TP5: design de code orienté objet en pratique
C++

Objectifs pédagogiques :
- Comprendre la différence entre la philosophie C et celle C++ en matière d'organisation du code.
- Découper un problème en classes en suivant l'approche CRC.
- Utiliser UML pour représenter le design d'un code orienté objet.
- Implémenter un code orienté objet.
- Utiliser les différents mécanismes du C++ dans son code.

★ Exercice 1: Jeu de UNO.

Vous trouverez sur l'autre feuille le code du jeu de UNO écrit en C++ [1], mais en suivant la philosophie du C. Ce code, relativement bien documenté, tire partie des conteneurs et algorithmes standards du C++, mais aucune classe d'objet n'est définie, il n'y a pratiquement pas de fonction pour découper et abstraire le code. L'objectif de cet exercice est de le refactorer pour le rendre plus lisible.

Exemple de partie

```
Joueur 1, à vous.
Carte sur la défausse: 7 bleu;
Votre jeu: a:3 rouge | b:4 rouge | c:7 vert | d:9 vert | e:change de sens vert | f:2 jaune | g:Joker |
Quelle carte jouer ('z' pour tirer une nouvelle carte)? g


Joueur 2, à vous.
Carte sur la défausse: Joker;
Votre jeu: a:+2 rouge | b:passe ton tour vert | c:3 jaune | d:2 jaune | e:+2 jaune | f:0 bleu | g:7 bleu |
Quelle carte jouer ('z' pour tirer une nouvelle carte)? a


2 cartes de pénalité pour le joueur 1!!
Carte sur la défausse: +2 rouge;
Votre jeu: a:3 rouge | b:4 rouge | c:9 rouge | d:7 vert | e:9 vert | f:change de sens vert | g:2 jaune | h:5 jaune |
Quelle carte jouer ('z' pour tirer une nouvelle carte)? ^C
```

▷ **Question 1:** Modifiez ce code pour que la main du joueur soit triée lorsque vient son tour.

▷ **Question 2:** Proposez une fonction pour séparer et abstraire proprement les lignes 57, 58 et 59. Proposez un nom, donnez son implémentation et modifiez la fonction `main()` pour utiliser cette nouvelle fonction.

▷ **Question 3:** Proposez un découpage en classes de ce problème. Vous donnerez les responsabilités de chaque classe (au sens CRC) éventuellement avec les lignes du code existant correspondant à chaque responsabilité, avant de dessiner le diagramme UML liant les classes. Justifiez ce qui doit l'être.

▷ **Question 4:** Modifiez le code pour suivre le découpage que vous avez proposé. Si vous avez correctement découpé le projet, la boucle principale ne devrait pas dépasser une quinzaine de lignes.

★ Exercice 2: Échelles et serpents.

Le jeu traditionnel indien *échelles et serpents* ressemble un peu au jeu de l'oie. Chaque joueur à son tour lance un dé, puis avance son pion du nombre de cases indiqué. Les cases *échelle* font avancer le pion vers la case indiquée, tandis que les cases *serpent* font reculer le joueur. Le premier joueur à arriver sur la case 100 gagne. C'est donc un pur jeu de chance.

Le code `snake_ladder.cpp` donné en annexe est une implémentation presque complète de ce jeu. Seule la fonction `waitkey()` (qui attend qu'une touche soit pressée) est omise, ainsi que les entêtes. Ce code est fourni dans l'archive associée à ce TP.

▷ **Question 1:** (prise en main) Expliquez les lignes 27 à 32.

La qualité du code proposé pose problème, tant au niveau des traitements effectués qu'au niveau des données manipulées. L'objectif de cet exercice est de proposer des pistes d'amélioration.

▷ **Question 2:** Proposez un redécoupage de ce code en plusieurs fonctions. Vous devriez pouvoir reprendre lee Indiquez le prototype de chaque fonction ainsi que les lignes du code original correspondantes, mais il n'est pas nécessaire d'écrire le corps de chaque fonction. Expliquez simplement les éventuelles modifications à apporter au code repris.

