

DVD Miage: Module Programmation Objet

Chapitre 10 - Éléments de Conception à objets

Un exemple illustré en Smalltalk

Pascal André, Gilles Ardourel



1 Introduction

Dans cette fiche nous montrons comment coder un programme en Smalltalk avec VisualWorks. L'exemple support est la gestion de compétitions de Tennis.

Plus généralement, cette fiche peut aussi servir d'introduction à la programmation à objets. Le cas d'étude est la gestion d'un match de tennis. Cet exemple est inspiré de de [?]. Nous partons d'un algorithme de programmation structurée pour aboutir à une programme à objets. Nous mettons en évidence un certain nombre de problèmes d'évolution du programme. Nous proposons ensuite une modélisation et une programmation à objet.

2 Enoncé

On souhaite écrire un programme qui simule une partie de tennis entre deux joueurs, selon les règles classiques du tennis (inspiré librement de [?]). Un match se joue en 3 sets gagnants (soit au maximum 3 ou 5 jeux).

Un joueur gagne un set s'il a au moins 6 jeux gagnés et deux jeux d'écart avec son adversaire. Si le score est de 6 jeux à 5 alors plusieurs cas sont possibles : soit le premier joueur gagne le jeu et il remporte le set 7-5, soit le second joueur gagne le jeu et un *tie-break* est joué, le gagnant du *tie-break* remporte le set par le score de 7 à 6.

Chaque jeu se joue en 4 points maximum pour le gagnant et deux points d'écart avec son adversaire. les points sont notés 0, 15, 30, 40, jeu. En cas d'égalité 40-40, les points sont notés : égalité, avantage au serveur ou avantage au receveur.

Le *tie-break* se joue en 7 points minimum pour le gagnant et deux points d'écart avec son adversaire (par exemple, 7-4, 9-7...).

C'est le même joueur qui sert pour tout un set, hormis le *tie-break* pour lequel, le serveur du set sert une fois puis les deux joueurs servent alternativement. Le joueur qui sert en premier est choisi au hasard. On ne tiendra pas compte des fautes ou des deux balles de service. Pour chaque balle engagée, le point sera simplement gagnant ou perdant.

On ne tient pas compte pour l'instant des impondérables : blessures des joueurs, intempéries, abandons.

a) Ecrire un programme impératif qui modélise un match de tennis. On pourra utiliser des tableaux et une numérotation numérique des joueurs. On pourra aussi coder le codage des points pour faciliter le calcul des scores.

b) On souhaite pouvoir adapter le programme aux évolutions suivantes.

1. Dans un tournoi certains matchs se jouent en deux sets gagnants (début du tournoi, tournoi féminin...).
2. On souhaite inclure ce programme dans le cadre d'un tournoi complet et conserver tous les scores jusqu'à la finale.
3. On souhaite modifier le programme pour des matchs de doubles.
4. On souhaite écrire un programme similaire pour un tournoi de tennis de table.
5. Un joueur peut abandonner la partie.

Commenter l'adaptabilité du programme aux modifications souhaitées.

c) Proposer une conception abstraite à objets. On proposera par exemple une modélisation avec la notation UML.

d) Reprendre la question b) avec la conception à objets.

e) Implanter la conception à objets en Smalltalk. Discuter des alternatives de codage.

3 Programmation structurée

3.1 Programme

Ecrire un programme impératif qui modélise un match de tennis. On pourra utiliser des tableaux et une numérotation numérique des joueurs. On pourra aussi coder le codage des points pour faciliter le calcul des scores.

Voici le programme en langage impératif type Pascal correspondant.

```
program tennis;
  (* auteur : P. André, février 2001
   programme inspiré de Nerson *)
  (* Les joueurs sont représentés par des index dans des tableaux.
   Le changement de service est simplement une opération arithmétique.
   Les comptages de points sont des entiers, une table de conversion
   fait éventuellement la conversion pour l'affichage.
   Toutes les limites sont définies par des constantes.
   Le vainqueur d'un échange est calculé aléatoirement pour éviter
   des saisies rébarbatives.
   La partie est naturellement découpée en procédures :
   match, set, jeu, tie-break.
  *)
```

```
const
  nb_joueurs = 2;
  nb_sets = 5;
  nb_sets_g = 3;
  nb_jeux = 7;
  nb_pts_jeu = 4;
  nb_pts_tie = 7;
  points = array[1..nb_pts_jeu] of integer = {0, 15, 30, 40};
  pointsup = array[1..2] of string = {'égalité', 'avantage'};
```

```
type
  score_jeux = 0..nb_jeux;
  type_jeu = {norm, tie};
```

```
var
  joueurs : array[1..nb_joueurs] of string;
  match : array[1..nb_sets, 1..nb_joueurs] of score_jeux;
  serveur : 1..nb_joueurs;
  set_gagnes : array[1..nb_joueurs] of integer;
  set_en_cours : 1..nb_sets;
  score_set : array[1..nb_joueurs] of score_jeux;
  nom : string;
  i, j : integer;
```

```
procedure afficher_points_jeu (pt : integer, jeu : type_jeu);
begin
  case jeu of
    norm : write (points[pt]);
    tie : write (pt);
  end; (* case *)
end; (* afficher_points_jeu *)
```

```
procedure afficher_msg_jeu (jeu : type_jeu);
begin
  case jeu of
    norm : write ('jeu ');
    tie : write ('tie break ');
  end; (* case *)
end; (* afficher_points_jeu *)
```

```
procedure jouer_jeu (nb_pts_max : integer,
  tj : type_jeu,
  var gagnant : 1..nb_joueurs);
```

```

(* Cette procédure sert à la fois aux jeux normaux et au
   tie-break, car les règles sont similaires.
   La distinction se fait par les variables nb_pts_max et tj *)
var
  jeu : array[1..nb_joueurs] of integer;
  fin_jeu : boolean;
  gagn, perd : 1..nb_joueurs;
  i : integer;

begin
for i := 1 to nb_joueurs do jeu[i] := 0;
fin_jeu := false;
repeat
  gagn := random(nb_joueurs); (* un point est joué *)
  perd := (gagn mod nb_joueurs) + 1;
  (* pas de sens si nb_joueurs < > 2 *)
  jeu[gagn] := jeu[gagn] + 1;
  if (jeu[gagn] > nb_pts_max) then begin
    case (jeu[gagn] - jeu[perd]) of
      0 : writeln (pointsup[1]);
      1 : writeln (pointsup[2], joueurs[gagn]);
      2 : fin_jeu := true
    end; (* case *)
  end else begin
    write (joueurs[gagn], ' ');
    afficher_points_jeu (jeu[gagn], tj);
    write (' - ');
    afficher_points_jeu (jeu[perd], tj);
    writeln (joueurs[perd])
  end (* if *)
until fin_jeu ;
afficher_msg_jeu (tj);
writeln (joueurs[gagn]);
gagnant := gagn
end; (* jouer_jeu *)

procedure jouer_set (var serveur : 1..nb_joueurs,
  var set : array[1..nb_joueurs] of score_jeux;
  var gagnant : 1..nb_joueurs);

var
  fin_set : boolean;
  gagn : 1..nb_joueurs;
  i: integer;

begin
for i := 1 to nb_joueurs do set[i] := 0;
fin_set := false;
repeat
  jouer_jeu (nb_pts_jeu, norm, gagn); (* un jeu est joué *)
  perd := (gagn mod nb_joueurs) + 1;
  (* pas de sens si nb_joueurs < > 2 *)
  if (perd = serveur) then
    writeln ('break pour ',joueurs[gagn]);
  (* end if *)
  set[gagn] := set[gagn] + 1;
  writeln (joueurs[serveur], ': ',set[serveur])
  write (' - ');
  serveur := (serveur mod nb_joueurs) + 1;
  (* pas de sens si nb_joueurs < > 2 *)
  writeln (joueurs[serveur], ': ',set[serveur])
  if (set[gagn] > nb_sets) then begin
    case (set[gagn] - set[perd]) of
      0 : begin
        writeln ('tie break');
        jouer_jeu (nb_pts_tie, tie, gagn);
        fin_set := true
        end; (* case 0 *)
      2 : fin_set := true
        end; (* case *)
    (* end if *)
  end
until fin_set ;
writeln ('set ',joueurs[gagn]);

