

# Rendu final écrit

- Cédric Bevilacqua
- Tommy Saucey
- Kévin Williams
- Théo Lecoublet
- Julien Pactat

## Introduction

Le projet Qwirkle a été une très bonne expérience pour nous tous. Il nous a permis de mettre en pratique notre cours dans un projet commun mais ça a aussi été l'occasion d'échanger nos connaissances, d'apprendre de nouvelles choses annexes, de mettre au point des algorithmes complexes dans un cas concret choses que nous ne faisons qu'en TD sans réel autre but que de terminer le sujet et surtout d'appliquer ce que nous avons appris dans un projet de plus grande envergure. Nous sommes fiers de vous présenter notre travail qui nous fait sentir plus informaticien.

**Lien du dépôt GIT :** <https://gitlab.com/sauceyt/projetqwirkle.git>

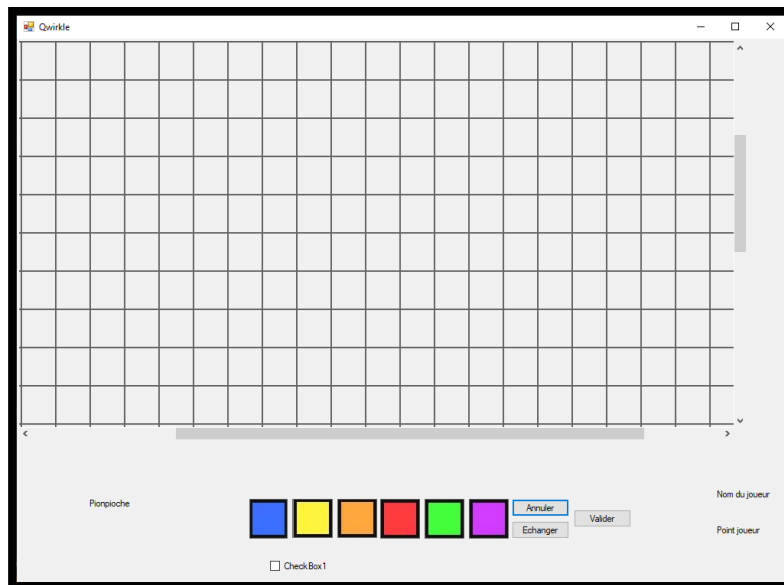
Que ce soit au niveau de l'interface, du moteur et des tests, nous avons tous fait un peu de tout pour mener ce projet à son terme. A ce jour, le jeu fonctionne, les plus grosses fonctionnalités ont été implémentées mais nous confronter à un projet de cette envergure pour la première fois a été l'occasion d'apprendre de nos erreurs notamment dans la mauvaise gestion de notre temps. En effet, nous avons commis certaines erreurs dans les choix les plus stratégiques dès le départ ce qui nous a fait perdre beaucoup de temps notamment pour la classe Plateau.

Malheureusement, certaines fonctionnalités comme le calcul des points ou encore la vérification du placement des pièces et la détermination des coups possibles n'ont pas pu être implémentées dans les temps. C'est une erreur que nous assumons, des algorithmes en pseudo-code ont été réalisés mais nous n'avons pas eu le temps d'implémenter ces fonctionnalités sans devoir impacter grandement les révisions pour nos partiels. Ce n'est pas un problème de compétence, nous aurions pu le faire. Il s'agit d'un problème de gestion du temps qui nous a fait apprendre beaucoup.

## Interface VB.NET

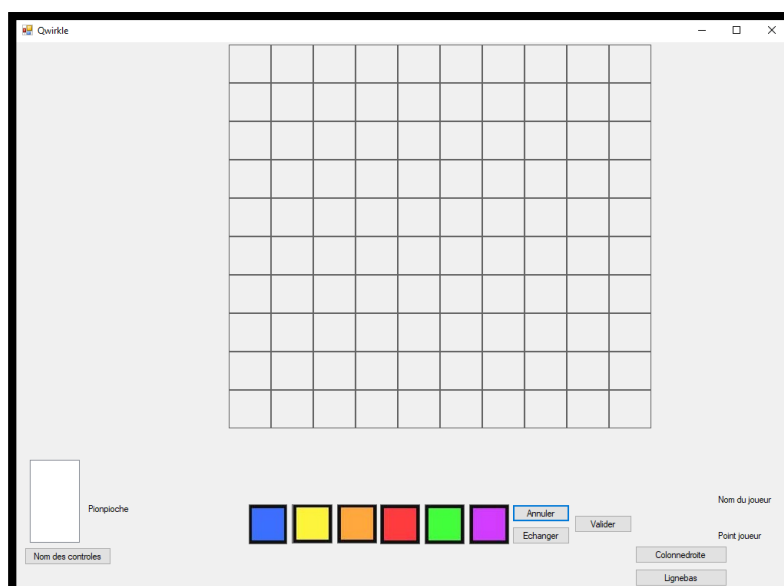
### Présentation

Ces explications ont pour but de comprendre le travail qui a été réalisé par le groupe sur la partie VB. Nous avons réalisé la partie graphique pour le plateau statique de 30\*30 avec un panel dynamique dans un panel statique et l'ajout de PictureBox. Les parties graphiques de paramétrage de notre jeu se sont aussi réalisées rapidement puis nous sommes partis sur notre plateau dynamique pour avoir le temps d'implémenter les fonctionnalités voulues.