▷ **Question 3:** Modifiez maintenant le programme pour permettre à un nombre variable de joueurs de jouer. Pour plus de convivialité, on demandera le nom de chaque joueur en début de partie. On souhaite également enregistrer les scores d'une partie sur l'autre pour organiser des championnats en plusieurs manches.

1. Les règles sont légèrement simplifiées par rapport au vrai jeu de UNO, mais l'esprit demeure.

▷ **Question 4:** On souhaite maintenant ajouter une nouvelle interface graphique basée sur SFML plus de l'existante. Plus précisément, on souhaite faire un moteur de jeu générique qui puisse être utilisé avec l'une ou l'autre des interfaces. Proposez un découpage en plusieurs classes, en listant les responsabilités et collaborateurs de chaque classe. Vous n'avez pas à écrire de code à cette question.

▷ **Question 5:** (optionnelle) Implémentez l'interface graphique en question.

★ Exercice 3: Enveloppe convexe (optionnel)

Vous trouverez sur l'autre feuille le code d'un calcul de l'enveloppe convexe d'un ensemble de points en utilisant l'algorithme de Graham. Quand on exécute ce code, puis qu'on utilise le code d'affichage en bas de la page (qui utilise SFML), on obtient la figure représentée à droite où les coordonnées de chaque point sont écrites à sa position, et où l'enveloppe convexe est représentée graphiquement.

Ce code utilise quelques calculs mathématiques. Aux lignes 22, 38 et 58, on calcule le module du vecteur étant le produit vectoriel de deux vecteurs $\|\overrightarrow{v1} \wedge \overrightarrow{v2}\| = v1.x \times v2.y - v1.y \times v2.x$. Si cette grandeur est positive, alors $\overrightarrow{v1}$ et $\overrightarrow{v2}$ sont dans le sens direct ; si elle est nulle, les deux vecteurs sont colinéaires ; dans le dernier cas, $\overrightarrow{v1}$ et $\overrightarrow{v2}$ sont dans le sens contraire ($\overrightarrow{v2}$ et $\overrightarrow{v1}$ sont donc dans le sens direct). Aux lignes 25, 26 et 27, on trie les points selon la distance euclidienne sans calculer de racine carrée. `distj0` est le carré de la distance entre le point 0 et le point j, tandis que `distj1` est celui de la distance entre le point 0 et le point $j+1$.

Ce code compile en C++, mais il caricature la philosophie du C. Il n'y a aucune classe, les conteneurs et algorithmes de la bibliothèque standard du C++ ne sont pas utilisés, et il n'y a même pas de fonctions déclarées. L'objectif de cet exercice est de refactorer le code du calcul pour le rendre plus lisible. Le code d'affichage n'est fourni que pour référence et pour les tests.

▷ **Question 1:** Nommez l'algorithme de tri utilisé des lignes 18 à 33.

▷ **Question 2:** Écrivez une classe `Point` permettant de représenter un point du plan, sous forme de deux doubles x et y. Vous doterez cette classe d'un constructeur prenant une `std::string` comme paramètre et desserialisera les coordonnées du point de la chaîne. Vous pouvez supposer que la chaîne à désérialiser est toujours valide, sans jamais comporter d'erreur de syntaxe, ou vous pouvez optionnellement rendre votre parseur plus robuste.

▷ **Question 3:** Écrivez le code nécessaire pour que les lignes numérotées 64-68 ci-dessous fassent la même chose que les lignes 12 à 16 du code fourni.

```
57 std::vector<Point> p;
58 std::ifstream input("hull-points.txt");
59
60 std::string text_line;
61 while (std::getline(input, text_line))
62    p.push_back(Point(text_line));
63
64 // Trouve le pivot = point de coordonnées minimales
65 for (int i=1; i<p.size(); i++)
66   if (p[i] < p[0]) {
67     Point t = p[0];   p[0] = p[i];   p[i] = t;
68   }
```

On suppose donnée une classe `Vecteur` reliant deux points et dotée d'un opérateur `int Vecteur::operator*()` calculant le produit scalaire ainsi que d'un opérateur `bool Vecteur::operator<(Vecteur o)` retournant vrai si la norme de l'objet courant est inférieur à celle de l'objet `o`.