```

```

gagnant := gagn
end; (* jouer_set *)

begin (* Début du programme *)
  for i := 1 to nb_joueurs do begin
    readln('nom du joueur : ', nom);
    joueurs[i] := nom;
    for j := 1 to nb_sets do match[j, i] := 0
  end; (* for *)
  serveur := random(nb_joueurs);
  set_en_cours := 0;
  repeat
    set_en_cours := set_en_cours + 1;
    jouer_set (serveur, score_set, gagnant);
    (* le changement de service y est effectué *)
    for i := 1 to nb_joueurs do
      match[set_en_cours, i] := score_set[i];
    end; (* end for *)
    set_gagnes[gagnant] := set_gagnes[gagnant] + 1;
  until (set_gagnes[gagnant] = nb_sets.g);
  writeln('Le joueur ', joueurs[gagnant], ' a gagné en ', set_en_cours, ' sets. ');
  writeln('Le score final est : ');
  for i := 1 to nb_joueurs do begin
    writeln(joueurs[i], ' : ');
    for j := 1 to set_en_cours do
      write (match[j, i], ' ');
    end;
    writeln(' ');
  end; (* for *)
end. (* Fin du programme *)

```

Figure 1 : Programme impératif du match de tennis

3.2 Evolution de l'énoncé

On souhaite pouvoir adapter le programme aux évolutions suivantes.

1. Un joueur peut abandonner la partie.
2. Dans un tournoi certains matchs se jouent en deux sets gagnants (début du tournoi, tournoi féminin...).
3. On souhaite inclure ce programme dans le cadre d'un tournoi complet et conserver tous les scores jusqu'à la finale.
4. On souhaite modifier le programme pour des matchs de doubles.
5. On souhaite écrire un programme similaire pour un tournoi de tennis de table.

Commenter l'adaptabilité du programme aux modifications souhaitées.

Nous avons utilisé un maximum de constantes pour paramétrer le programme. Une factorisation a été obtenue en groupant jeux et tie-break. Nous avons, autant que possible, rendu indépendantes les différentes procédures (cohérence, faible couplage). La structure du programme est relativement visible.

Discutons maintenant des différentes évolutions proposées.

1. Un joueur peut abandonner la partie. L'abandon est une fonction aléatoire à l'intérieur de la procédure `jouer_un_jeu`. On arrête un set s'il y a un abandon dans un jeu. On peut utiliser les variables `fin_jeu`, `fin_set` et `match.fin` pour arrêter un jeu ou un set en cours. On ajoute un paramètre dans chaque procédure qui indique que la partie est terminée pour propager la décision. A l'appel d'un nouveau set ou d'un nouveau jeu, on vérifie les points suivants. On joue un jeu s'il n'y a pas d'abandon dans le jeu précédent.

L'ensemble du programme est modifié même si les modifications sont mineures.

2. Matchs en deux sets gagnants.

Il suffit de changer la constante `nb_sets.g` en 2 au lieu de 3. On peut aussi affiner `nb_sets = 3`. On gagne ainsi de l'espace mémoire.

3. Tournoi complet.

Le programme principal et ses déclarations de variables sont rangés dans une procédure. Il faut ajouter de nouvelles structures de données pour stocker les résultats des différents matchs. On peut s'inspirer de l'analogie set-match pour concevoir le binôme match-tournoi. Ainsi, les résultats d'un match sont recopiés au niveau du tournoi. La structure des joueurs est à élever au niveau du tournoi. Il faut faire le lien entre les joueurs d'un match et ceux du tournoi.

4. Matchs en doubles.

Soit on considère l'équipe et non le joueur et il n'y a rien à changer. Soit il faut dissocier la notion de joueur (conservée pour le changement de service) de la notion d'équipe (mémoire des scores).

5. Tennis de tables.

On change les règles de calcul des points (constantes) et la règle de jeu d'un set. Le changement de service est aussi différent. Cette dernière modification implique des modifications assez subtiles dans l'ensemble du programme.

3.3 Enseignements

Le paramétrage du programme est essentiel à son évolution. Un soin particulier est à apporter à la lisibilité des actions.

Il est difficile de mesurer l'impact d'une modification. Il faut repérer données et instructions relatives à une même action. Selon les modifications, l'ensemble du programme peut être remis en cause.

La réutilisation de morceaux de code implique une forte abstraction lors de l'écriture.

4 Conception abstraite à objets

Proposer une conception abstraite à objets. On proposera par exemple une modélisation avec la notation UML.

La lecture de l'énoncé fait apparaître quatre abstractions distinctes : les joueurs, les matchs, les sets et les jeux. En fait, on remarque une imbrication des différents objets entre eux : les joueurs d'un set sont ceux du match du set.

- Joueur. Un joueur est simplement caractérisé par un nom, un prénom et son classement.
- Match. Un match est une partie entre deux joueurs. Le match est défini par un ensemble de sets (une liste si on souhaite mémoriser la progression). Le match est terminé dès qu'un joueur a remporté le nombre de sets gagnants.
- Un set une partie d'un match. Un set est défini par un ensemble de jeux (une liste si on souhaite mémoriser la progression). On souhaite ici ne mémoriser que le score de chaque jeu du set. Le set est terminé lorsqu'un joueur a remporté six jeux avec deux jeux d'écart ou un tie-break.
- Un jeu met en compétition deux joueurs, le serveur et le receveur. Le jeu se joue en 4 points minimum. Le vainqueur a au moins 4 points avec deux points d'écart. Les points sont notés selon la convention 0, 15, 30, 40 et la règle de l'égalité ou de l'avantage. Un jeu gagné par le receveur est un avantage psychologique, appelé *break*.

Nous proposons dans la figure 2 une conception abstraite du problème. Nous avons adopté une notation à la Smalltalk des variables et associations pour faciliter l'implantation. La navigation des associations indique une orientation non symétrique des liens entre objets. Nous proposons, comme pour le programme structuré, des constantes pour paramétrer le programme.

Nous avons mis en évidence un schéma d'abstraction : match, set, jeu et tie-break sont des parties entre adversaires. Il y a `NbJoueurs` adversaires (variable de classe). Pour simplifier, concerne deux joueurs (opération `adversaires`) dont l'un doit servir (opération `serveur`). Ces joueurs sont indexés (à partir du joueur, on trouve son score). Le receveur est l'adversaire qui ne sert pas (opération `receveur`). Le principe est le suivant, la partie est une suite de coups (opération `jouerUnePartie`). Chaque coup est gagné par un joueur qui augmente son score (opération `jouerUnCoup`). La fin de la partie est déterminée dans la mise-à-jour de la partie (opération `majPartie`) et stockée dans la variable `fin` (opération `fin`). Elle est fonction de nombre de points à atteindre (variable de classe `NbPtsMax`). Cette variable évoluant dans les sous-classe,

