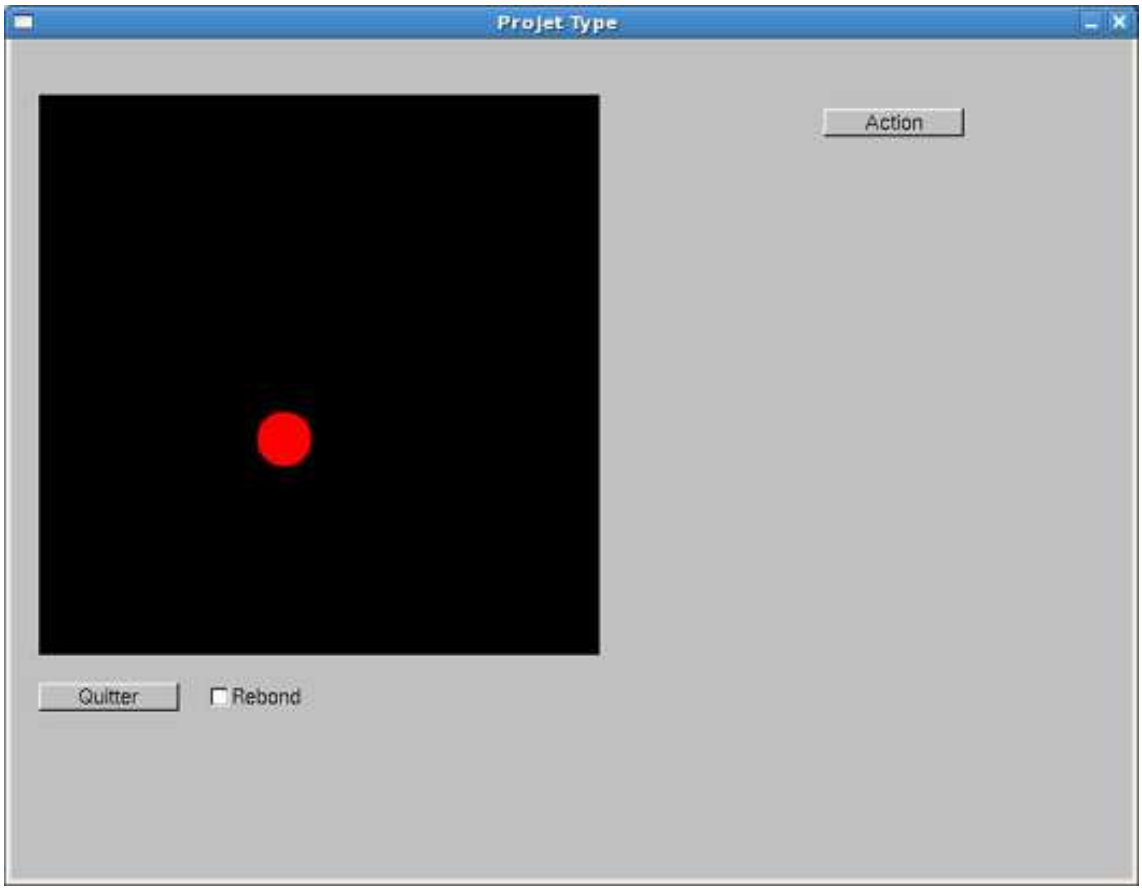
Mastermind graphique pas à pas

Objectifs :

- Prise en main rapide et efficace du projet type
- Réaliser pas à pas un programme mastermind graphique à partir du projet-type-basique
- Ce kit de démarrage ne comprend que 2 boutons (Quitter, Action), une case à cocher (Rebond) et une boule rouge en mouvement



Pas n° 0 : 2 procédures graphiques pour définir les couleurs des pions

u2-dessin.cpp : Ajouter

```
// 2 procedures deja faites fournies pour le projet Mastermind : ChoisirCouleurPion et ChoisirCouleurResultat

// Fixer la couleur des pions de jeu Mastermind
void ChoisirCouleurPion(int Pion) // Correspondance numero de pion avec couleur associee, sert au dessin des pions du jeu
{
    switch (Pion) {
        // -1 : case vide pour placer un pion de couleur
        case -1 : fl_color(FL_DARK2); break;
        case 0 : fl_color(FL_CYAN); break;
        case 1 : fl_color(FL_BLUE); break;
        case 2 : fl_color(FL_RED); break;
        case 3 : fl_color(FL_YELLOW); break;
        case 4 : fl_color(FL_GREEN); break;
        case 5 : fl_color(FL_MAGENTA); break;
    }
}

// Fixer la couleur des pions de resultats-aides Mastermind
void ChoisirCouleurResultat(int Resultat) // Couleurs pion place(1), mal place(0), pas place (-1), sert au dessin des pions resultats-aides
{
    switch (Resultat) {
        case -1 : fl_color(FL_DARK3); break;      // -1 : pas place : gris fonce
        case 0 : fl_color(FL_WHITE); break;       // 0 : bonne couleur, mal place : blanc
        case 1 : fl_color(FL_RED); break;         // 1 : bonne couleur, bien place : rouge
    }
}
```

u2-dessin.h : Ajouter

```
void ChoisirCouleurPion(int Pion);
void ChoisirCouleurResultat(int Resultat);
```

Pas n° 1 : Renommer la fenêtre de projet :

u1-interface.cpp - ligne 22 : Modifier

AVANT :
```
gInterface.Fenetre->label("Projet type") ;
```

APRES :
```
gInterface.Fenetre->label("PhelMastermind") ;
```

Pas n° 2 : Changer la taille de la fenêtre graphique

u1-interface.cpp - ligne 21 : Modifier

AVANT :
```
gInterface.Fenetre = new Fl_Double_Window(800,600);
```
APRES :
```
gInterface.Fenetre = new Fl_Double_Window(530,660);
```

Pas n°3 : Déplacer le bouton « Quitter », la case à cocher « Rebond », le bouton « Action »

u1-interface.cpp - lignes 32, 36 et 40: Modifier

AVANT :
```
gInterface.BoutonQuitter = new Fl_Button(20, 460, 100, 20, "Quitter") ;
gInterface.CaseRebond = new Fl_Check_Button(140, 460, 100, 20, "Rebond") ;
gInterface.BoutonAction = new Fl_Button(580, 50, 100, 20, "Action");
```

APRES :
```
gInterface.BoutonQuitter = new Fl_Button(10*40+20, 15*40+20, 100, 20, "Quitter") ;
gInterface.CaseRebond = new Fl_Check_Button(10*40+20, 14*40+20, 100, 20, "Rebond") ;
gInterface.BoutonAction = new Fl_Button(10*40+20, 40+20, 100, 20, "Action") ;
```

Pas n° 4 : Changer la taille de la zone de dessin :

u1-interface.h - lignes 21 à 24 : Modifier

AVANT :
```
#define X_ZONE   20        // X de la zone
#define Y_ZONE   40        // Y de la zone
#define L_ZONE   400       // Largeur de la zone
#define H_ZONE   400       // Hauteur de la zone
```