La suite a été plus compliquée quand nous avons dû mettre le tout en dynamique. On a donc fait le choix de garder les deux panels et établir un code qui pourrait gérer l'ajout de colonne et de ligne dans les panels.

On a commencé par créer un plateau de 10\*10 dynamiquement et renommer toutes les PictureBox.



On a poursuivi en implémentant nos programmes de gestions de l'extension du plateau dans deux boutons, un pour les lignes et un autre pour les colonnes



A chaque passage de l'événement du clic de la souris on fait un ajout de colonne ou de ligne selon le bouton

## Ajout de colonne

```
Private Sub ButtonDynamique_Click(sender As Object, e As EventArgs) Handles ButtonDynamique.Click
    Dim intervalcolonne = 0
    Dim numerobox = 0
    Dim Nbligne = 1
    Dim nom() As String = Paneldynamique.Controls(Paneldynamique.Controls.Count - 1).Name.Split(CChar("L"))

    For bouclecolonne As Integer = 0 To 0

        For boucleligne As Integer = 0 To 9 + cliquocol
            Dim PictureBox As New PictureBox
            Const taillebox As Integer = 55
            PictureBox.Width = taillebox
            PictureBox.Top = intervalcolonne
            PictureBox.Left = 10 * 55 + 275 + Nbtour
            PictureBox.SizeMode = PictureBoxSizeMode.StretchImage
            PictureBox.BorderStyle = BorderStyle.FixedSingle
            PictureBox.AllowDrop = True
            PictureBox.Name = "C" & 10 + nbcolonnedynamique & "L" & Nbligne
            Nbligne = Nbligne + 1

            AddHandler PictureBox.MouseMove, AddressOf pic_MouseMove
            AddHandler PictureBox.DragDrop, AddressOf pic_DragDrop
            AddHandler PictureBox.DragEnter, AddressOf pic_DragEnter
            AddHandler PictureBox.GiveFeedback, AddressOf pb1_GiveFeedback
            Me.Paneldynamique.Controls.Add(PictureBox)
            intervalcolonne = intervalcolonne + 50
        Next
        intervalcolonne = 0
    Next
    cliquelig = cliquelig + 1
    Nbtour = Nbtour + 55
    nbcolonnedynamique = nbcolonnedynamique + 1
End Sub
```

## Ajout de ligne

```
Private Sub Buttonlignebas_Click(sender As Object, e As EventArgs) Handles Buttonlignebas.Click
    Dim numerobox = 0
    Dim nom() As String = Paneldynamique.Controls(Paneldynamique.Controls.Count - 1).Name.Split(CChar("L"))
    Dim espacement = 0
    Dim nbcolonne = 1

    For bouclecolonne As Integer = 0 To 9 + cliquelig

        For boucleligne As Integer = 0 To 55 - 55 Step 55
            Dim PictureBox As New PictureBox
            Const taillebox As Integer = 55
            PictureBox.Width = taillebox
            PictureBox.Top = 9 * 55 + 5 + espacementbas
            PictureBox.Left = 275 + espacement
            PictureBox.SizeMode = PictureBoxSizeMode.StretchImage
            PictureBox.BorderStyle = BorderStyle.FixedSingle
            PictureBox.AllowDrop = True
            PictureBox.Name = "C" & nbcolonne & "L" & 10 + Nblignedynamique
            nbcolonne = nbcolonne + 1

            AddHandler PictureBox.MouseMove, AddressOf pic_MouseMove
            AddHandler PictureBox.DragDrop, AddressOf pic_DragDrop
            AddHandler PictureBox.DragEnter, AddressOf pic_DragEnter
            AddHandler PictureBox.GiveFeedback, AddressOf pb1_GiveFeedback
            Me.Paneldynamique.Controls.Add(PictureBox)
            espacement = espacement + 55
        Next
        espacementbas = espacementbas + 50
        cliquocol = cliquocol + 1
        Nblignedynamique = Nblignedynamique + 1
    Next
End Sub
```

L'évènement DragEnter s'appuie sur les TP réalisés en cours qui permettent d'autoriser le fait de prendre pour ensuite déplacer une tuile.

```
Private Sub pic_DragEnter(sender As Object, e As DragEventArgs) Handles pic1joueur.DragEnter, pic2joueur.DragEnter, pic3joueur.DragEnter, pic4joueur.DragEnter, pic5joueur.DragEnter, pic6jou
    If e.Data.GetDataPresent(DataFormats.Bitmap) Then
        e.Effect = DragDropEffects.Move
    Else
        e.Effect = DragDropEffects.None
    End If
End Sub
```