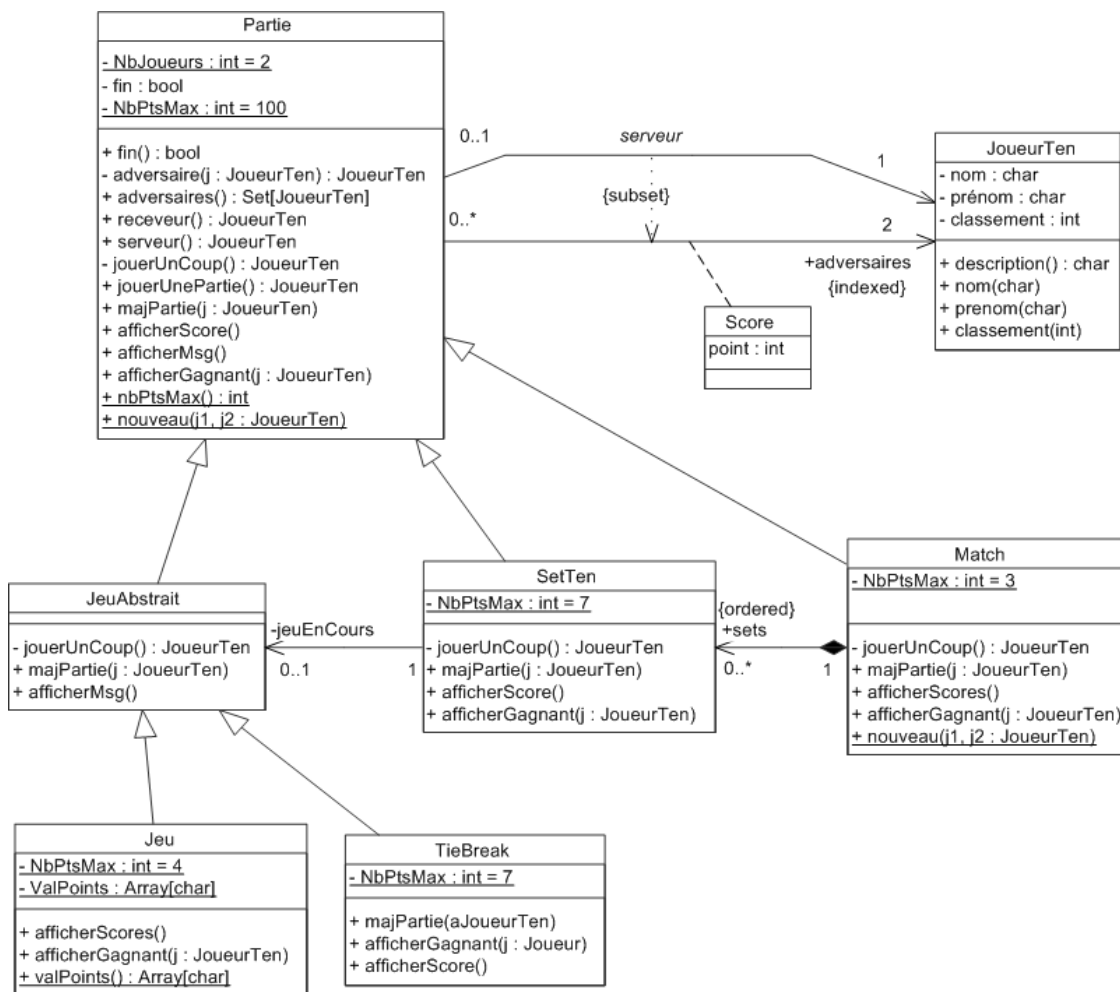


Figure 2 : Conception objet du programme *tennis*

elle sera implantée comme une variable d'instance de la métaclasse en Smalltalk. Un message est affiché en début de partie (opération `afficherMsg`), à chaque coup (opération `afficherScore`) et à la fin de la partie (opération `afficherGagnant`).

context `Partie` inv:

```

self.adversaires->size = Partie.nbJoueurs
-- nous utilisons une notation propre pour accéder à la variable de classe
self.adversaires->includesAll( self.serveur )
-- le serveur est un des adversaires
self.adversaires->forAll( j : JoueurTen | self.score[j] <= Partie.nbPtsMax )
-- le score est limité, s'agissant d'une collection indexée,
-- nous utilisons la notation des associations qualifiées

```

Cette structure est ensuite redéfinie au besoin dans les sous-classes. Nous avons ajouté un niveau d'abstraction pour marquer les éléments communs entre jeu et tie-break. Jouer un coup signifie jouer un points. L'affichage des scores, l'évolution de la partie et le nombre de points diffèrent entre jeu et tie-break. L'affichage des points des jeux ordinaires est noté 0, 15, 30, 40, égalité, avantage (opération `valPoints`). Le changement de service intervient dans le tie-break (opération `majPartie`). Les messages sont personnalisés.

Un set est une partie dont les coups sont des jeux. On ne mémorise dans le score que les résultats des jeux. Le set s'arrête si le nombre de jeux gagnants est atteint et qu'il y a un écart de deux jeux. Un tie break est joué si le score est de 6-6. Le changement de service intervient à chaque jeu (opération `majPartie`). Les affichages de scores varient aussi.

Un match est une partie dont les coups sont des sets. On ne mémorise dans l'ordre les résultats des sets (association `sets`). Le match s'arrête si le nombre de sets gagnants est atteint. Le changement de service est fonction du set précédent. Au départ, le serveur est tiré au sort (opération `nouveau`). Les affichages de

scores varient aussi.

Les contraintes suivantes sont ajoutées aux contraintes de la classe **Partie**. Les joueurs d'un set sont les joueurs du match du set. Le nombre maximum de sets d'un match est de $(\text{Match.nbPtsMax} * 2) - 1$ par exemple 5 avec 3 sets gagnants. Si le match est fini, le vainqueur a exactement nbPtsMax sets gagnants.

```
context Match inv:
  self.sets → forAll( s : SetTen | s.adversaires = self.adversaires )
  self.sets → size ≤ (Match.nbPtsMax * 2) - 1
  -- nbPtsMax est le nombre de sets gagnants d'un match (var. de classe)
  self.fin() implies
    self.sets.adversaires → exists( j : JoueurTen |
      self.score[j] = Match.nbPtsMax and
      ( self.sets.adversaires → forAll( i : JoueurTen |
        i <> j and self.score[i] < Match.nbPtsMax ))
    )
  -- on vérifie l'unicité
```

Pour le set en cours, les adversaires du jeu en cours sont ceux du set. Si le jeu est fini, alors au moins un des scores est supérieur à $\text{nbPtsMax} - 1$ (en principe le set se joue en 6 jeux).

```
context SetTen inv:
  self.jeuEnCours.adversaires = self.adversaires
  self.fin() implies
    self.sets.adversaires → exists( j : JoueurTen |
      self.score[j] ≥ SetTen.nbPtsMax - 1 and
      ( self.sets.adversaires → forAll( i : JoueurTen |
        i <> j and (( self.score[i] ≤ self.score[j] - 2) or
          ( self.score[i] = SetTen.nbPtsMax - 1 and
            self.score[j] = SetTen.nbPtsMax ))))
    )
  -- on vérifie l'unicité avec un éventuel tie-break
```

Dans la classe **JeuAbstrait**, nous avons la contrainte suivante : si le jeu ou le tie-break est fini, alors au moins un des scores est supérieur ou égal à nbPtsMax .

```
context SetTen inv:
  self.fin() implies
    self.sets.adversaires → exists( j : JoueurTen |
      self.score[j] ≥ SetTen.nbPtsMax and
      ( self.sets.adversaires → forAll( i : JoueurTen |
        i <> j and (self.score[i] ≤ self.score[j] - 2)))
    )
```

5 Conception et programmation avec Smalltalk

5.1 Implantation en Smalltalk

Implanter la conception à objets en Smalltalk. Discuter des alternatives de codage.

Le code ci-après est une traduction quasi-systématique de la conception. Les assertions n'ont pas été programmées sous cette forme, mais on peut vérifier qu'elles sont valides. Un certain nombre d'optimisations dues à l'héritage ont été effectuées.