APRES :
```
#define X_ZONE   10        // X de la zone
#define Y_ZONE   10        // Y de la zone
#define L_ZONE   400       // Largeur de la zone
#define H_ZONE   640       // Hauteur de la zone
```

Pas n°5 : Dessiner la liste des pions disponibles en haut de la zone graphique

u1-interface.h - ligne 25 : Ajouter

```
#define RAYON_PION   20 // Rayon d'un pion standard
```

u2-dessin.cpp - Ajouter le code suivant après la ligne 22, procédure ZoneDessinDessinerCB :

```
// On dessine la liste des 6 pions disponibles en haut de la zone
ChoisirCouleurPion(0) ;
fl_pie( X_ZONE + 40, Y_ZONE + 40,   2*RAYON_PION,   2*RAYON_PION,   0, 360 );
ChoisirCouleurPion(1) ;
fl_pie( X_ZONE + 2*40, Y_ZONE + 40,   2*RAYON_PION,   2*RAYON_PION,   0, 360 );
ChoisirCouleurPion(2) ;
fl_pie( X_ZONE + 3*40, Y_ZONE + 40,   2*RAYON_PION,   2*RAYON_PION,   0, 360 );
ChoisirCouleurPion(3) ;
fl_pie( X_ZONE + 4*40, Y_ZONE + 40,   2*RAYON_PION,   2*RAYON_PION,   0, 360 );
ChoisirCouleurPion(4) ;
fl_pie( X_ZONE + 5*40, Y_ZONE + 40,   2*RAYON_PION,   2*RAYON_PION,   0, 360 );
ChoisirCouleurPion(5) ;
fl_pie( X_ZONE + 6*40, Y_ZONE + 40,   2*RAYON_PION,   2*RAYON_PION,   0, 360 );
```

Pas n°6 : On dessine les cases du jeu

u2-dessin.cpp : Ajouter

Ajouter les variables pour les boucles en début de procédure ZoneDessinDessinerCB :

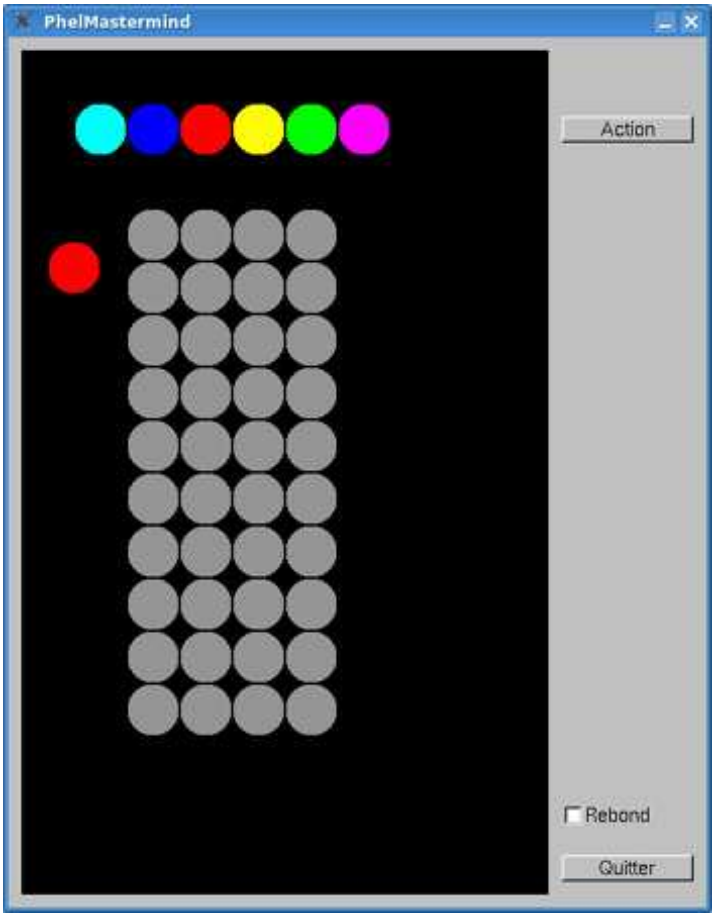
```
int i, j, k;
```

Ajouter en fin de procédure ZoneDessinDessinerCB :

```
for (i=0; i <=9; i++)
{
      for (j=0; j<=3; j++) // Dessin des cases de jeu
      {
          //Dessin des cases de jeu
          ChoisirCouleurPion(-1);
          fl_pie( X_ZONE + 2*40 + j * 40, Y_ZONE + 3*40 + i * 40,   2*RAYON_PION,   2*RAYON_PION,   0, 360 );
      }
}
```

Cette étape n°6 clot la découverte initiale du mode graphique.

Résultat obtenu :



Pas n°7 : Importer le code mastermind mode « console » et la définition des données

Il s'agit ici d'importer le code déjà fait pour le mastermind non graphique

u4-fonctions.h :

Ajouter les constantes supplémentaires

```
#define NB_COULEURS 6                 // nombre de couleurs de pions
#define NB_MAX_COUPS 10               // le joueur doit trouver en moins de NB_MAX_COUPS
#define NB_PIONS 4                    // NB_PIONS dans une combinaison (couleurs differentes)
```

Modifier struct Donnees :

```
struct Donnees
{
    struct Boule      Boule ;
    int               Rebond ;

    // Declaration des variables globales pour les structures de donnees du mastermind
    int Solution[NB_PIONS];           // La solution du jeu tiree au sort par le programme
    int TableauJeu[NB_MAX_COUPS][NB_PIONS];    // Le tableau des pions de toutes les lignes de jeu
    int LigneCourante;                // numero de la ligne de jeu courante entre 0 et NB_MAX_COUPS-1 inclus
    int BienPlaces[NB_MAX_COUPS];     // le nombre de BP et MP pour chaque coup
    int MalPlaces[NB_MAX_COUPS];      // Ces tableaux ne sont pas necessaires en mode texte mais tres utiles en mode
graphique
    int GameOver;                     // 0 : jeu en cours, 1 : jeu fini et perdu, 2 : jeu fini et gagne

};
```