L'évènement DragDrop s'inspire de documents réalisés en cours qui servent à autoriser le fait de déposer une tuile aux endroits souhaités.

```
Private Sub pic_DragDrop(sender As Object, e As DragEventArgs) Handles pic1joueur.DragDrop, pic2joueur.DragDrop, pic3joueur.DragDrop, pic4joueur.DragDrop, pic5joueur.DragDrop, pic6joueur.Dr
    Dim pic As PictureBox = sender
    pic.Image = e.Data.GetData(DataFormats.Bitmap)

End Sub
```

Evènement *pic\_MouseMove* sert à déplacer une tuile avec la souris.

```
Private Sub pic_MouseMove(sender As Object, e As MouseEventArgs) Handles pic1joueur.MouseMove, pic2joueur.MouseMove, pic3joueur.MouseMove, pic4joueur.MouseMove, pic5joueur.MouseMove, pic6j
    Dim effetRealise As DragDropEffects
    Dim pic As PictureBox = sender

    If e.Button = MouseButtons.Left AndAlso pic.Image IsNot Nothing Then
        pic.AllowDrop = False
        effetRealise = pic.DoDragDrop(pic.Image, DragDropEffects.Move)

        If effetRealise = DragDropEffects.Move Then
            pic.Image = Nothing
        End If
        pic.AllowDrop = True
    End If
End Sub
```

Form1\_Load sert à créer un tableau de 10\*10 au lancement du jeu et à implémenter une scroll bar verticale et horizontale qui a pour but de se déplacer sur le plateau.

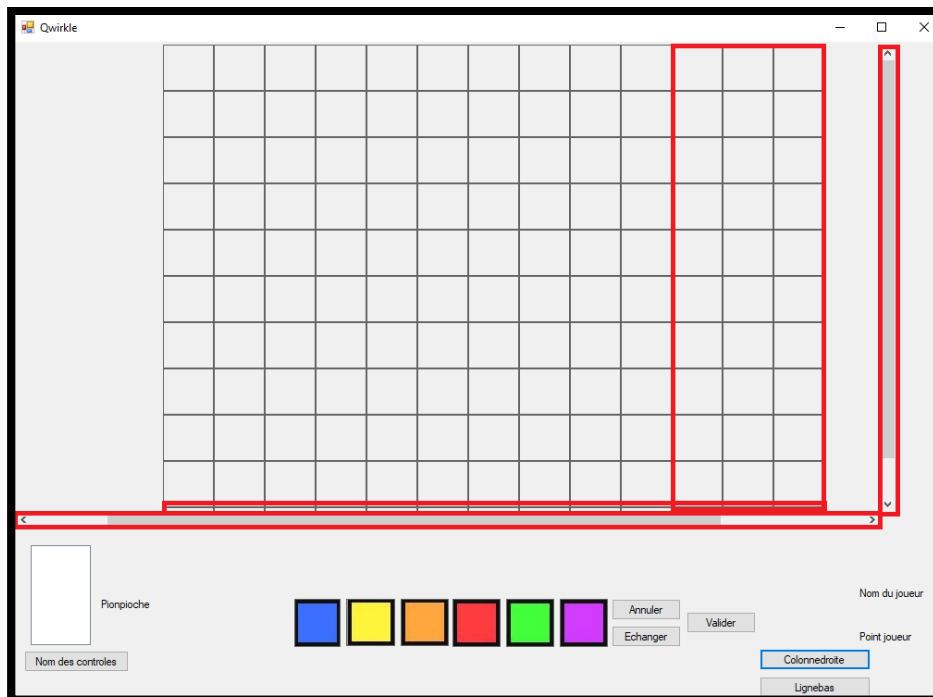
```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    Me.Text = "Qwirkle"
    pic2joueur.AllowDrop = True
    pic3joueur.AllowDrop = True
    pic4joueur.AllowDrop = True
    pic5joueur.AllowDrop = True
    pic6joueur.AllowDrop = True

    Dim intervalcolonne = 0
    Dim numerobox = 0
    Dim Nbligne = 1
    For bouclecolonne As Integer = 0 To 9
        For boucleligne As Integer = 0 To 55 - 55 Step 55
            Dim PictureBox As New PictureBox
            Const taillebox As Integer = 55
            PictureBox.Width = taillebox
            PictureBox.Top = intervalcolonne
            PictureBox.Left = boucleligne + 275
            PictureBox.SizeMode = PictureBoxSizeMode.StretchImage
            PictureBox.BorderStyle = BorderStyle.FixedSingle
            PictureBox.AllowDrop = True
            If numerobox = 10 Then
                PictureBox.Name = "C" & 1 & "L" & Nbligne
                numerobox = 1
            Else
                numerobox = numerobox + 1
                PictureBox.Name = ("C" & numerobox & "L" & Nbligne)
            End If

            AddHandler PictureBox.MouseMove, AddressOf pic_MouseMove
            AddHandler PictureBox.DragDrop, AddressOf pic_DragDrop
            AddHandler PictureBox.DragEnter, AddressOf pic_DragEnter
            AddHandler PictureBox.GiveFeedback, AddressOf pb1_GiveFeedback
            Me.Paneldynamique.Controls.Add(PictureBox)

            Next
            intervalcolonne = intervalcolonne + 50
            Nbligne = Nbligne + 1
        Next
        Panelstatique.AutoScrollPosition = New Point(Panelstatique.Size.Height / 2, (Panelstatique.Size.Width / 2))
    End Sub
```

On peut voir sur cette image l'apparition des deux scroll bars après l'ajout de 3 colonnes et de lignes.