▷ **Question 4:** Écrivez un équivalent des lignes 18 à 33 du code fourni permettant de trier la collection de points par angle polaire ou par distance en cas d'égalité. Vous vous attacherez à utiliser les mécanismes offerts par la STL.

▷ **Question 5** (bonus)**:** Écrivez un filtrage basé sur des ranges C++20 pour remplacer les lignes 35 à 47 du code fourni où l'on supprime les points successifs colinéaires de la collection.

▷ **Question 6:** Écrivez une classe templatée `Pile` permettant de retrouver le sommet de la pile ainsi que l'élément sous le sommet sans modifier la pile. Votre implémentation peut utiliser la classe `std::vector` pour stocker les données. Il doit s'agir d'une template permettant de stocker n'importe quel type d'éléments.

Votre classe doit implémenter les opérations suivantes : `push` (empile), `pop` (dépile), `top` (renvoyant l'élément au sommet), `prev` (renvoyant l'élément juste avant top), `size` (renvoyant la taille) et l'opérateur d'indexation `operator[]` (renvoyant l'élément à l'index indiqué).

▷ **Question 7:** Réécrivez l'algorithme de Graham avec les éléments ainsi constitués, en vous inspirant autant que nécessaire du code fourni.

Jeu de UNO en pseudo-C

```
std::string to_str(int card) { // Représentation textuelle d'une carte identifiée par son num.
  std::string couleur[] {"rouge", "vert", "jaune", "bleu"};
  if (card <= 103) {
    int v = card % 26;
    if (v > 12)
      v -= 13; // Toutes les cartes sont en double
    if (v < 10)
      return std::to_string(v % 10) + " " + couleur[card / 26];
    if (v == 10)
      return "+2 " + couleur[card / 26];
    if (v == 11)
      return "change de sens " + couleur[card / 26];
    if (v == 12)
      return "passe ton tour " + couleur[card / 26];
  } else if (card <= 107)
    return "Joker";
  return "Super joker +4";
}
int main() {
  srand(time(nullptr)); // Initialize the pseudo-random
  std::vector<int> deck, defausse;
  for (int i = 0; i < 112; i++)
    deck.push_back(i);
  std::random_shuffle(deck.begin(), deck.end()); // permutation aléatoire
  std::vector<int> hand, other;  // jeu de chaque joueur (hand: joueur actuel; other: l'autre)
  for (int i = 0; i < 7; i++) {
    hand.push_back(deck.back());     deck.pop_back();
    other.push_back(deck.back());    deck.pop_back();
  }
  defausse.push_back(deck.back());   deck.pop_back(); // Place une carte dans la défausse
  int player = 1;
  while (true) { // La partie continue
    int idx;
    int penalite = (defausse.back() % 26 == 10 || defausse.back() % 26 == 23) ? 2
                 : ((defausse.back() > 107) ? 4 : 0);
    if (penalite > 0) {
      std::cout << "\n\n" << penalite << " cartes de pénalité pour le joueur "<<player<<"!!\n";
      for (int i = 0; i < penalite; i++) {
        hand.push_back(deck.back());
        deck.pop_back();
    } } else std::cout << "\n\nJoueur " << player << ", à vous.\n";
    while (true) { // Attend une carte valide
      std::cout << "Carte sur la défausse: " << to_str(defausse.back()) << ";\nVotre jeu: ";
      for (unsigned i = 0; i < hand.size(); i++)
        std::cout << static_cast<char>('a' + i) << ":" << to_str(hand[i]) << " | ";
      std::cout << "\nQuelle carte jouer ('z' pour tirer une nouvelle carte)? ";
      char choix;
      std::cin >> choix;
      if (choix == 'z' || (choix >= 'a' && choix - 'a' < hand.size())) {
        idx = (choix == 'z') ? 0 : choix - 'a' + 1;
        if (idx == 0 // le joueur tire une carte
            || defausse.back() > 103 || hand[idx - 1] > 103 // l'une est joker
            || defausse.back() / 26 == hand[idx - 1] / 26 // même couleur
            || (defausse.back() % 26)%13 ==  (hand[idx - 1] % 26)%13) // même valeur
          break; // choix validé
    } }
    if (idx == 0) {
      hand.push_back(deck.back());      deck.pop_back();
    } else {
      defausse.push_back(hand[idx - 1]);
      hand.erase(hand.begin() + idx - 1); // Retire hand[idx] du vecteur hand
    }
    if (hand.empty()) {
      std::cout << "LE JOUEUR " << player << " A GAGNÉ!!\n";
      exit(0);
    }
    if (deck.empty()) {
      std::swap(deck, defausse); // inverse les deux piles
      std::random_shuffle(deck.begin(), deck.end()); // mélange
      defausse.push_back(deck.back());       deck.pop_back();
    }
    std::vector<int> pt{11, 12, 24, 25}; // Cartes qui passent son tour, dans chaque couleur
    if (std::none_of(pt.begin(), pt.end(),
                     [&defausse](auto i) { return defausse.back() % 26 == i; })) {
      player = (player == 1 ? 2 : 1);
      std::swap(hand, other);
} } }
```