Ajouter dans les déclarations des sous programmes :

```
void CompteBPMP();
```

u4-fonctions.cpp :
Ajouter le code de la procédure InitialiserDonnees déjà réalisée en mode console

```
void InitialiserDonnees()
{

    // Initialisations pour le mastermind
    int couleurDejaTiree[NB_COULEURS];   // drapeau pour le tirage au sort
    int i, j, tirage;

    // On initialise le generateur de nombres aleatoires
    srand(time(NULL));

    gDonnees.LigneCourante = 0;                    // 1ere proposition en attente
    gDonnees.GameOver = 0;                         // 0 : jeu en cours, 1 : jeu fini et perdu, 2 : jeu fini et gagne

    for(i=0; i<NB_MAX_COUPS ; i++)
    {
        gDonnees.BienPlaces[i] = -1;               // valeur impossible tant qu'on n'a pas joue
        gDonnees.MalPlaces[i] = -1;
    }

    for(i=0; i<NB_MAX_COUPS ; i++)
        for(j=0; j<NB_PIONS; j++)
            gDonnees.TableauJeu[i][j]=-1;          // valeur impossible tant qu'on n'a pas joue = pas de couleur

    for(i=0; i<NB_COULEURS; i++)
        couleurDejaTiree[i] = 0;                   // Drapeau pour savoir si une couleur est deja   sortie

    for(i=0; i<NB_PIONS; i++)                      // Tirage au sort de la solution avec 4 couleurs differentes
    {
        do
            tirage = rand() % NB_COULEURS;         // on tire une couleur au sort entre 0 et NB_COULEURS-1
        while(couleurDejaTiree[tirage]!=0);        // jusqu'a   ce que l'on obtienne une couleur pas sortie

        couleurDejaTiree[tirage]=1;                // drapeau leve
        gDonnees.Solution[i]=tirage;
    }

}
```

u4-fonctions.cpp :
Ajouter le code de la procédure CompteBPMP déjà réalisée en mode console

```
// Compte des BP et MP
void CompteBPMP()
{
    int j, k;
    int CopieSolution[4]; // Copie de la solution aidant pour le calcul du nombre de pions de bonne couleur mal places

    gDonnees.BienPlaces[gDonnees.LigneCourante]=0;                  // 0 a priori (avant les tests)
    gDonnees.MalPlaces[gDonnees.LigneCourante]=0;

    for(j=0; j<NB_PIONS; j++)
        if (gDonnees.Solution[j]==gDonnees.TableauJeu[gDonnees.LigneCourante][j]) // Test si pion de bonne couleur
bien place
            gDonnees.BienPlaces[gDonnees.LigneCourante]++;

    // Comptage des pions de bonne couleur mal places - solution prenant en compte des pions joues de meme couleur
    for (k=0;k<=3;k++) // Initialisation de la copie de la solution
        CopieSolution[k] = gDonnees.Solution[k];

    for (j=0;j<=3;j++) // Calcul du nombre de pions de bonne couleur (gMalPlaces(LigneCourante] va inclure dans ce
premier temps les bien ou mal places)
        for (k=0; k<=3; k++)
        {
            if (gDonnees.TableauJeu[gDonnees.LigneCourante][j] == CopieSolution[k])
            {
                gDonnees.MalPlaces[gDonnees.LigneCourante]++;
                CopieSolution[k] = -1; // Mise a -1 (couleur impossible) pour eviter de prendre en compte plusieurs fois la
meme couleur
                break; // On sort de la boucle for si la couleur a ete trouvee pour eviter de compter une couleur plusieurs
fois
            }
        }

    gDonnees.MalPlaces[gDonnees.LigneCourante] = gDonnees.MalPlaces[gDonnees.LigneCourante] -
gDonnees.BienPlaces[gDonnees.LigneCourante]; // Calcul final du nombre de pions de bonne couleur mal places

}
```

Nota bene : pas de changements graphiques visibles après cette étape.

Pas n°8 : Pour commencer à comprendre

Le principe général du projet type fltk est le suivant :

- Les données du jeu sont traitées séparément du graphisme.
- La partie graphique ne fait qu'afficher la situation des données du jeu à chaque instant

u2-dessin.cpp - modifier l'affichage du tableau de jeu en le liant aux données du jeu

Remplacer :

```
for (i=0; i <=9; i++)
{
    for (j=0; j<=3; j++) // Dessin des cases de jeu
    {
        //Dessin des cases de jeu
        ChoisirCouleurPion(-1);
```

Par :

```
for (i=0; i <=9; i++)
{
    for (j=0; j<=3; j++) // Dessin des cases de jeu
    {
        //Dessin des cases de jeu
        ChoisirCouleurPion(gDonnees.TableauJeu[i][j]);
```

Petit exemple pour commencer à comprendre :

- Tester le nouvel affichage graphique en modifiant l'initialisation des données du tableau de jeu (mettre les pions de jeu à 2 au lieu de -1). Résultat obtenu : affichage des cases de jeu en rouge.

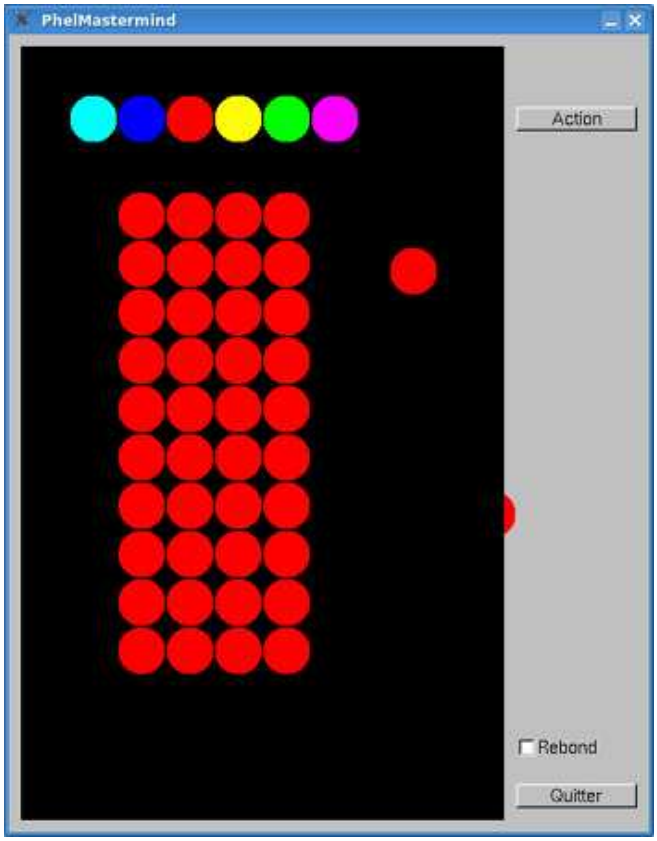
u4-fonctions.cpp - ligne 50

Remplacer

```
for(i=0; i<NB_MAX_COUPS ; i++)
    for(j=0; j<NB_PIONS; j++)
        gDonnees.TableauJeu[i][j]=-1;
```

Par :

```
for(i=0; i<NB_MAX_COUPS ; i++)
    for(j=0; j<NB_PIONS; j++)
        gDonnees.TableauJeu[i][j]=2;
```



Puis remettre la valeur initiale dans u4-fonctions.cpp : gTableauJeu[i][j]=-1;

Pas n°9 - Gestion du clic souris

u4-fonctions.h :

Ajouter dans struct Donnees
int PionCourant; // Pion de couleur selectionne courant parmi les 6 possibles, numero de 0 a NB_COULEURS-1, utile en mode graphique uniquement