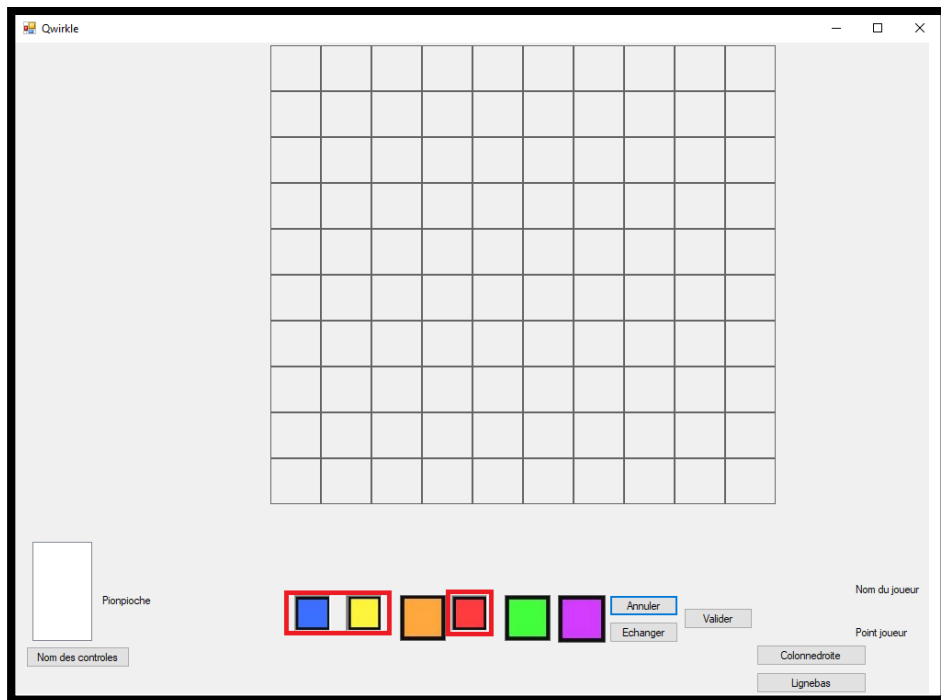
L'événement GiveFeedBack sert à voir la tuile que l'on veut déplacer suivre notre souris tant que le clic est enfoncé.

```
Private Sub pb1_GiveFeedback(ByVal sender As Object, ByVal e As System.Windows.Forms.GiveFeedbackEventArgs) Handles pic1joueur.GiveFeedback, pic2joueur.GiveFeedback, pic3joueur.GiveFeedback,
    e.UseDefaultCursors = False
    Dim myPic As New Bitmap(CType(sender, PictureBox).Image)
    Dim cursorImage
    cursorImage = myPic.GetThumbnailImage(50, 50, Nothing, IntPtr.Zero)
    Cursor.Current = New Cursor(CType(cursorImage, System.Drawing.Bitmap).GetHicon())
End Sub
```

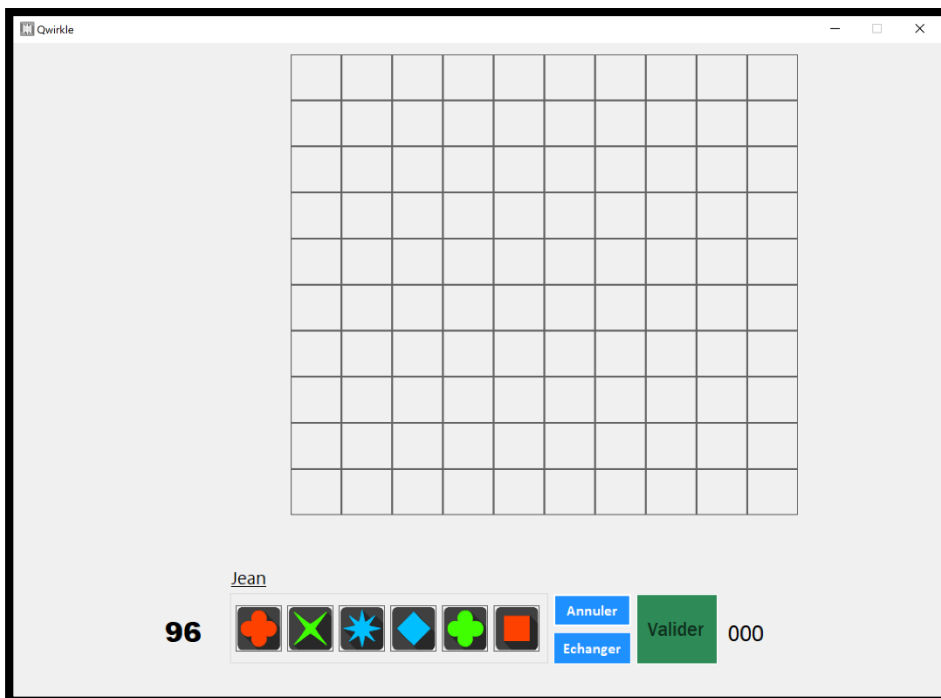
L'évènement du clic sur les PictureBox du joueur sert à afficher les tuiles à échanger en diminuant leur taille.

```
Dim clickevenement = 0
Private Sub picjoueur_Click(sender As Object, e As EventArgs) Handles pic1joueur.Click, pic2joueur.Click, pic3joueur.Click, pic4joueur.Click, pic5joueur.Click, pic6joueur.Click
    If (clickevenement = 0) Then
        sender.BorderStyle = BorderStyle.Fixed3D
        sender.Size = New System.Drawing.Size(40, 40)
        clickevenement = 1
        sender.Location = New Point(sender.Location.X + 5, sender.Location.Y + 5)
    ElseIf (clickevenement = 1) Then
        sender.BorderStyle = BorderStyle.FixedSingle
        sender.Size = New System.Drawing.Size(50, 50)
        clickevenement = 0
        sender.Location = New Point(sender.Location.X - 5, sender.Location.Y - 5)
    End If
End Sub
```

