TSE1 – QCM Mini-projet informatique TDC
le 08/06/2012

Rappel de l'énonce du mini-projet

Il s'agit de développer une application permettant de jouer au morpion. Dans ce jeu, on dispose d'un damier de 3x3 cases et de 2 types de pions (par exemple les O et les X). Chacun des 2 joueurs dépose à son tour un pion sur une case libre. Le gagnant est le premier qui parvient à aligner 3 de ses pions soit sur une ligne, soit sur une colonne soit sur une diagonale. Si le damier est rempli sans qu'aucun des joueurs ne gagne, la partie est déclarée nulle. On peut enchaîner plusieurs parties et décompter le score de chaque joueur.

L'application devra disposer des fonctionnalités suivantes :

1. Gérer une partie à deux joueurs
2. Détecter la fin de la partie et indiquer le gagnant (ou le cas échéant que la partie est nulle).
3. Enchaîner plusieurs parties en tenant à jour le nombre de parties gagnées par chaque joueur (c'est à dire le score des joueurs).
4. Enregistrer la partie en cours et le score des joueurs dans un fichier et permettre de relire un fichier pour reprendre cette parties en cours.
5. Permettre aux 2 joueurs de revoir le déroulement de la partie qui vient d'être jouée.

Partie 1 : Spécifications

Les 2 écrans choisis pour interagir avec l'application sont :

- un écran de menu qui permet à l'utilisateur de voir les scores et de choisir ce qu'il veut faire
- un écran de jeu qui permet de gérer le déroulement d'une partie de morpion.

Les écrans auront l'allure suivante:

Ecran de menu

```
Jeu de morpion
Joueur 1(X) : 0      Joueur 2 (O) : 0

Menu
1) Initialiser les scores
2) Reprendre une partie enregistrée
3) Revoir la dernière partie
4) Commencer une nouvelle partie
5) Enregistrer la partie
6) Quitter l'application

Entrer votre choix :
```

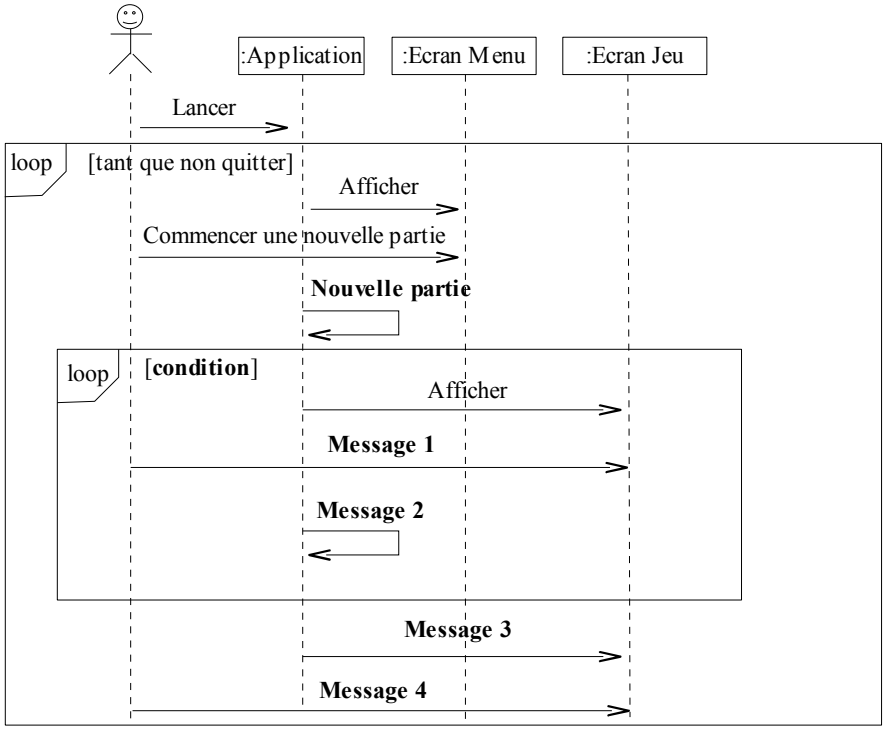
Ecran de jeu

```
Jeu de morpion
Joueur 1 (X) : 0     Joueur 2 (O) : 0

      1  2  3
   A  .  O  .
   B  X  .  .
   C  .  .  .

Joueur 1 c'est à vous :
 - Entrer les coordonnées de la
   case à jouer (ex : A1, B2,...)
 - Taper 0 pour revenir au menu
Entrer votre choix :
```