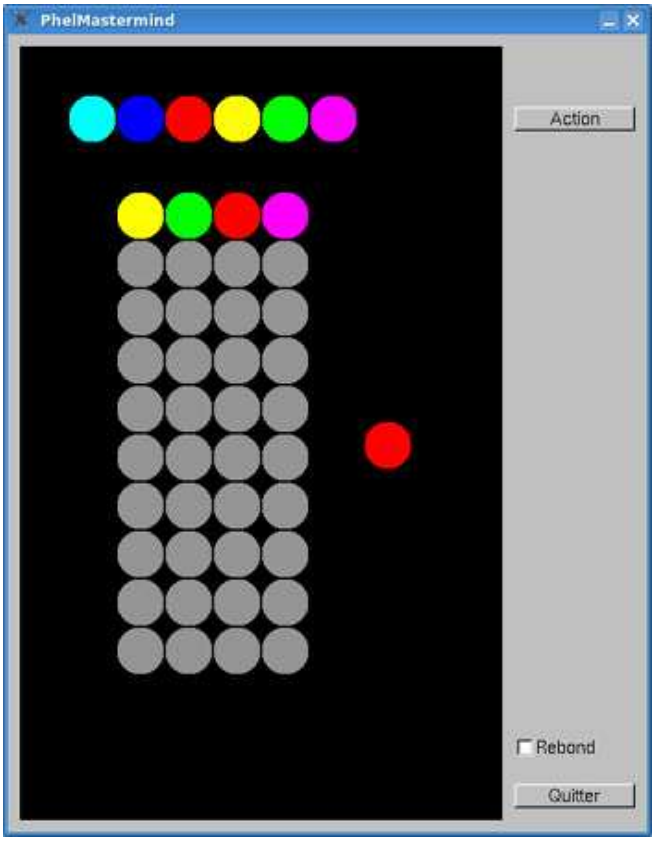
u4-fonctions.cpp :

Ajouter dans la procédure InitialiserDonnees
gDonnees.PionCourant = -1; // debut de jeu : aucun pion de couleur selectionne pour jouer

u3-callbacks.cpp - remplacer le code existant de ZoneDessinSourisCB par :

```
// ATTENTION : X et Y ne sont pas relatifs a la zone mais a la fenetre qui la contient !!!!
// Définition des variables
int X, Y ;              // X et Y du clic
int IClic, JClic;

if ( Fl::event() == FL_PUSH )
{
    // Gestion des clics souris du jeu mastermind

    // Recuperation des coordonnees X,Y du clic souris
    // ATTENTION : X et Y ne sont pas relatifs à la zone mais à la fenetre qui la contient !!!!"
    X =   Fl::event_x() ;
    Y = Fl::event_y() ;

    // Transformation des coordonnees du clic souris en coordonnes IClic/JClic du tableau de jeu
    JClic = ( X - X_ZONE ) / 40;
    IClic = ( Y - Y_ZONE ) / 40;

    if (IClic == 1 && JClic > 0 && JClic < 7) // Pion de couleur selectionne
        gDonnees.PionCourant = JClic -1;

    if (IClic-3 == gDonnees.LigneCourante && JClic > 1 && JClic < 6)
        gDonnees.TableauJeu[IClic-3][JClic-2]=gDonnees.PionCourant;

    // On redessine la zone
    gInterface.ZoneDessin->redraw() ;
}
```

Résultat obtenu : on peut choisir une couleur de pion et la positionner sur la première ligne de jeu



Pas n°10 : Ajout d'un bouton valider pour valider la ligne de jeu (si elle est complète) et passer à la suivante

Modification du bouton « Action » existant

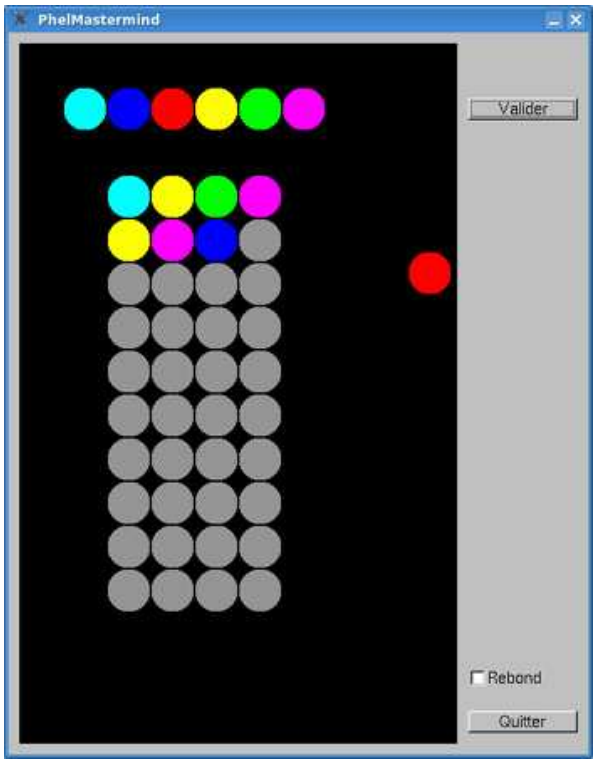
u1-interface.cpp - ligne 40 : modifier le libellé du bouton

```
gInterface.BoutonAction = new Fl_Button(10*40+20, 40+20, 100, 20, "Valider") ;
```

u3-callbacks.cpp : remplacer le code existant de void BoutonActionCB(Fl_Widget* w, void* data) par :

```
    cout << "BoutonValiderCB" << endl;
    int j;
    int CompteurPionsJoues = 0;

    for(j=0;j<=3;j++)
       if (gDonnees.TableauJeu[gDonnees.LigneCourante][j] != -1) CompteurPionsJoues++;

    if (CompteurPionsJoues == 4)
    {
        CompteBPMP();
        if (gDonnees.BienPlaces[gDonnees.LigneCourante] == 4)
        {
                gDonnees.GameOver = 2;
        }
        else if(gDonnees.LigneCourante < NB_MAX_COUPS-1)
        {
            gDonnees.LigneCourante++;
            gInterface.Fenetre->redraw();
        }
        else gDonnees.GameOver = 1;

        // On redessine la zone
        gInterface.ZoneDessin->redraw() ;
    }
```

Résultat obtenu :