Les associations sont représentées par des variables d'instances, sauf **jeuEnCours** car c'est une variable locale. Du fait des choix de navigation, une seule variable suffit. Elle porte le nom du rôle associé et se trouve dans la classe à l'origine de la navigation. La contrainte **indexed** est représentée en utilisant les dictionnaires de Smalltalk :

```
score : Dictionary [JoueurTen] of Integer.
```

L'héritage est simple et ne pose pas de difficultés. Noter que la variable de classe **NbPtsMax** est définie par une variable d'instance de ma métaclasse car sa valeur est redéfinie par la méthode de classe **initialize** dans les sous-classes de **Partie**.

5.1.1 Code Smalltalk de l'application tennis

Le code est fourni en annexe [A](#).

5.1.2 Exécution

Le résultat est fourni en annexe [B](#).

5.2 Evolution de la conception

Reprenre la question de l'évolution avec la conception à objets.

Nous avons utilisé des variables de classes pour paramétrer le programme. Une factorisation a été obtenue par abstraction dans la classe **Partie**. Une fois cette classe conçue et programmée, l'écriture des sous-classe est rapide. Il faut noter qu'à chaque fois qu'un comportement commun est mis en évidence, il est factorisé dans les super-classes. La structure du programme est bien visible par le modèle des classes de la figure 2.

Discutons maintenant des différentes évolutions proposées.

1. Un joueur peut abandonner la partie. L'abandon est une fonction aléatoire à l'intérieur de la procédure **jouerUnePartie**. A priori, il suffit de mettre à jour la variable **fin** pour arrêter le jeu. Deux précautions sont à prendre :
 - (a) Pour répercuter, l'abandon dans un jeu au niveau du set puis du match, on rend un résultat supplémentaire dans la méthode **jouerUnCoup**. Dans la méthode appelante, on teste ce résultat.
 - (b) Les contraintes posées dans la conception ne sont valables que si le jeu est fini normalement.

La solution consiste donc à ajouter une variable d'instance **abandon** dans la classe **Partie** et à modifier les méthodes suivantes :

- (a) La méthode **jouerUnCoup** rend un paramètre supplémentaire..
- (b) La méthode **jouerUnePartie** boucle jusqu'à la fin ou l'abandon.

Il faut vérifier la cohérence avec la redéfinition de ces méthodes dans les sous-classes.

Seule une partie (verticale) du code est modifiée.

2. Matches en deux sets gagnants.

Il suffit de changer la constante **NbPtsMax** en 2 au lieu de 3. dans la classe **Match**.

3. Tournoi complet.

Un tournoi complet est un ensemble de matches. Des contraintes doivent être ajoutées sur les matches joués : tours, adversaires qualifiés, etc. Le code existant n'est nullement remis en cause : c'est la réutilisation horizontale.

4. Matches en doubles.

Soit on considère l'équipe et non le joueur et il n'y a rien à changer. Soit il faut dissocier la notion de joueur (conservée pour le changement de service) de la notion d'équipe (mémoire des scores).

5. Tennis de tables.

On change les règles de calcul des points (constantes) et la règle de jeu d'un set. Le changement de service est aussi différent.

Les deux dernières évolutions impliquent des modifications assez subtiles dans l'ensemble du programme. Nous avons deux possibilités :

- Soit on recopie l'ensemble de la hiérarchie de classe, en renommant les classes. Puis on modifie localement les méthodes **jouerUnCoup**, **jouerUnePartie** **majPartie** et les variables de classes.
- Soit on définit un schéma d'héritage multiple comme le montre partiellement la figure 3. Pour être exploitable, il faudrait définir un héritage de groupes de classes, qu'on peut qualifier de *pattern* ici.

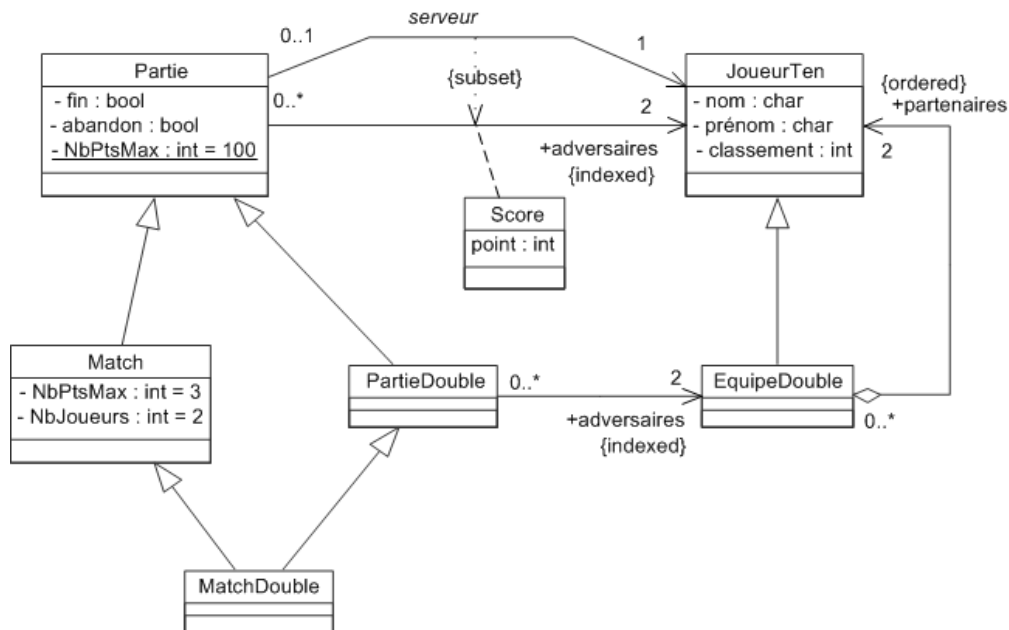


Figure 3 : Conception objet du programme *tennis en double*

5.3 Enseignements

La conception objet est plus perméable aux évolutions que la conception structurée. Par réutilisation verticale (héritage) ou horizontale (copie ou composition), il est souvent aisé de modifier une conception. Néanmoins, l'ajout de nouvelles sous-classe peut amener, pour une utilisation optimale de l'héritage et une meilleure lisibilité du code, à restructurer les classes de niveau intermédiaire.

6 Vers un pattern

Nous avons mis en évidence deux *patterns*. Le premier regroupe l'ensemble des classes de la figure 2. Deux personnalisations sont : parties de doubles et parties de tennis de table. Le second regroupe l'ensemble des classes relative à une partie d'un jeu dans la figure 4.

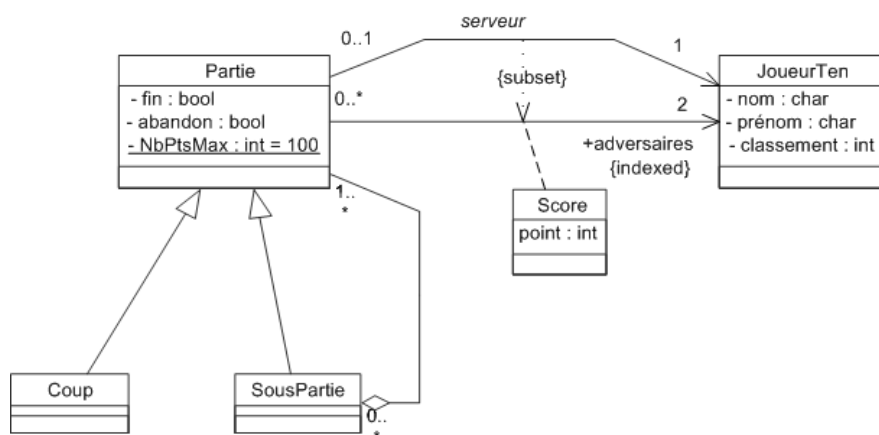


Figure 4 : Conception objet du programme *tennis en double*

7 Bilan

La conception abstraite est en fait issue d'une suite d'itérations, *conception - implantation*. Les classes de Smalltalk jouent un rôle dans la réutilisation.