Échelle et serpents en pseudo-C

```
int main() {
  int player1 = 1, player2 = 1;
  srand(time(nullptr)); // Initialize the pseudo-random

  // Some cells teleport the player to another cell.
  // mover[i]=j means that when you arrive to the cell i, you're teleported to cell j
  std::vector<int> mover;
  for (int i = 0; i <= 106; i++)
    mover.push_back(i);
  for (auto elem : std::vector<std::vector<int>> {
      {98,28}, {95,24}, {92,51}, {83,19}, {69,33}, {64,36}, {48,9},{8,26},{55,7},{73,1},{46,5},
      {21,82}, {43,77}, {50,91}, {54,93}, {62,96}, {66,87},{44,22},{52,11}, {59,17},{80,100} })

    mover[elem[0]] = mover[elem[1]];

  std::cout << "============================================================\n\n";
  std::cout << "\t\tSNAKE LADDER GAME\n\n";
  std::cout << "============================================================\n\n";

  int turn = 1;
  while (player1 < 100 && player2 < 100) {

    std::cout << "-----------------[ turn " << turn++ << " ]-------------------------------\n";
    for (int cell = 1; cell <= 100; cell++) {
      std::string content;
      if (player1 == cell)
        content = "*P1* ";
      if (player2 == cell)
        content += "*P2* ";
      if (mover[cell] > cell)
        content += "ladder to " + std::to_string(mover[cell]);
      if (mover[cell] < cell)
        content += "snake to " + std::to_string(mover[cell]);
      std::cout << std::setw(3) << cell << "[" << std::setw(13) << content << "]";
      if (cell % 10 == 0) // Print 10 cells on each line
        std::cout << std::endl;
    }
    std::cout << "------------------------------------------------------------\n";
    std::cout << "\n---> Player 1: press a key to roll the dice... ";
    waitkey();
    int lastposition = player1;
    int dice         = random() % 6 + 1;
    player1          = mover[player1 + dice];
    std::cout << "You rolled a " << dice << "!! Now you are at position " << player1;
    if (player1 < lastposition)
      std::cout << ". Oops!! Snake found !!";
    else if (player1 > lastposition + 6)
      std::cout << ". Great!! you got a ladder !!";

    std::cout << "\n\n---> Player 2: press a key to roll the dice... ";
    waitkey();
    lastposition = player2;
    dice         = random() % 6 + 1;
    player2      = mover[player2 + dice];
    std::cout << "You rolled a " << dice << "!! Now you are at position " << player2;
    if (player2 < lastposition)
      std::cout << ". Oops!! Snake found !!";
    else if (player2 > lastposition + 6)
      std::cout << ". Great!! you got a ladder !!";
    std::cout << "\n\n(press a key to start the next turn)\n\n";
    waitkey();
  }
  std::cout << "\n\n++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++\n";
  if (player1 >= player2)
    std::cout << "\t\tPlayer 1 wins the game\n";
  else
    std::cout << "\t\tPlayer 2 wins the game\n";
  std::cout << "++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++\n\n";
  return 0;
}
```