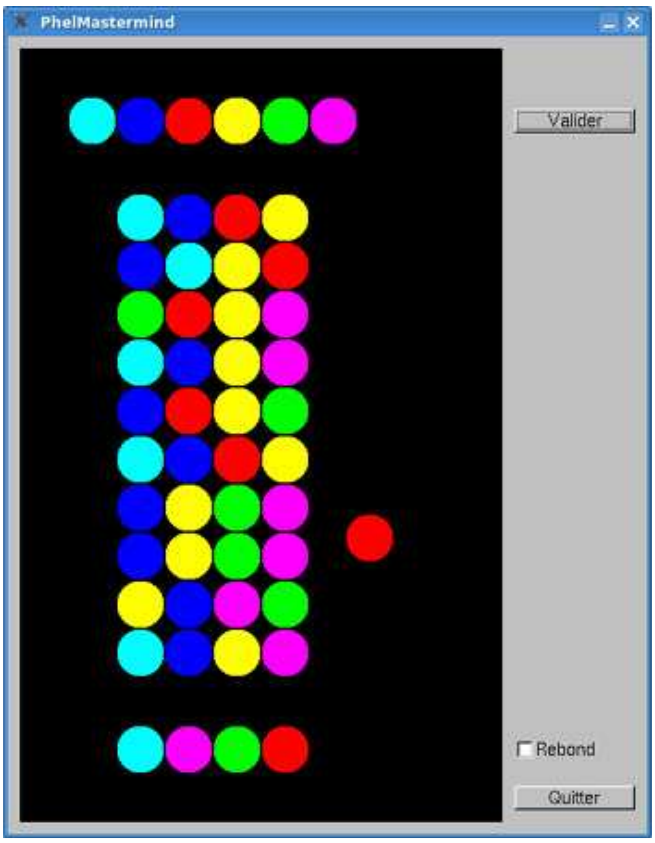


Pas n°11 : Gestion de la fin de jeu : gagné ou nombre maximal d'essais atteint, affichage de la solution

u2-dessin.cpp : ajouter à la fin de la procédure ZoneDessinDessinerCB

```
if (gDonnees.GameOver != 0) // Dessin de la solution si jeu fini (perdu ou gagne)
{
    for(i=0;i<=3;i++) // determination de la couleur de chaque pion de la solution et affichage
    {
        ChoisirCouleurPion(gDonnees.Solution[i]);
        fl_pie( X_ZONE + 2*40 + i * 40, Y_ZONE + 14 * 40,   2*RAYON_PION,   2*RAYON_PION,   0, 360 );
    }
}
```

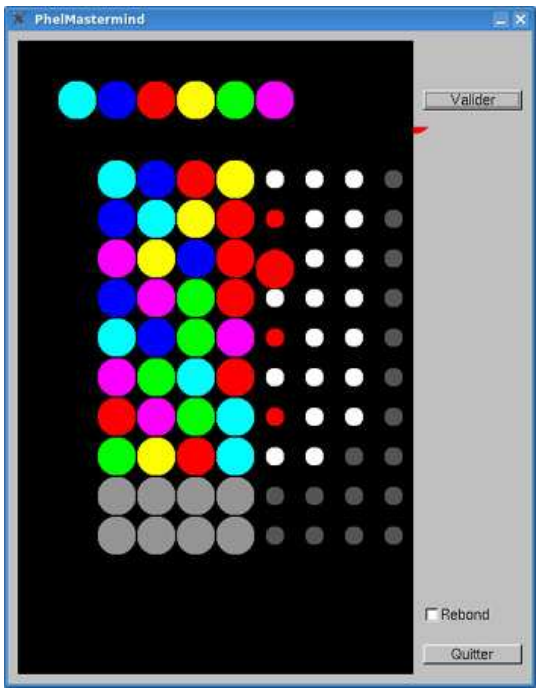
Résultat obtenu

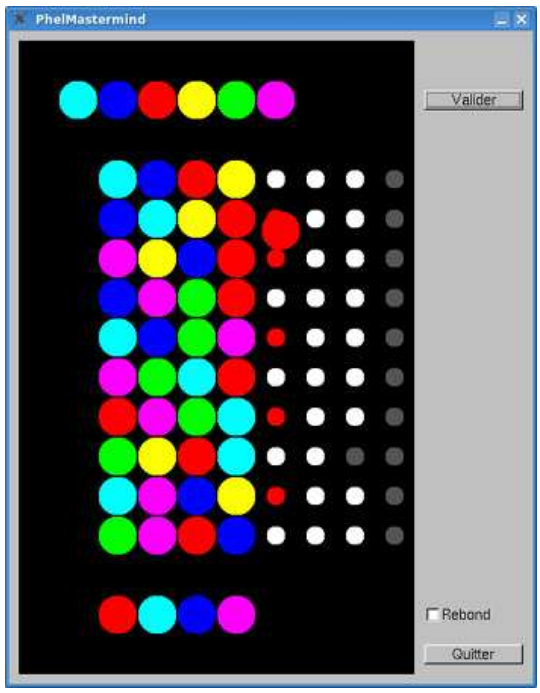


Pas n°12 – Affichage des pions de bonne couleur bien placés et mal placés

u2-dessin.cpp : remplacer le code de la boucle d'affichage des cases de jeu par :

```
    // On dessine les cases de jeu et les cases de resultats/aides
    for (i=0; i <=9; i++)
    {
        for (j=0; j<=3; j++) // Dessin des cases de jeu
        {
            //Dessin des cases de jeu
            ChoisirCouleurPion(gDonnees.TableauJeu[i][j]);
            fl_pie( X_ZONE + 2*40 + j * 40, Y_ZONE + 3*40 + i * 40,   2*RAYON_PION,   2*RAYON_PION,   0, 360
);
        }

        k=0;
        while( k < gDonnees.BienPlaces[i] ) // Dessin des pions de bonne couleur bien places
        {
            ChoisirCouleurResultat(1);
            fl_pie( X_ZONE + 6*40 + RAYON_PION/2 + k * 40, Y_ZONE + 3*40 + RAYON_PION/2 + i * 40,
RAYON_PION,   RAYON_PION,   0, 360 );
            k++;
        }
        while( k < gDonnees.BienPlaces[i] + gDonnees.MalPlaces[i] ) // Dessin des pions de bonne couleur mal places
        {
            ChoisirCouleurResultat(0);
            fl_pie( X_ZONE + 6*40 + RAYON_PION/2 + k * 40, Y_ZONE + 3*40 + RAYON_PION/2 + i * 40,
RAYON_PION,   RAYON_PION,   0, 360 );
            k++;
        }
        while( k < NB_PIONS ) // Dessin des pions de mauvaise couleur
        {
            ChoisirCouleurResultat(-1);
            fl_pie( X_ZONE + 6*40 + RAYON_PION/2 + k * 40, Y_ZONE + 3*40 + RAYON_PION/2 + i * 40,
RAYON_PION,   RAYON_PION,   0, 360 );
            k++;
        }

    }
```





Pas n°13 – Ajout du bouton « Solution »

1) Suppression de la case à cocher « Rebond » qui n'est pas utile pour le mastermind

u1-interface.cpp - dans la procédure CreerInterface, supprimer (ou mettre en commentaire) :

```
// Creation de la case a cocher Rebond
// gInterface.CaseRebond = new Fl_Check_Button(10*40+20, 14*40+20, 100, 20, "Rebond") ;
// gInterface.CaseRebond->callback( CaseRebondCB, NULL ) ;
```

Dans la procédure ActualiserInterface, supprimer (ou mettre en commentaire) :

```
// Actualisation de Rebond
// gInterface.CaseRebond->value( gDonnees.Rebond ) ;
```

u3-callbacks.cpp :