A travers ce petit exemple, nous avons mis en évidence qu'un découpage en objet permet de définir une meilleure abstraction du code. Ce découpage facilite la lisibilité, la conception, le test des différents modules.

La cohésion des objets et l'héritage facilitent l'extension de l'application et la réutilisation de code. Noter aussi que la notation UML est un bon outil de documentation du code.

A Code Smalltalk de l'application

Listing 1: Code Smalltalk de l'application tennis

```
Object subclass: #JoueurTen
  instanceVariableNames: 'nom prenom classement '
  classVariableNames: ''
  poolDictionaries: ''
  category: 'Tennis'!
JoueurTen comment:
'Cette classe implante la classe JoueurTen du match de tennis.'

Les variables d''instance sont :
nom : String
prenom : String
classement : Number'!

!JoueurTen methodsFor: 'printing'!

description
  "Ecrit le nom, le prénom et le classement dans le flot."

  | aStream |
  aStream ← WriteStream on: (String new: 16).
  self description: aStream.
  ↑aStream contents!

description: aStream
  "Ecrit le nom et le prénom dans le flot."

  aStream cr; nextPutAll: 'Je m''appelle ', nom, ' ', prenom.
  aStream nextPutAll: ' et je suis classé ', classement printString!

printOn: aStream
  "Ecrit le nom et le prénom dans le flot."

  aStream nextPutAll: nom, ' ', prenom! !

!JoueurTen methodsFor: 'accessing'!

classement
  "rend le classement du JoueurTen"

  ↑classement!

classement: aNumber
  "affecte le classement du joueur"

  classement ← aNumber!

nom
  "rend le nom du joueur"

  ↑nom!

nom: aString
  "affecte le nom du joueur"

  nom ← aString!

prenom
  "rend le prenom du joueur"

  ↑prenom!

prenom: aString
  "affecte le prenom du joueur"
```

```

    prenom ← aString! !

    "*****"!
Object subclass: #Partie
    instanceVariableNames: 'score fin serveur '
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Tennis'!
Partie comment:
    'Cette classe implante une version abstraite d''une partie du match de tennis.
    Elle rassemble le comportement commun à un set, un jeu et un tie break.
    La partie est définie par des adversaires dont on mémorise les scores
    dans un dictionnaire, l''adversaire qui sert pour le gain de la partie
    (serveur) un booléen indiquant si la partie est terminée ou pas.

    Les variables d''instance sont :
        score : Dictionary [JoueurTen] : Integer
        serveur : JoueurTen
        fini : Boolean

    - la variable score mémorise le score actuel de chaque adversaire du
      jeu dans un dictionnaire
    - la variable fini mémorise la fin du jeu. Sa valeur pourrait être
      calculée à partir du score, mais nous utilisons une variable pour
      optimiser le temps de traitement.
    - la variable serveur indique le joueur qui sert pour ce jeu.

    Les variables d''instance de la métaclasse sont :
        NbPtsMax : Integer nombre de coups nécessaires pour gagner la partie.'!

!Partie methodsFor: 'initialize-release'!

initializeJ1 : aJoueurTen1 j2: aJoueurTen2
    "Démarre un nouveau jeu, les scores sont mis à zéro pour chaque
    joueur. Par défaut, le serveur est le premier joueur."

    fin ← false.
    score ← Dictionary withKeysAndValues: (Array
        with: aJoueurTen1
        with: 0
        with: aJoueurTen2
        with: 0).
    serveur ← aJoueurTen1.
    ↑self! !

!Partie methodsFor: 'printing'!

afficherGagnant: aJoueurTen
    Transcript cr; show: aJoueurTen printString , ' gagne'!

afficherMsg
    "Affichage d'un message. Version abstraite."!

afficherScores
    "Affichage du score de la partie. Version abstraite."

    Transcript show: (score at: self serveur) printString , ' - ' ,
        (score at: self receveur) printString! !

!Partie methodsFor: 'update'!

jouerUnCoup
    "On tire le gagnant au sort.
    Si le gagnant est 1 alors le serveur gagne, sinon le receveur gagne."

    | gagnant sort |
    sort ← (Random new next * 10000 rem: 2) truncated + 1.
    sort = 1
        ifTrue: [gagnant ← self serveur]
        ifFalse: [gagnant ← self receveur].
    score at: gagnant put: (score at: gagnant)

```

```

        + 1.
    ↑gagnant!

jouerUnePartie
    "On joue les coups jusqu'à ce que la partie soit terminée."
    "On rend le gagnant"

    | gagn |
    self afficherMsg.
    [ self fin ]
        whileFalse:
            [gagn ←self jouerUnCoup.
             self afficherScores .
             self majPartie: gagn].
    self afficherGagnant: gagn.
    ↑gagn!

majPartie: aJoueurTen
    "On met à jour la partie connaissant le gagnant d'un coup"

    fin ←true! !

!Partie methodsFor: 'private'!

adversaire: aJoueurTen
    "rend l'adversaire d'un joueur"

    | adv |
    (score keys includes: aJoueurTen)
        ifFalse: [↑nil]
        ifTrue: [self adversaires do: [:a | a == aJoueurTen
            ifFalse: [adv ←a ]]].
    ↑adv! !

!Partie methodsFor: 'accessing'!

adversaires
    "rend les adversaires opposés dans le jeu"

    ↑score keys!

fin
    "Détermine la fin de la partie. Dans cette version on rend la valeur
    de la variable"

    ↑fin !

receveur
    "rend le serveur du jeu"

    ↑self adversaire: self serveur!

serveur
    "rend le serveur du jeu"

    ↑serveur! !
"-----"!

Partie class
    instanceVariableNames: 'NbPtsMax '!

!Partie class methodsFor: 'instance creation'!

nouveauJ1: aJoueurTen1 j2: aJoueurTen2
    "Démarre une nouvelle partie"

    ↑self new initializeJ1 : aJoueurTen1 j2: aJoueurTen2! !

!Partie class methodsFor: 'accessing'!

nbPtsMax
    "accès à la variable d'instance de la métaclasse"