On peut voir sur cet exemple que la tailles des tuiles a diminuée afin de bien faire comprendre au joueur quelles sont les tuiles qu'il a sélectionné pour l'échange.



Voici à la fin à quoi ressemble l'interface de jeu après avoir épuré l'interface et mis en place les fonctions de base du moteur de jeu :



## Ecran de lancement de partie

Qwirkle

**Joueur 1**  
Nom du joueur  
Age du joueur

**Joueur 2**  
Nom du joueur  
Age du joueur

**Joueur 3**  
Nom du joueur  
Age du joueur

**Joueur 4**  
Nom du joueur  
Age du joueur

Annuler    —    +    Démarrer

Cette partie s'est faite sans réels problèmes. On a le choix d'ajouter jusqu'à 4 joueurs mais il faut au minimum deux joueurs. Il a fallu ici travailler sur deux points essentiels :

- **L'ajoute et le retrait des deux joueurs supplémentaires** : A l'aide d'une variable qui s'incrémente et se décrémente en fonction des joueurs ajoutés, il a fallu gérer l'activation et la désactivation des champs et des boutons.
- **Le contrôle de la saisie** : Lors du lancement de la partie, on a dû vérifier si les âges entrés étaient cohérents et surtout numériques et que aucun joueur n'avait le même nom.

## Moteur du jeu C#

### Fonctionnalités implantées

Le moteur de jeu se charge de gérer toute la partie algorithmie qui n'est pas relative à l'interface. Entre autres :

- **La gestion de la pioche** (proportion des tuiles aux caractéristiques similaires et pioche)
- **La gestion des caractéristiques des tuiles** (notamment leur forme et leur couleur)
- **La gestion du plateau** (ajout des pièces, comptage des points, vérification des placements...)
- **La gestion des caractéristiques des joueurs** (nombre, nom, nombre de points, gagnant...)

### Problèmes rencontrés

La gestion de la pioche n'a pas posé de problèmes, il a fallu remplir la pioche de tuiles aux proportions précises dès le démarrage de la partie et gérer avec la fonction *random* le retour d'une pièce aléatoire de la pioche. Le tout s'est fait dans une classe Pioche entièrement statique.

La gestion des caractéristiques des tuiles n'a posé aucun problème non plus. Une classe tuile a été créée pour permettre la gestion des tuiles sous forme d'objets. Cette classe ne gère finalement que les deux attributs d'une tuile soient sa forme et sa couleur. On notera une différence par rapport à l'UML donné

précédemment qui indiquait de nombreuses méthodes dans la classe tuile qui sont finalement superflues car gérées par les autres classes qui s'occupent du plateau de jeu notamment.

La classe joueur est plus grande que les deux classes précédentes mais elle a été mise en place sans réelles difficultés.

La classe plateau a été la plus complexe. Le plateau a dû être géré de manières dynamiques et les tuiles ont dû être gérées de manière particulière et suivre toute une procédure pour les ajouter. Malheureusement, nous avons mal estimé ce temps nécessaire à la mise en place de cette classe et nous avons pris du retard. Finalement, certaines fonctionnalités n'ont pas pu être implémentées comme le comptage des points, la vérification du placement des tuiles et la vérification des possibilités restantes. Des algorithmes en pseudo code avec des tests ont été réalisés et sont disponibles dans le Git, mais ils n'ont pas pu être implémentés.

Pour que le jeu soit néanmoins jouable, nous avons choisi arbitrairement d'ajouter 1 point par tuile posée afin qu'il y ait en fin de jeu un gagnant. Le système vérifie aussi qu'on ne tente pas de placer une pièce sur une autre, c'est le minimum à faire afin d'éviter de faire planter le jeu. Enfin, la fin de la pioche n'a pas pu être correctement gérée à temps et la partie se termine lorsque le nombre de tuiles disponibles dans la pioche atteint 0.