Supprimer ou mettre en commentaires la procédure CaseRebondCB :

```
//void CaseRebondCB(Fl_Widget* w, void* data)
//{
//      gDonnees.Rebond = gInterface.CaseRebond->value() ;
//}
```

Dans la procédure TraiterCycleCB, supprimer ou mettre en commentaires tout sauf DeplacerBouleSansRebond :

```
// Deplacement de la boule
//if ( gDonnees.Rebond == 0 )
DeplacerBouleSansRebond() ;
//else
//      DeplacerBouleAvecRebonds() ;
```

u4-fonctions.cpp :

Dans la procédure InitialiserDonnees, supprimer (ou mettre en commentaire) :

```
 // Initialisation de Rebond
 //gDonnees.Rebond = 0 ;
```

2) Création du nouveau bouton « Solution »

u1-interface.h :
Dans struct Interface, ajouter la variable globale :
```
Fl_Button*		BoutonSolution ;
```

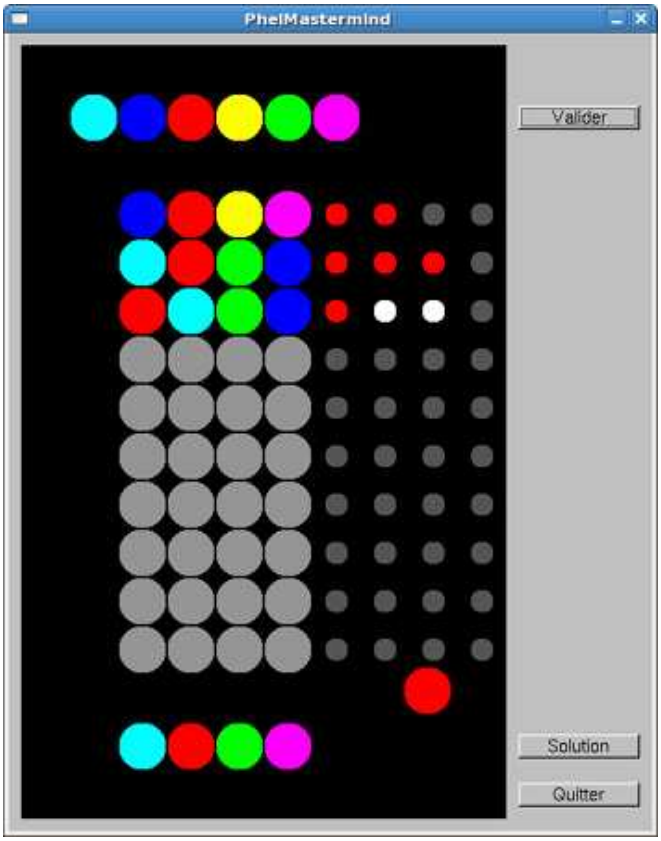
u1-interface.cpp :
Dans la procédure CreerInterface, ajouter après le bouton « Valider » :
```
    // Creation du bouton Solution
    gInterface.BoutonSolution = new Fl_Button(10*40+20, 14*40+20, 100, 20, "Solution") ;
    gInterface.BoutonSolution->callback( BoutonSolutionCB, NULL ) ;
```

u3-callbacks.cpp - ajouter :
```
void BoutonSolutionCB(Fl_Widget* w, void* data)
{
    cout << "BoutonSolutionCB" << endl;
    gDonnees.GameOver = 1;

    // On redessine la zone
    gInterface.ZoneDessin->redraw();
}
```

u3-callbacks.h - ajouter la déclaration :
```
void BoutonSolutionCB( Fl_Widget* w, void* data ) ;
```

Avec ce bouton « Solution », on peut tester que tout le programme marche bien (si on affiche la solution, on peut continuer à jouer quand même et tester le reste du programme...).

Résultat obtenu :



Pas n°14 – Creation du bouton « Rejouer »

1) Déplacement du bouton « Valider » au niveau de la 1ère ligne de jeu, pour faire la place pour le bouton « Rejouer»

u1-interface.cpp - dans la procédure CreerInterface
AVANT :
gInterface.BoutonAction = new Fl_Button(10*40+20, 40+20, 100, 20, "Valider") ;
APRES :
gInterface.BoutonAction = new Fl_Button(10*40+20, 3*40+20, 100, 20, "Valider") ;

2) Création du nouveau bouton « Rejouer »

u1-interface.h :
Dans struct Interface, ajouter la variable globale :
Fl_Button* BoutonRejouer ;

u1-interface.cpp :
Dans la procédure CreerInterface, ajouter après le bouton « Valider » :

```
// Creation du bouton Rejouer
gInterface.BoutonRejouer = new Fl_Button(10*40+20, 40+20, 100, 20, "Rejouer") ;
gInterface.BoutonRejouer->callback( BoutonRejouerCB, NULL ) ;
```

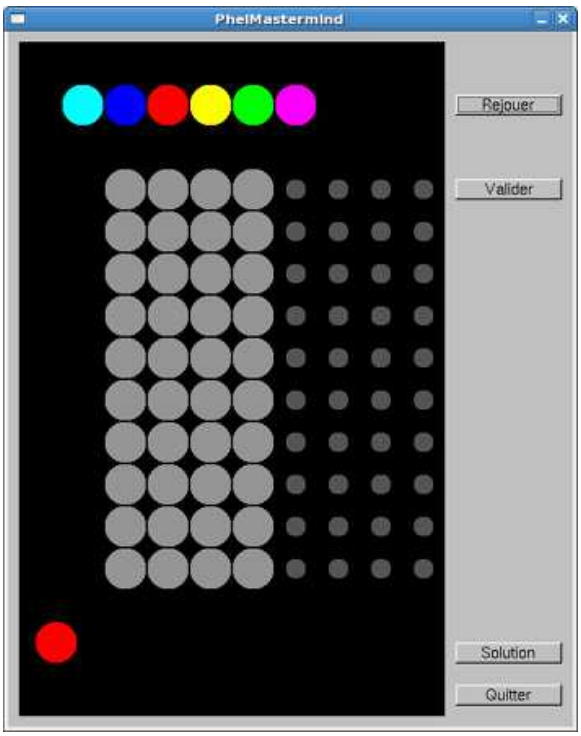
u3-callbacks.cpp - ajouter :

```
void BoutonRejouerCB(Fl_Widget* w, void* data)
{
    printf("BoutonRejouerCB\n");
    InitialiserDonnees();
}
```

u3-callbacks.h - ajouter la déclaration :

```
void BoutonRejouerCB( Fl_Widget* w, void* data ) ;
```

Résultat obtenu :