```

```

↑NbPtsMax! !

!Partie class methodsFor: 'class initialization'!

initialize
    "Partie initialize "

    NbPtsMax ← 0.!!

    "*****"!
Partie subclass: #JeuAbstrait
    instanceVariableNames: ''
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Tennis'!
JeuAbstrait comment:
    'Cette classe implante une version abstraite de la classe Jeu du
    match de tennis. Elle est définie par spécialisation de la classe
    abstraite Partie. Elle rassemble le comportement commun à un jeu et
    un tie break.
    jouerUnCoup signifie jouerUnPoint ici
    jouerUnePartie signifie jouerUnJeu ici'!

!JeuAbstrait methodsFor: 'update'!

jouerUnCoup
    "On tire le gagnant au sort.
    Si le gagnant est 1 alors le serveur gagne, sinon le receveur gagne."

    | gagnant sort |
    sort ← (Random new next * 10000 rem: 2) truncated + 1.
    sort = 1
        ifTrue: [gagnant ← self serveur]
        ifFalse: [gagnant ← self receveur].
    score at: gagnant put: (score at: gagnant)
        + 1.
    ↑gagnant!

majPartie: aJoueurTen
    "On met à jour la partie connaissant le gagnant d'un coup"

    | gagn sg sp |
    gagn ← aJoueurTen.
    sg ← score at: gagn.
    sp ← score at: (self adversaire: gagn).
    sg >= self class nbPtsMax & (sg - sp >= 2) ifTrue: [fin ← true]! !

!JeuAbstrait methodsFor: 'printing'!

afficherMsg
    "Affichage d'un message en début de partie."

    Transcript cr; show: self serveur printString , ' sert'!!

    "*****"!
JeuAbstrait subclass: #Jeu
    instanceVariableNames: ''
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Tennis'!
Jeu comment:
    'Cette classe implante la classe Jeu du match de tennis.
    Elle est définie par héritage de JeuAbstrait.
    Le jeu évolue selon la spécification de la classe abstraite JeuAbstrait.
    Seul l'affichage des scores change.

    Les variables d'instance de la métaclasse sont :
        NbPtsMax : Integer est redéfinie avec la valeur 4.
        ValPoints : Array[String] valeur d'affichage des points
    Cette dernière variable permet de transcrire le score selon les règles habituelles :
    elle contient une table de codage : score --> affichage.
```

```

Elle est utilisée par une méthode de classe valPoints et initialisée à la création de la classe.'!

!Jeu methodsFor: 'printing'!

afficherGagnant: aJoueurTen
    super afficherGagnant: aJoueurTen.
    Transcript show: ' le jeu'!

afficherScores
    "Affichage du score du jeu"

    | gagn |
    (score at: self serveur)
        > (score at: self receveur)
            ifTrue: [gagn ←self serveur]
            ifFalse: [gagn ←self receveur].
    (score at: gagn)
        >= self class nbPtsMax
            ifTrue:
                [| diff |
                 diff ←(score at: gagn)
                     - (score at: (self adversaire: gagn)).
                 diff = 0
                 ifTrue: [Transcript cr; show: 'égalité']
                 ifFalse: [diff = 1
                           ifTrue: [Transcript cr; show: 'avantage ', gagn printString]
                           ifFalse: [fin ←true]]]
            ifFalse: [Transcript cr; tab; show:
                      (self class valPoints at: (score at: self serveur) + 1)
                      , ' - ', (self class valPoints at:
                              (score at: self receveur)+ 1)]! !
    "-----"!

Jeu class
    instanceVariableNames: 'ValPoints '!

!Jeu class methodsFor: 'class initialization'!

initialize
    "Jeu initialize "

    NbPtsMax ←4.
    ValPoints ←#('0' '15' '30' '40' 'jeu')! !

!Jeu class methodsFor: 'accessing'!

valPoints
    "accès à la variable d'instance de la métaclasse"

    ↑ValPoints! !

"*****"!
JeuAbstrait subclass: #TieBreak
    instanceVariableNames: ''
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Tennis'!
TieBreak comment:
    'Cette classe implante la classe Tie Break du match de tennis.
    Elle est définie par héritage de JeuAbstrait.

    Seule l'évolution du jeu change : le serveur change tous les points pairs.

    Les variables d'instance de la métaclasse sont :
        NbPtsMax : Integer est redéfinie avec la valeur 7. '!

!TieBreak methodsFor: 'update'!

majPartie: aJoueurTen
    "On met à jour la partie connaissant le gagnant d'un coup.
    Cette méthode est redéfinie pour inclure le changement de service."

```

```

| gagn sg sp |
gagn ← aJoueurTen.
sg ← score at: gagn.
sp ← score at: (self adversaire: gagn).
sg >= self class nbPtsMax & (sg - sp >= 2)
ifTrue: [fin ← true]
ifFalse: [(sg + sp rem: 2)
= 1
ifTrue:
[serveur ← self receveur.
Transcript tab; show: 'changement de service ', self serveur printString]]! !

!TieBreak methodsFor: 'printing'!

afficherGagnant: aJoueurTen
super afficherGagnant: aJoueurTen.
Transcript show: ' le tie break'!

afficherScores
"Affichage du score de la partie. Version abstraite."

Transcript cr; tab.
super afficherScores ! !
"-----"!

TieBreak class
instanceVariableNames: ''!

!TieBreak class methodsFor: 'class initialization'!

initialize
"TieBreak initialize "

NbPtsMax ← 7.!!

"*****"!
Partie subclass: #SetTen
instanceVariableNames: ''
classVariableNames: ''
poolDictionaries: ''
category: 'Tennis'!
SetTen comment:
'Cette classe implante la classe SetTen (pour ne pas confondre avec la
classe Set) du match de tennis. Elle est définie par héritage de JeuAbstrait.

jouerUnCoup signifie jouerUnJeu ici
jouerUnePartie signifie jouerUnSet ici

L''évolution du jeu et l''affichage des scores changent.
Le serveur change à chaque jeu.

Les variables d''instance de la métaclasse sont :
NbPtsMax : Integer est redéfinie avec la valeur 7. '!

!SetTen methodsFor: 'printing'!

afficherGagnant: aJoueurTen
super afficherGagnant: aJoueurTen.
Transcript show: ' le set'!

afficherScores
"Affichage du score du jeu. Version Set."

Transcript cr; tab; show: 'Set : ', self serveur printString , ' '.
super afficherScores .
Transcript show: ' ', self receveur printString! !

!SetTen methodsFor: 'update'!

jouerUnCoup
"Jouer un coup signifie jouer un jeu dans le set.
Le serveur est donné en premier. "

```

```

| jeuEnCours gagnant |
jeuEnCours ←Jeu nouveauJ1: self serveur j2: self receveur.
gagnant ←jeuEnCours jouerUnePartie.
score at: gagnant put: (score at: gagnant)
+ 1.
↑gagnant!

majPartie: aJoueurTen
    "On joue un set, on change de serveur entre chaque jeu.
    Si le score est de 6 à 6, on joue un tie-break. "

    | gagn sg sp |
    gagn ←aJoueurTen.
    gagn = self receveur ifTrue: [Transcript tab; tab; show:
        ' / Break pour ', self receveur printString].
    sg ←score at: gagn.
    sp ←score at: (self adversaire: gagn).
    sg >= (self class nbPtsMax - 1) & (sg - sp = 1) not
    ifTrue:
        [sg - sp >= 2
        ifTrue: [fin ←true]
        ifFalse:
            ["sg = sp"
            Transcript cr; cr; show: 'Tie break'.
            gagn ←(TieBreak nouveauJ1: self receveur j2: self
                serveur) jouerUnePartie.
            score at: gagn put: (score at: gagn)
            + 1].
            fin ←true]
        ifFalse: [serveur ←self receveur]! !
    "-----"!

SetTen class
    instanceVariableNames: ''!

!SetTen class methodsFor: 'class initialization'!

initialize
    "SetTen initialize "

    NbPtsMax ←7!!

    "*****"!
Partie subclass: #Match
    instanceVariableNames: 'sets adversaires '
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Tennis'!
Match comment:
    'Cette classe implante la classe Match de tennis. Elle est définie
    par héritage de JeuAbstrait car elle définit deux adversaires et
    le déroulement d'une partie.

    Les variables d'instance sont :
        sets : OrderedCollection [SetTen]

    - la variable sets mémorise dans l'ordre les sets joués
    - le score contient le nombre de sets gagnés par joueur.

    Les variables d'instance de la métaclasse sont :
        NbPtsMax : Integer    nombre de sets gagnants pour gagner le match.

    jouerUnCoup signifie jouerUnSet ici
    jouerUnePartie signifie jouerUnMatch ici'!

!Match methodsFor: 'printing'!

afficherGagnant: aJoueurTen
    super afficherGagnant: aJoueurTen.
    Transcript show: ' le match'!