Sachant que les rectangles « loop » représentent des boucles de type « tant-que », le diagramme de collaboration correspondant au scénario « déroulement d'une partie » est :



1) Que peut-on dire du message *Nouvelle partie* ?
   a) Il devrait être envoyé par l'utilisateur
   b) C'est un message interne à la classe *Application*
   c) Il déclenche l'affichage de l'écran de jeu
   d) Il n'a pas de destinataire

2) Quelle est la condition à utiliser dans le 2ème bloc *loop* ?
   a) Tant que partie non terminée
   b) Tant qu'il n'y a pas de gagnant
   c) Tant que le coup n'est pas permis
   d) Tant que la réponse est incorrecte

3) A quel(s) message(s) peut correspondre *Message 1* ?
   a) Taper sur une touche
   b) Afficher
   c) Saisie de « 0 »
   d) Saisie des coordonnées de la case à jouer

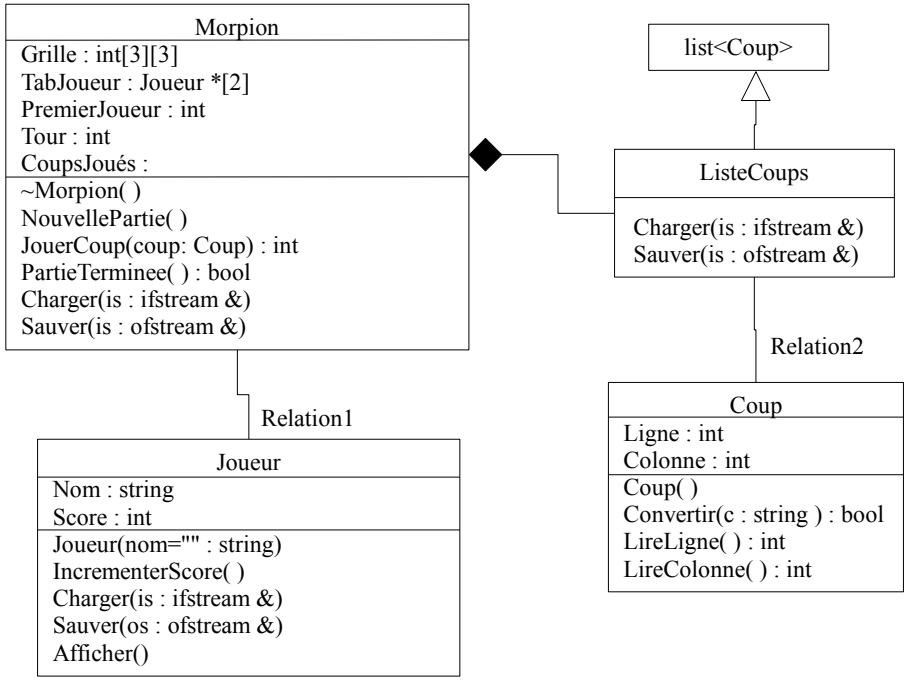
4) A quel(s) message(s) peut correspondre *Message 2* ?
   a) Partie terminée
   b) Ré-afficher l'écran de jeu
   c) Jouer le coup
   d) Sauvegarder le coup joué

5) A quel(s) message(s) peut correspondre *Message 3* ?
   a) Afficher le résultat de la partie
   b) Ré-afficher l'écran de jeu
   c) Proposer l'enregistrement de la partie
   d) Effacer l'écran

6) A quel(s) message(s) peut correspondre *Message 4* ?
   a) Saisir le nom du fichier d'enregistrement
   b) Saisir un coup
   c) Taper sur une touche
   d) Accepter le retour au menu

Partie 2 : Conception

On donne un diagramme de classes de l'application. Dans ce diagramme, certaines méthodes ne figurent pas et certaines informations ont été volontairement omises.