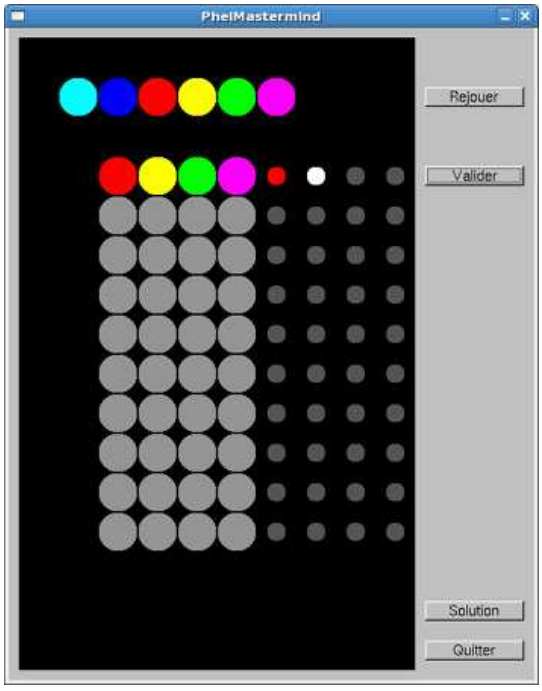


Pas n°15 : Suppression de la boule mobile du projet type initial

u2-dessin.cpp - dans la procédure ZoneDessinDessinerCB, supprimer (ou mettre en commentaire) :

```
// On dessine la boule
// fl_color(FL_RED) ;
// fl_pie( X_ZONE + gDonnees.Boule.X - RAYON_BOULE, Y_ZONE + gDonnees.Boule.Y - RAYON_BOULE,
2*RAYON_BOULE,   2*RAYON_BOULE,   0, 360 );
```

Résultat obtenu :



Nettoyage plus complet :

u3-callbacks.cpp - supprimer ou commenter le code :

```
//      DeplacerBouleSansRebond() ;
```

u4-fonctions.cpp - supprimer les procédures DeplacerBouleSansRebond et DeplacerBouleAvecRebonds

u4-fonctions.h - supprimer les déclarations :

```
// Declaration des sous-programmes
//void DeplacerBouleSansRebond() ;
//void DeplacerBouleAvecRebonds() ;
```

Pas n°16 – Finitions – code optionnel complet pour parfaire le jeu

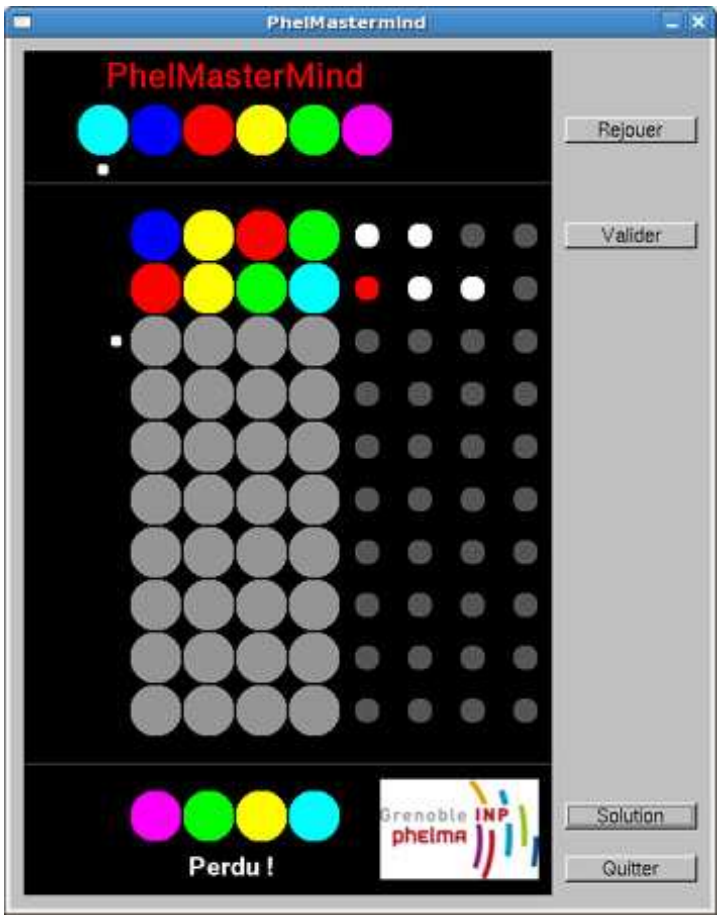
u2-dessin.cpp :
Ajouter les déclarations include suivantes au début :

```
#include <string.h>                    // ajoute pour le mastermind les fonctions strcpy ...
#include <FL/Fl_BMP_Image.H>           // ajoute la gestion des images externes de format BMP
```

Ajouter le code suivant :

```
    // Tout le code ci-dessous est optionnel

    // Optionnel : Indicateur du pion selectionne
    if(gDonnees.PionCourant != -1)
    {
        fl_color(FL_WHITE);
        fl_pie( X_ZONE + 40 + 15 + gDonnees.PionCourant * 40, Y_ZONE + 2 * 40 + 5, 10, 10,   0, 360 );
    }

    // Optionnel : Marquage de la ligne de jeu en cours
    fl_color(FL_WHITE);
    fl_pie(X_ZONE + 40 + 25, Y_ZONE + 3*40+ 15 + gDonnees.LigneCourante*40, 10, 10,   0, 360 );

    // Optionnel : Message de GameOver
    char TexteGagnePerdu[30];

    if (gDonnees.GameOver != 0)
    {
        if (gDonnees.GameOver == 2)
            strcpy(TexteGagnePerdu,"Gagne !");
        else
            strcpy(TexteGagnePerdu,"Perdu !");
        fl_font(FL_HELVETICA_BOLD, 20);
        fl_color(FL_WHITE);
        fl_draw(TexteGagnePerdu, X_ZONE + 3 * 40 + 5, Y_ZONE + 16 * 40 - 15);
    }

    // Optionnel : Titre du jeu
    char TexteTitre[30];
    fl_font(FL_HELVETICA_BOLD, 28);
    fl_color(FL_RED);
    strcpy(TexteTitre,"PhelMasterMind");
    fl_draw(TexteTitre, X_ZONE + 40 + 22, Y_ZONE + 28);

    // Optionnel : lignes de separation des zones de jeu
    fl_color(FL_DARK3);
    fl_line(X_ZONE,Y_ZONE+2*40+20, X_ZONE+10*40-1, Y_ZONE+2*40+20);
    fl_line(X_ZONE,Y_ZONE+13*40+20, X_ZONE+10*40-1, Y_ZONE+13*40+20);

    // Optionnel : affichage du logo phelma
    Fl_BMP_Image ImageLogoPhelma("media/logophelma.bmp") ;
    //Fl_JPEG_Image ImageLogoPhelma("media/logophelma.jpg") ;
    ImageLogoPhelma.draw(X_ZONE+6*40+30, Y_ZONE + 13*40+32);
```

Résultat obtenu :



Pas n°17 : Finition avancée : déplacement du bouton « Valider » selon la ligne de jeu en cours

u3-callbacks.cpp :
Dans la procédure BoutonActionCB, remplacer le code :

```
else if(gDonnees.LigneCourante < NB_MAX_COUPS-1)
{
    gDonnees.LigneCourante++;
}
```