### Classe tuile

```
private string Form; //Star, Cross, Lozenge, Flower, Circle, Square
private string Color; //Yellow, Blue, Pink, Green, Orange, Red
```

C#

Chaque tuile n'aura que deux attributs. Une couleur et une forme définie chaque tuile.

```
// Constructeur
public Tuile(string Form, string Color)
{
    this.Form = Form;
    this.Color = Color;
}

//Méthodes
public string GetForme()
{
    return Form;
}

public string GetCouleur()
{
    return Color;
}
```

C#

Cette classe est assez simple et très classique. Elle n'est composée que d'un constructeur permettant de créer une pièce en spécifiant sa forme et sa couleur et deux accesseurs pour récupérer ces deux caractéristiques. On n'a besoin de rien d'autres, une pièce ne changera jamais au cours de la partie.



## Classe pioche

```
private static int NbPieces = 108;
private static List<Tuile> ListTuiles = new List<Tuile>();
private static Random rand = new Random();
```

C#

Toute la classe est statique. Les seuls attributs sont le nombre de pièces contenues dans la pioche et une collection de tuiles qui sont tout simplement les tuiles présentes dans la pioche. La fonction *random* est initialisée.

```
static Pioche()
{
    List<string> FormsListOrder = new List<string> { "Star", "Cross",
"Lozenge", "Flower", "Circle", "Square" };
    List<string> ColorsListOrder = new List<string> { "Yellow", "Blue",
"Pink", "Green", "Orange", "Red" };
    Tuile[] TabPioche = new Tuile[NbPieces];
    int CreatingNumber = 0;
    foreach (string FormBoucle in FormsListOrder)
    {
        foreach (string ColorBoucle in ColorsListOrder)
        {
            for (int Boucle = 0; Boucle < 3; Boucle++) //3 tuiles de chaque
            {
                TabPioche[CreatingNumber] = new Tuile(FormBoucle, Color-
Boucle);
                ListTuiles.Add(TabPioche[CreatingNumber]);
                CreatingNumber++;
            }
        }
    }
}
```

C#

Cette partie du code s'exécute dès le début du programme. Des objets de type tuile sont massivement créées à partir d'un tableau puis ajoutées à la collection qui représentera la pioche. Comme il faut créer 3 pièces de chaque type, on a opté pour deux boucles imbriquées afin de couvrir toutes les combinaisons possibles avec 3 créations de tuiles à chaque itération.

Ces boucles imbriquées sont particulières car elles parcourront des listes créées à cet effet et prendront directement la valeur de la couleur et de la forme de la pièce à créer.

C#

```

public static Tuile RandomPiece()
{
    int Aleat = rand.Next(NbPieces);
    int EachCompteur = 0;
    foreach (Tuile tuiles in ListTuiles)
    {
        if (Aleat == EachCompteur)
        {
            ListTuiles.RemoveAt(Aleat);
            NbPieces--;
            return tuiles;
        }
        EachCompteur++;
    }
    return null;
}

```

Cette méthode retourne une pièce aléatoire de la pioche. Un nombre aléatoire est choisi et la pioche est ensuite parcourue par une boucle qui va s'arrêter à la pièce choisie aléatoirement. Cette tuile sera supprimée de la pioche et retournée. Si aucune tuile n'est disponible dans la pioche, on retournera une valeur nulle.

### Classe joueur

C#

```

// Attributs
private string Name;
private int Age;
private List<Tuile> Main = new List<Tuile>();
private int Points = 0;
private static int NbJoueurs = 2;
private static int ActualPlayer = 1; //Joueur actuellement en jeu
private static Joueur[] TabPlayers = new Joueur[4];
private static bool[] MainToChange = new bool[6];

// Constructeur

static Joueur()
{
    for (int Boucle = 0; Boucle < 4; Boucle++)
    {
        TabPlayers[Boucle] = new Joueur();
    }
}

```

Cette classe est plus complexe car il y a plus d'attributs et de méthodes. Chaque joueur est défini par son nom et son âge mais aussi par sa main qui est représentée par une liste des tuiles qu'il possède et le nombre de points qu'il a.

Les autres attributs sont plus généraux et sont statiques donc uniques pour tous les joueurs. On retrouve notamment le nombre de joueurs dans la partie, le tour du joueur et enfin un tableau qui contiendra les 4 joueurs de la partie (ou moins en fonction du nombre souhaité). Un dernier tableau servira à l'échange des pions de la main. Chaque pion possédé par le joueur étant représenté dans ce tableau, la valeur de l'indice 3 du tableau passant à vrai signifiera que le pion 2 (car l'indice commence à 0) a été sélectionné par le joueur pour être échangé.

Le constructeur se lance dès le début du programme et se chargera de créer 4 joueurs et de les placer dans le tableau *TabPlayers*. S'il y a moins de 4 joueurs, 4 objets seront tout de même créés mais ils ne seront pas tous utilisés.