```



```

afficherScores
  "Affichage du score du match.
  Problème : l'ordre des joueurs dans les sets peut varier d'un set à l'autre."

  Transcript cr; tab; show: 'Score du match : '.
  sets do: [:s | s afficherScores ]! !

!Match methodsFor: 'update'!

jouerUnCoup
  "Jouer un coup signifie jouer un set dans le match.
  Le serveur est donné en premier. Les scores sont mémorisés.
  La fin du set détermine le changement de service."

  | gagnant setEnCours |
  setEnCours ← SetTen nouveauJ1: self serveur j2: self receveur.
  gagnant ← setEnCours jouerUnePartie.
  score at: gagnant put: (score at: gagnant)
    + 1.
  sets add: setEnCours.
  serveur ← setEnCours receveur.
  ↑gagnant!

majPartie: aJoueurTen
  "On met à jour le match connaissant le gagnant d'un set.
  Cette méthode est redéfinie pour inclure le changement de service.
  On s'arrête lorsque le nombre de set gagnants est atteint."

  (score at: aJoueurTen)
    = self class nbPtsMax ifTrue: [fin ← true]! !

!Match methodsFor: 'initialize-release'!

initializeJ1 : aJoueurTen1 j2: aJoueurTen2
  "Démarre un nouveau match.
  Par défaut, le serveur est tiré au sort."

  super initializeJ1 : aJoueurTen1 j2: aJoueurTen2.
  (Random new next * 10000 rem: 2) truncated + 1 = 1
    ifTrue: [serveur ← aJoueurTen1]
    ifFalse: [serveur ← aJoueurTen2].
  Transcript cr; show: 'Le tirage au sort donne le service à ',
    serveur printString , '.'.
  sets ← OrderedCollection new.
  ↑self! !
"-----"!

Match class
  instanceVariableNames: ''!

!Match class methodsFor: 'class initialization'!

initialize
  "Match initialize"

  NbPtsMax ← 3.! !

!Match class methodsFor: 'examples'!

unMatch
  "Match unMatch"

  | joueur1 joueur2 leMatch |
  joueur1 ← (JoueurTen new) nom: 'Pierre'; prenom: 'Afeux'; classement: 12.
  joueur2 ← (JoueurTen new) nom: 'René'; prenom: 'Gossie'; classement: 11.
  Transcript cr; show: joueur1 description; show: joueur2 description.
  leMatch ← self nouveauJ1: joueur1 j2: joueur2.
  leMatch jouerUnePartie! !

Partie initialize !
Match initialize !
SetTen initialize !

```

Jeu initialize !
TieBreak initialize !

B Résultat de l'exécution de l'application Smalltalk

```
%inphb.im created at February 26, 2001 10:51:05 am

Je m'appelle Pierre Afeux et je suis classé 12
Je m'appelle René Gossie et je suis classé 11
Le tirage au sort donne le service à René Gossie.
René Gossie sert
  15 - 0
  15 - 15
  15 - 30
  30 - 30
  40 - 30
René Gossie gagne le jeu
  Set : René Gossie 1 - 0 Pierre Afeux
Pierre Afeux sert
  0 - 15
  0 - 30
  0 - 40
  15 - 40
René Gossie gagne le jeu
  Set : Pierre Afeux 0 - 2 René Gossie      / Break pour René Gossie
René Gossie sert
  15 - 0
  15 - 15
  30 - 15
  30 - 30
  30 - 40
Pierre Afeux gagne le jeu
  Set : René Gossie 2 - 1 Pierre Afeux      / Break pour Pierre Afeux
Pierre Afeux sert
  15 - 0
  30 - 0
  40 - 0
  40 - 15
  40 - 30
Pierre Afeux gagne le jeu
  Set : Pierre Afeux 2 - 2 René Gossie
René Gossie sert
  0 - 15
  0 - 30
  15 - 30
  30 - 30
  40 - 30
René Gossie gagne le jeu
  Set : René Gossie 3 - 2 Pierre Afeux
Pierre Afeux sert
  15 - 0
  30 - 0
  30 - 15
  30 - 30
  30 - 40
René Gossie gagne le jeu
  Set : Pierre Afeux 2 - 4 René Gossie      / Break pour René Gossie
René Gossie sert
  0 - 15
  0 - 30
```

15 - 30
 30 - 30
 40 - 30
 40 - 40
 avantage Pierre Afeux
 Pierre Afeux gagne le jeu
 Set : René Gossie 4 - 3 Pierre Afeux / Break pour Pierre Afeux
 Pierre Afeux sert
 0 - 15
 0 - 30
 0 - 40
 René Gossie gagne le jeu
 Set : Pierre Afeux 3 - 5 René Gossie / Break pour René Gossie
 René Gossie sert
 0 - 15
 0 - 30
 0 - 40
 15 - 40
 30 - 40
 Pierre Afeux gagne le jeu
 Set : René Gossie 5 - 4 Pierre Afeux / Break pour Pierre Afeux
 Pierre Afeux sert
 15 - 0
 30 - 0
 40 - 0
 40 - 15
 40 - 30
 40 - 40
 avantage René Gossie
 égalité
 avantage Pierre Afeux
 Pierre Afeux gagne le jeu
 Set : Pierre Afeux 5 - 5 René Gossie
 René Gossie sert
 0 - 15
 15 - 15
 30 - 15
 40 - 15
 René Gossie gagne le jeu
 Set : René Gossie 6 - 5 Pierre Afeux
 Pierre Afeux sert
 15 - 0
 30 - 0
 30 - 15
 30 - 30
 30 - 40
 René Gossie gagne le jeu
 Set : Pierre Afeux 5 - 7 René Gossie / Break pour René Gossie
 René Gossie gagne le set
 Score du match :
 Set : Pierre Afeux 5 - 7 René Gossie
 René Gossie sert
 15 - 0
 30 - 0
 30 - 15
 30 - 30
 30 - 40
 Pierre Afeux gagne le jeu
 Set : René Gossie 0 - 1 Pierre Afeux / Break pour Pierre Afeux
 Pierre Afeux sert
 15 - 0
 15 - 15
 15 - 30

15 - 40
 René Gossie gagne le jeu
 Set : Pierre Afeux 1 - 1 René Gossie / Break pour René Gossie
 René Gossie sert
 15 - 0
 30 - 0
 30 - 15
 30 - 30
 30 - 40
 40 - 40
 avantage René Gossie
 René Gossie gagne le jeu
 Set : René Gossie 2 - 1 Pierre Afeux
 Pierre Afeux sert
 15 - 0
 15 - 15
 15 - 30
 15 - 40
 René Gossie gagne le jeu
 Set : Pierre Afeux 1 - 3 René Gossie / Break pour René Gossie
 René Gossie sert
 15 - 0
 15 - 15
 15 - 30
 15 - 40
 30 - 40
 Pierre Afeux gagne le jeu
 Set : René Gossie 3 - 2 Pierre Afeux / Break pour Pierre Afeux
 Pierre Afeux sert
 0 - 15
 0 - 30
 15 - 30
 30 - 30
 40 - 30
 Pierre Afeux gagne le jeu
 Set : Pierre Afeux 3 - 3 René Gossie
 René Gossie sert
 0 - 15
 15 - 15
 30 - 15
 40 - 15
 René Gossie gagne le jeu
 Set : René Gossie 4 - 3 Pierre Afeux
 Pierre Afeux sert
 0 - 15
 0 - 30
 15 - 30
 30 - 30
 40 - 30
 Pierre Afeux gagne le jeu
 Set : Pierre Afeux 4 - 4 René Gossie
 René Gossie sert
 0 - 15
 15 - 15
 30 - 15
 30 - 30
 30 - 40
 Pierre Afeux gagne le jeu
 Set : René Gossie 4 - 5 Pierre Afeux / Break pour Pierre Afeux
 Pierre Afeux sert
 15 - 0
 30 - 0
 30 - 15

30 - 30
 30 - 40
 René Gossie gagne le jeu
 Set : Pierre Afeux 5 - 5 René Gossie / Break pour René Gossie
 René Gossie sert
 15 - 0
 15 - 15
 15 - 30
 15 - 40
 Pierre Afeux gagne le jeu
 Set : René Gossie 5 - 6 Pierre Afeux / Break pour Pierre Afeux
 Pierre Afeux sert
 15 - 0
 15 - 15
 15 - 30
 15 - 40
 René Gossie gagne le jeu
 Set : Pierre Afeux 6 - 6 René Gossie / Break pour René Gossie