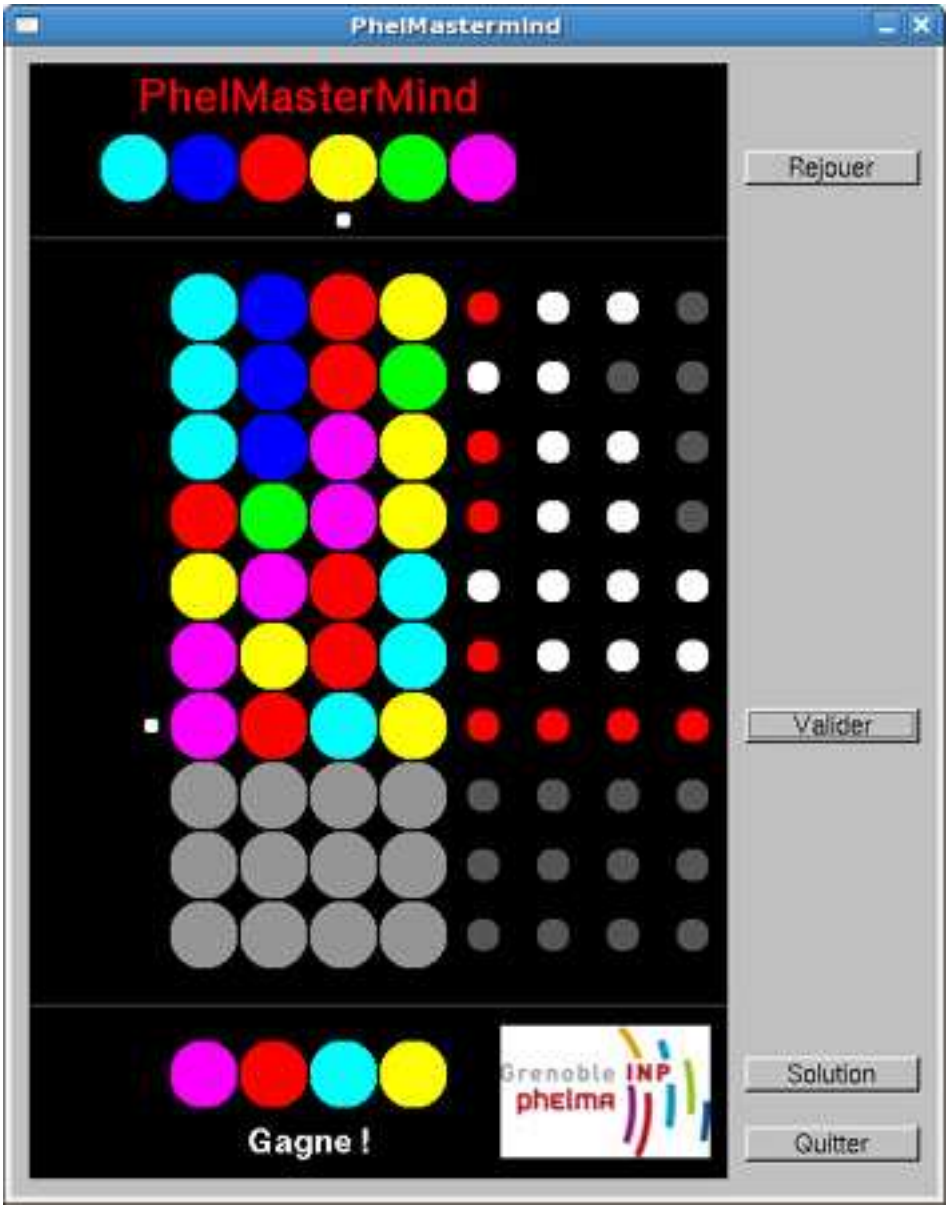
Par le code suivant :

```
else if(gDonnees.LigneCourante < NB_MAX_COUPS-1)
{
     gDonnees.LigneCourante++;
     gInterface.BoutonAction->position(10*40+20,(gDonnees.LigneCourante+3)*40+20);
     gInterface.Fenetre->redraw();
     // Solution alternative plus localisee pour redessiner le bouton valider
     //gInterface.BoutonAction ->hide();
     //gInterface.BoutonAction ->redraw();
     //gInterface.BoutonAction ->show();
}
```

Dans la procédure BoutonRejouerCB, ajouter le code :

```
gInterface.BoutonAction->position(10*40+20,(gDonnees.LigneCourante+3)*40+20); // Postionner le bouton Valider en face de la ligne de jeu courante
gInterface.Fenetre->redraw();
// Solution alternative plus localisee pour redessiner le bouton valider
//gInterface.BoutonAction->hide();
// gInterface BoutonAction->redraw();
// gInterface BoutonAction->show();
```

Résultat final obtenu :



Pas n°18 : Finition avancée : prise en compte si des pions de même couleur sont joués

u4-fonctions.cpp : modification de la procédure compteBPMP

```
// Compte des BP et MP
void CompteBPMP()
{
    int j, k;
    int CopieSolution[4]; // Copie de la solution aidant pour le calcul du nombre de pions de bonne couleur mal places

    gDonnees.BienPlaces[gDonnees.LigneCourante]=0;                  // 0 a priori (avant les tests)
    gDonnees.MalPlaces[gDonnees.LigneCourante]=0;

    for(j=0; j<NB_PIONS; j++)
        if (gDonnees.Solution[j]== gDonnees.TableauJeu[gDonnees.LigneCourante][j]) // Test si pion de bonne couleur
bien place
            gDonnees.BienPlaces[gDonnees.LigneCourante]++;

    // Comptage des pions de bonne couleur mal places - solution prenant en compte des pions joues de meme couleur
    for (k=0;k<=3;k++) // Initialisation de la copie de la solution
        CopieSolution[k] = gDonnees.Solution[k];

    for (j=0;j<=3;j++) // Calcul du nombre de pions de bonne couleur (gMalPlaces(LigneCourante] va inclure dans ce
premier temps les bien ou mal places)
        for (k=0; k<=3; k++)
        {
            if (gDonnees.TableauJeu[gDonnees.LigneCourante][j] == CopieSolution[k])
            {
                gDonnees.MalPlaces[gDonnees.LigneCourante]++;
                CopieSolution[k] = -1; // Mise a -1 (couleur impossible) pour eviter de prendre en compte plusieurs fois la
meme couleur
                break; // On sort de la boucle for si la couleur a ete trouvee pour eviter de compter une couleur plusieurs
fois
            }
        }

    gDonnees.MalPlaces[gDonnees.LigneCourante] = gDonnees.MalPlaces[gDonnees.LigneCourante] -
gDonnees.BienPlaces[gDonnees.LigneCourante]; // Calcul final du nombre de pions de bonne couleur mal places
}
```