  // Affiche le dictionnaire
  cout<<"Taille du Dicitionnaire = "
      <<dico.size()
      <<endl;
  copy(dico.begin(),
       dico.end(),
       ostream_iterator<string>(cout, " "));
  cout<<endl<<endl;

  // Instancie les joueurs
  Humain j1("Humain");
  Machine j2("PC", dico);

  vector<Joueur*> joueurs;
  joueurs.push_back(&j1);
  joueurs.push_back(&j2);
```

```
  // Instancie un jeu du pendu
  JeuPendu jeu(joueurs);
  // Effectue les parties
  cout<<"Une autre partie (o/n)? ";
  char c;
  cin>>c;
  while (tolower(c) == 'o')
  {
    jeu.jouer();
    jeu.afficher();
  }
}
```

3 Références générales

Comprend [Chappelier-CPP1 :c7 :ex45] ■