Je ne détaillerais pas ici les accesseurs qui servent à accéder aux attributs et à définir le nom et l'âge du joueur, lui ajouter des points, sélectionner et retirer des pièces à échanger et spécifier le nombre de joueurs dans la partie.

```
C#  
  
public void ChangeSelectedPieces()  
{  
    for (int NbMainCompteur = 0; NbMainCompteur < 6; NbMainCompteur++)  
    {  
        if (MainToChange[NbMainCompteur] == true)  
        {  
            ReplacePieceInMain(NbMainCompteur);  
        }  
    }  
    MainToChange = new bool[6];  
}  
  
public void ReplacePieceInMain(int IndexToReplace)  
{  
    if(Pioche.GetNbPieces() != 0)  
    {  
        Main.RemoveAt(IndexToReplace);  
        List<Tuile> TuileToIncrust = new List<Tuile>();  
        TuileToIncrust.Add(Pioche.RandomPiece());  
        Main.InsertRange(IndexToReplace, TuileToIncrust);  
    }  
}
```

Ces deux méthodes permettent l'échange de tuiles dans la main du joueur. Cette fonctionnalité a été découpée en deux méthodes, une qui se charge de cibler les pièces qui ont été sélectionnées pour l'échange, et une autre qui se sert de remplacer chaque pièce dans la main du joueur.

```
C#  
  
public static void DistributionPion()  
{  
    for (int JoueurTirage = 0; JoueurTirage < NbJoueurs; JoueurTirage++)  
    {  
        for (int PiecesTirees = 0; PiecesTirees < 6; PiecesTirees++)  
        {  
            TabPlayers[JoueurTirage].Main.Add(Pioche.RandomPiece());  
        }  
    }  
}
```

Cette méthode permet la distribution des pions. Elle est utilisée dès le démarrage de la partie et utilise la classe *Pioche* pour distribuer aléatoirement 6 tuiles dans la main de chaque joueur.

C#

```

public static void NextPlayer()
{
    ActualPlayer++;
    if (ActualPlayer > NbJoueurs)
    {
        ActualPlayer = 1;
    }
}

public static int SeeWinner()
{
    int NbPlayerWinner = 0;
    int BestScore = TabPlayers[0].Points;
    for (int Boucle = 1; Boucle < NbJoueurs; Boucle++)
    {
        if (TabPlayers[Boucle].Points > BestScore)
        {
            NbPlayerWinner = Boucle;
            BestScore = TabPlayers[Boucle].Points;
        }
    }
    return NbPlayerWinner + 1;
}

```

Ces deux méthodes ne sont pas liées. L'une permet le passage au tour suivant en changeant le numéro de joueur et la deuxième calcul quel joueur a le plus de points pour déterminer le gagnant. On remarquera que cet algorithme n'a pas été conçu pour prendre en compte l'égalité de deux joueurs. En cas d'égalité, ce sera le joueur dont le numéro d'identification est le plus faible qui sera gagnant.

### Classe plateau

C#

```

private static List<Tuile> GrillePiecesContent = new List<Tuile>();
private static List<Tuile> PiecesToAdd = new List<Tuile>();
private static int[,] GrilleIndex = new int[10, 10]; //Il faut retirer 1 à
l'index, car sinon on n'aurait pas pu distinguer une case vide d'une case conte-
nant la première pièce qui est à l'index 0
private static int[,] PosToAdd = new int[6, 2];
private static int NbToAdd = 0;
private static int NbLn = 10;
private static int NbCol = 10;

```

La classe plateau est entièrement statique. Elle contient une collection des pièces déjà placées sur le plateau et une autre des pièces ajoutées par le joueur qui n'ont pas encore été ajoutées. On a ensuite un tableau indiquant l'existence et la position des pièces dans la collection en fonction de ses coordonnées sur le plateau. On a la même chose pour les pièces pas encore ajoutées, leurs coordonnées sont stockées dans un tableau avant d'être ajoutées.

C#

```

public static void ExtendSize()
{
    int[,] TempGrilleIndex = new int[NbLn, NbCol];
    for (int BoucleLn = 0; BoucleLn < NbLn; BoucleLn++)
    {
        for (int BoucleCol = 0; BoucleCol < NbCol; BoucleCol++)
        {
            TempGrilleIndex[BoucleLn, BoucleCol] = GrilleIndex[BoucleLn,
BoucleCol];
        }
    }

    NbCol = NbCol + 2;
    NbLn = NbLn + 2;
    GrilleIndex = new int[NbLn, NbCol];

    for (int BoucleLn = 0; BoucleLn < NbLn - 2; BoucleLn++)
    {
        for (int BoucleCol = 0; BoucleCol < NbCol - 2; BoucleCol++)
        {
            GrilleIndex[BoucleLn + 1, BoucleCol + 1] = TempGrilleIn-
dex[BoucleLn, BoucleCol];
        }
    }

    for (int BoucleToAdd = 0; BoucleToAdd < NbToAdd; BoucleToAdd++)
    {
        PosToAdd[BoucleToAdd, 0]++;
        PosToAdd[BoucleToAdd, 1]++;
    }
}

```

On va aller vite sur la présentation de la classe. Cette méthode permet d'étendre la taille du tableau au niveau du moteur. Le contenu de la grille est copié dans un tableau intermédiaire pour ensuite agrandir le tableau principal en déclarant un nouveau tableau. Les données sont ensuite remises dans le nouveau tableau.