 Tie break
 René Gossie sert
 0 - 1 changement de service Pierre Afeux
 1 - 1
 1 - 2 changement de service René Gossie
 3 - 1
 4 - 1 changement de service Pierre Afeux
 2 - 4
 2 - 5 changement de service René Gossie
 5 - 3
 6 - 3 changement de service Pierre Afeux
 3 - 7
 René Gossie gagne le tie break
 René Gossie gagne le set
 Score du match :
 Set : Pierre Afeux 5 - 7 René Gossie
 Set : Pierre Afeux 6 - 7 René Gossie
 René Gossie sert
 0 - 15
 15 - 15
 15 - 30
 15 - 40
 Pierre Afeux gagne le jeu
 Set : René Gossie 0 - 1 Pierre Afeux / Break pour Pierre Afeux
 Pierre Afeux sert
 0 - 15
 0 - 30
 15 - 30
 15 - 40
 René Gossie gagne le jeu
 Set : Pierre Afeux 1 - 1 René Gossie / Break pour René Gossie
 René Gossie sert
 0 - 15
 15 - 15
 30 - 15
 40 - 15
 René Gossie gagne le jeu
 Set : René Gossie 2 - 1 Pierre Afeux
 Pierre Afeux sert
 0 - 15
 15 - 15
 30 - 15
 40 - 15
 Pierre Afeux gagne le jeu

Set : Pierre Afeux 2 - 2 René Gossie
René Gossie sert
0 - 15
0 - 30
0 - 40
15 - 40
30 - 40
Pierre Afeux gagne le jeu
Set : René Gossie 2 - 3 Pierre Afeux / Break pour Pierre Afeux
Pierre Afeux sert
15 - 0
30 - 0
40 - 0
40 - 15
40 - 30
40 - 40
avantage Pierre Afeux
Pierre Afeux gagne le jeu
Set : Pierre Afeux 4 - 2 René Gossie
René Gossie sert
0 - 15
0 - 30
15 - 30
30 - 30
40 - 30
René Gossie gagne le jeu
Set : René Gossie 3 - 4 Pierre Afeux
Pierre Afeux sert
15 - 0
30 - 0
40 - 0
Pierre Afeux gagne le jeu
Set : Pierre Afeux 5 - 3 René Gossie
René Gossie sert
15 - 0
30 - 0
40 - 0
40 - 15
40 - 30
40 - 40
avantage Pierre Afeux
égalité
avantage René Gossie
René Gossie gagne le jeu
Set : René Gossie 4 - 5 Pierre Afeux
Pierre Afeux sert
0 - 15
0 - 30
0 - 40
15 - 40
30 - 40
40 - 40
avantage Pierre Afeux
Pierre Afeux gagne le jeu
Set : Pierre Afeux 6 - 4 René Gossie
Pierre Afeux gagne le set
Score du match :
Set : Pierre Afeux 5 - 7 René Gossie
Set : Pierre Afeux 6 - 7 René Gossie
Set : Pierre Afeux 6 - 4 René Gossie
René Gossie sert
0 - 15
15 - 15

15 - 30
 15 - 40
 Pierre Afeux gagne le jeu
 Set : René Gossie 0 - 1 Pierre Afeux / Break pour Pierre Afeux
 Pierre Afeux sert
 15 - 0
 30 - 0
 40 - 0
 40 - 15
 40 - 30
 40 - 40
 avantage Pierre Afeux
 Pierre Afeux gagne le jeu
 Set : Pierre Afeux 2 - 0 René Gossie
 René Gossie sert
 0 - 15
 0 - 30
 0 - 40
 15 - 40
 30 - 40
 40 - 40
 avantage Pierre Afeux
 Pierre Afeux gagne le jeu
 Set : René Gossie 0 - 3 Pierre Afeux / Break pour Pierre Afeux
 Pierre Afeux sert
 15 - 0
 15 - 15
 30 - 15
 30 - 30
 30 - 40
 René Gossie gagne le jeu
 Set : Pierre Afeux 3 - 1 René Gossie / Break pour René Gossie
 René Gossie sert
 0 - 15
 0 - 30
 15 - 30
 15 - 40
 Pierre Afeux gagne le jeu
 Set : René Gossie 1 - 4 Pierre Afeux / Break pour Pierre Afeux
 Pierre Afeux sert
 0 - 15
 0 - 30
 0 - 40
 15 - 40
 René Gossie gagne le jeu
 Set : Pierre Afeux 4 - 2 René Gossie / Break pour René Gossie
 René Gossie sert
 0 - 15
 0 - 30
 0 - 40
 Pierre Afeux gagne le jeu
 Set : René Gossie 2 - 5 Pierre Afeux / Break pour Pierre Afeux
 Pierre Afeux sert
 0 - 15
 0 - 30
 0 - 40
 René Gossie gagne le jeu
 Set : Pierre Afeux 5 - 3 René Gossie / Break pour René Gossie
 René Gossie sert
 15 - 0
 30 - 0
 30 - 15
 30 - 30

30 - 40
Pierre Afeux gagne le jeu
Set : René Gossie 3 - 6 Pierre Afeux / Break pour Pierre Afeux
Pierre Afeux gagne le set
Score du match :
Set : Pierre Afeux 5 - 7 René Gossie
Set : Pierre Afeux 6 - 7 René Gossie
Set : Pierre Afeux 6 - 4 René Gossie
Set : René Gossie 3 - 6 Pierre Afeux
Pierre Afeux sert
15 - 0
30 - 0
40 - 0
Pierre Afeux gagne le jeu
Set : Pierre Afeux 1 - 0 René Gossie
René Gossie sert
15 - 0
30 - 0
40 - 0
40 - 15
40 - 30
40 - 40
avantage Pierre Afeux
égalité
avantage René Gossie
René Gossie gagne le jeu
Set : René Gossie 1 - 1 Pierre Afeux
Pierre Afeux sert
0 - 15
15 - 15
30 - 15
40 - 15
Pierre Afeux gagne le jeu
Set : Pierre Afeux 2 - 1 René Gossie
René Gossie sert
0 - 15
0 - 30
15 - 30
15 - 40
30 - 40
40 - 40
avantage René Gossie
René Gossie gagne le jeu
Set : René Gossie 2 - 2 Pierre Afeux
Pierre Afeux sert
15 - 0
30 - 0
30 - 15
30 - 30
30 - 40
René Gossie gagne le jeu
Set : Pierre Afeux 2 - 3 René Gossie / Break pour René Gossie
René Gossie sert
15 - 0
30 - 0
40 - 0
René Gossie gagne le jeu
Set : René Gossie 4 - 2 Pierre Afeux
Pierre Afeux sert
15 - 0
30 - 0
30 - 15
40 - 15

Pierre Afeux gagne le jeu
Set : Pierre Afeux 3 - 4 René Gossie
René Gossie sert
15 - 0
30 - 0
40 - 0
René Gossie gagne le jeu
Set : René Gossie 5 - 3 Pierre Afeux
Pierre Afeux sert
0 - 15
0 - 30
0 - 40
René Gossie gagne le jeu
Set : Pierre Afeux 3 - 6 René Gossie / Break pour René Gossie
René Gossie gagne le set
Score du match :
Set : Pierre Afeux 5 - 7 René Gossie
Set : Pierre Afeux 6 - 7 René Gossie
Set : Pierre Afeux 6 - 4 René Gossie
Set : René Gossie 3 - 6 Pierre Afeux
Set : Pierre Afeux 3 - 6 René Gossie
René Gossie gagne le match

Figure 5 : *Exécution d'un match de tennis*