Calcul de l'enveloppe convexe en pseudo-C (à refactorer)

```
int main() {
  int x[] = { 7,-5,2,6,8, 7, 4, 8,0, 3, 6, 0,-8,-8,-8,-9,-2 };
  int y[] = { 8, 6,9,4,6,-2,-6,-7,0,-2,-9,-9,-9,-2, 0, 3, 2 };
  int length = sizeof(x)/sizeof(x[0]);

  // Trouve le pivot = point de coordonnées minimales
  for (int i=1; i<length; i++)
    if (y[i] < y[0] || (y[i] == y[0] && x[i]<x[0])) {
      int tx = x[0];    x[0] = x[i];   x[i] = tx;
      int ty = y[0];    y[0] = y[i];   y[i] = ty;
    }

  // Tri les points par angle polaire (ou par distance si égalité)
  int sorted = 0;
  while (!sorted) {
    sorted = 1;
    for (int j=1; j<length-1;j++) {
      int prod_vect = (x[j]-x[0]) * (y[j+1]-y[0]) - (y[j]-y[0]) * (x[j+1]-x[0]);

      int distj0 = (x[j+0]-x[0])*(x[j+0]-x[0]) + (y[j+0]-y[0])*(y[j+0]-y[0]);
      int distj1 = (x[j+1]-x[0])*(x[j+1]-x[0]) + (y[j+1]-y[0])*(y[j+1]-y[0]);
      if (prod_vect < 0 || (prod_vect == 0 && distj1<distj0)) {
        sorted = 0;
        int tx = x[j];    x[j] = x[j+1];   x[j+1] = tx;
        int ty = y[j];    y[j] = y[j+1];   y[j+1] = ty;
      }
    }
  }

  // Supprime les points co-linéaires par rapport au pivot et leur voisin
  int modified_size=1;
  for (int i=1; i<length; i++) {
    while (i < length-1 && (x[i]-x[0]) * (y[i+1]-y[0]) - (y[i]-y[0]) * (x[i+1]-x[0]) == 0)
      i++;

    if (i!=modified_size) {
      x[modified_size] = x[i];
      y[modified_size] = y[i];
    }
    modified_size++;
  }
  length = modified_size;

  // Initialise l'enveloppe convexe avec les 3 premiers points
  int x2[length] = {x[0], x[1], x[2]};
  int y2[length] = {y[0], y[1], y[2]};
  int top = 2;

  // Ajoute les points de la collection un à un
  for (int i = 3; i < modified_size; i++) {
    // Tant que le point au sommet ne tourne pas à gauche, on dépile
    while (top>1 &&
           (x2[top]-x2[top-1])*(y[i]-y2[top-1]) - (y2[top]-y2[top-1])*(x[i]-x2[top-1]) < 0)
      top --;

    // On ajoute un nouveau point à considérer
    top++;
    x2[top] = x[i];
    y2[top] = y[i];
  }
```

Code d'affichage (pour info)

```
// Chaque point
for (int i=0; i<length; i++) {
  text.setString(std::to_string(i)+" ("
       +std::to_string(x[i])+";"+std::to_string(y[i])+")");
  text.setPosition(x[i]*30+300, y[i]*-30+300);
  window.draw(text);
}
// Enveloppe
for (int i=0; i<top; i++) {
  line[0] = sf::Vector2f(x2[i]*30+300,y2[i]*-30+300);
  line[1] = sf::Vector2f(x2[i+1]*30+300,y2[i+1]*-30+300);
  line[0].color = line[1].color = sf::Color::Red;
  window.draw(line, 2, sf::Lines);
}// Ferme l'enveloppe
line[0] = sf::Vector2f(x2[top]*30+300,y2[top]*-30+300);
line[1] = sf::Vector2f(x2[0]*30+300,y2[0]*-30+300);
line[0].color = line[1].color = sf::Color::Red;
window.draw(line,2, sf::Lines);
```