## Tests unitaires

### Classe tuile

C# - Test Unitaire

```

[TestClass]
public class TuilesTests
{
    [TestMethod]
    public void ConstructorTest()
    {
        Tuile MaTuile = new Tuile("Star", "Yellow");
        Assert.AreEqual("Star", MaTuile.GetForme());
        Assert.AreEqual("Yellow", MaTuile.GetCouleur());
    }
}

```

Ce test est l'unique test de la classe Tuile. Il n'y en a pas besoin d'autres puisque cette classe se résume à un constructeur et des accesseurs. On test donc la création d'une tuile avec le constructeur et on vérifie ses attributs. On test du même coup les accesseurs de la classe.

### Classe pioche

C# - Test Unitaire

```
[TestMethod]
public void GetNbPiecesTest()
{
    Pioche.ResetPioche();
    Assert.AreEqual(108, Pioche.GetNbPieces());
}
```

Ce test vérifie le nombre de pièces dans la pioche. En effet, une méthode supplémentaire a été créée pour réinitialiser le contenu de la pioche. Elle sera surtout utilisée dans les tests car la distribution des pions est automatique au démarrage. On s'assure donc que la pioche contient bien le bon nombre de pièces.

Un autre test vérifie si les pièces ont les bonnes proportions. Le principe est d'extraire toutes les pièces de la pioche et de les trier pour s'assurer qu'on obtient bien 3 pièces de chaque type. On vérifie ainsi le système de tirage des tuiles. Ce test étant assez long en nombre de lignes, il n'a pas été exposé ici mais pourra être présenté plus en détail lors de la soutenance.

### Classe joueur

Cette classe contient de nombreux accesseurs permettant l'ajout d'un nom, d'un âge, de pions etc. Ils ont été testés mais le code ne sera pas détaillé ici pour aller à l'essentiel. Ce sont de simples tests qui manipulent les accesseurs pour insérer ou incrémenter une donnée puis de vérifier s'il est possible de l'obtenir.

C# - Test Unitaire

```
[TestMethod]
public void ReplacePieceInMainTest()
{
    string Color = Joueur.GetPlayerInTab(0).GetPieceFromMain(1).GetCou-
leur();
    string Form = Joueur.GetPlayerInTab(0).GetPieceFromMain(1).GetForme();
    do
    {
        Joueur.GetPlayerInTab(0).ReplacePieceInMain(1);
    } while ((Joueur.GetPlayerInTab(0).GetPieceFromMain(1).GetCouleur() !=
Color) || (Joueur.GetPlayerInTab(0).GetPieceFromMain(1).GetForme() != Form));
    //Essaie jusqu'à ce que la pioche soit vide et qu'une erreur se produise
}
```

Ce test vérifie la méthode permettant de remplacer une tuile dans la main du joueur. Le système est simple :

- On mémorise la forme et la couleur d'un pion de la main d'un joueur.
- On remplace ce pion.
- On vérifie que le pion obtenu a des attributs différents du précédent.

Etant donné que des pions du même type circulent en plusieurs exemplaires, une boucle tente plusieurs changements pour s'assurer que ce n'est pas un pion identique qui a été pioché par hasard.

```
[TestMethod]
public void NextPlayerTest()
{
    Assert.AreEqual(1, Joueur.GetActualPlayer());
    Joueur.NextPlayer();
    Assert.AreEqual(2, Joueur.GetActualPlayer());
    Joueur.NextPlayer();
    Assert.AreEqual(3, Joueur.GetActualPlayer());
    Joueur.NextPlayer();
    Assert.AreEqual(4, Joueur.GetActualPlayer());
    Joueur.NextPlayer();
    Assert.AreEqual(1, Joueur.GetActualPlayer());
}

[TestMethod]
public void SeeWinnerTest()
{
    while(Joueur.GetNbJoueur() < 4)
    {
        Joueur.AddNbJoueur();
    }


    Joueur.GetPlayerInTab(0).AddPoint(100);
    Joueur.GetPlayerInTab(1).AddPoint(200);
    Joueur.GetPlayerInTab(2).AddPoint(300);
    Joueur.GetPlayerInTab(3).AddPoint(400);

    Assert.AreEqual(4, Joueur.SeeWinner());


    Joueur.GetPlayerInTab(2).AddPoint(300);
    Assert.AreEqual(3, Joueur.SeeWinner());
}
```

On teste ici deux méthodes. Une qui permet de passer au joueur suivant. On l'utilise plusieurs fois et on vérifie bien que c'est le tour du joueur suivant grâce à l'accessoire. La deuxième indique qui est le gagnant, c'est celui qui a le plus de points. On incrémente donc des points aux joueurs pour vérifier que le gagnant indiqué correspond au joueur avec le plus de points.


## Jeu de test




Joueur 1



Joueur 2





Joueur 3



Joueur 4

Annuler





Démarrer

94

Jeanne

Annuler  
Echanger

Valider 0

93

Lucas

Annuler  
Echanger

Valider 2

92

Jeanne

Annuler  
Echanger

Valider 0

91

Lucas

Annuler  
Echanger

Valider 3

Plusieurs tous plus tard...

74

Lucas

Annuler  
Echanger

Valider 11