7) A quoi sert la classe *Morpion* ?
   a) Afficher le menu
   b) Gérer déroulement d'une partie
   c) Enregistrer la partie en cours dans un fichier
   d) Saisir un coup

8) De quel type peut-être la relation 1 ?
   a) Une association multiple
   b) Une composition multiple
   c) Une agrégation multiple
   d) Un héritage

9) De quel type peut-être la relation 2 ?
   a) Une association multiple
   b) Une composition multiple
   c) Une agrégation multiple
   d) Un héritage

10) De quel type est le champ *CoupsJoués*
   a) ListeCoups
   b) list<Coups>
   c) vector<Coups>
   d) ListeCoups *

11) A quoi peut servir le champ *Tour* ?
   a) Savoir si la partie est terminée
   b) Savoir qui est le gagnant
   c) Savoir quel joueur doit jouer au prochain coup
   d) Compter le nombre de coups joués

12) A quoi sert la classe *ListeCoups* ?
   a) Permettre de revoir la dernière partie jouée
   b) Convertir le coup saisi en coordonnées grille
   c) Donner la liste des coups possibles
   d) Gérer la liste des coups déjà joués

13) A quoi sert le champ *PremierJoueur* ?
   a) Connaître le symbole du premier joueur
   b) Avoir le numéro du joueur ayant commencé la partie
   c) Avoir le score du meilleur joueur
   d) Connaître l'adresse du premier joueur

14) Que contiennent les éléments du tableau *Grille* ?
   a) Les caractères '.', 'X' et 'O' suivant l'état de la case
   b) Le numéro du coup joué
   c) La valeur 0 au démarrage du jeu
   d) Une valeur indiquant le contenu de la case

Partie 3 : Implantation

Ci dessous le code incomplet d'une méthode :

```
bool Coup::Convertir(string ch)
{
string s1="ABC";
string s2="123";
if (ch.size()!=2) return true;
size_t lig= s1.find_first_of(ch[0]);
size_t col= s2.find_first_of(ch[1]);
if (lig!=string::npos && col!=string::npos)
  {
  Ligne=(int) lig;
  Colonne=(int) col;
  return false;
  }
else return true;
}
```

15) Que doit contenir le paramètre d'entrée *ch* :
   a) l'une des deux chaînes "ABCD" ou "123"
   b) Un caractère représentant une lettre ou un chiffre
   c) Deux caractères : une lettre et un chiffre
   d) Au moins un caractère ou un chiffre

16) La valeur retournée est :
   a) true s'il y a une erreur
   b) true s'il n'y a pas d' erreur
   c) false s'il y a une erreur
   d) false s'il n'y a pas d'erreur

17) Que fait cette méthode :
   a) Elle converti la chaîne d'entrée en coordonnées dans la grille
   b) Elle vérifie si la chaîne d'entrée est correcte
   c) Elle place un pion sur la grille
   d) Elle affiche les repères de la grille

Ci dessous le code incomplet d'une méthode :

```
void Morpion::Sauver(ofstream &os)
{
 à_compléter_1
 à_compléter_1bis
CoupsJoués.Sauver(os)
 à_compléter_2
}
```

18) Par quoi peut-on remplacer *à_completer_1* :
   a) os<<TabJoueur[0]<<endl;
   b) os<<*TabJoueur[0]<<endl;
   c) os.write((char *) TabJoueur[0],sizeof(Joueur));
   d) TabJoueur[0]->Sauver(os);

19) Par quoi peut-on remplacer *à_completer_2* :
   a) os<<CoupsJoués.size()<<endl;
   b) os<<PremierJoueur<<endl;
   c) os<<Tour<<endl;
   d) os<<endl;

Ci dessous le code incomplet d'une autre méthode.

```
int Morpion::JouerCoup(const Coup &c)
{
 int etat;
 Grille[c.LireLigne()][c.LireColonne()]=Tour;
 etat=VerifierCoup();
 à compléter
 Tour=1-Tour;
 return etat;
}
```

20) Par quoi peut-on remplacer *à_completer*
   a) CoupsJoués.Sauver(os);
   b) CoupsJoués[Tour]=c;
   c) CoupsJoués.push_back(new Coup(c));
   d) CoupsJoués.push_back(c);