Arthur Murillo
Xabi Avellan
TD2 | TP4

M2105 - IHM : Création du jeu de chifoumi

1. *Scénario nominal - 35 mns :*

Titre : Jouer
Résumé : Scénario décrivant le déroulé d'une partie **Acteur** : Utilisateur (acteur principal)
Pré-condition :
Post-condition :
Date de création : **Date de mise à jour** : 24/05 **Version** : 2 .
Créateur : Xabi Avellan, Arthur Murillo

Utilisateur	Système
L'utilisateur **choisit l'une de ses « cartes »**	
	Le système ***affiche la carte du joueur***
	Le système ***joue lui aussi une « carte »***
	Le système ***affiche sa carte***
	Le système ***compare les « cartes »***
	Le système ***détermine qui a gagner la manche***
	Le système ***augmente le nombre de point du vainqueur***
L'utilisateur choisit de **faire une nouvelle partie**	
	Le système ***réinitialise les scores***
	Le système ***supprime les cartes qui étaient affichées***

Note : A partir de ce point, l'utilisateur sera considéré comme **JOUEUR 1 (ou J1)** et la machine comme **JOUEUR 2 (ou J2).**

2. Diagramme de classe (UML) - aspects métier - 1H15 :

Attributs

Type	*Nom attribut*	*Signification*	*Exemple (si necessaire)*
`int`	`scoreJ1`	`Score du joueur 1`	`4`
`int`	`scoreJ2`	`Score du joueur 2`	`1`
`Principale*`	`laVue`	`La vue a laquelle est lié le contrôleur`	
`UnSigne(type énuméré)`	`signeJ2`	`signe joué par le joueur 2`	`Ciseaux`
`UnSigne`	`signeJ1`	`signe joué par le joueur 1`	`Pierre`
`UnEtat(type énuméré)`	`etatSysteme`	`Etat actuel du sytème (initial ou partie en cours)`	`initial`

Méthodes

Déclaration *// But*
`Jeu();` *Constructeur de la classe.*
`~Jeu();` *Destructeur de la classe.*

void initialiserControleur(); ***Met le contrôleur dans son état initial lors de l'instanciation.***
UnEtat getEtatSys(); ***Renvoie l'état actuel du système***
void setEtatSys(UnEtat etat); ***Définit l'état actuel du système***
UnSigne getSigneJ1(); ***Renvoie le signe utilisé par le joueur 1.***
void setSigneJ1(UnSigne signe); ***Définit le signe utilisé par le joueur 1.***
UnSigne getSigneJ2(); ***Renvoie le signe utilisé par le joueur 2.***
void setSigneJ2(UnSigne signe); ***Définit le signe utilisé par le joueur 2.***
void setVue(Principale*); ***Permet de définir la vue liée au contrôleur à l'aide du paramètre qui est un pointeur sur la vue.***
Principale* getVue(); ***Renvoie la vue liée au contrôleur.***
int getScoreJ1(); ***Renvoie le score du joueur 1.***

void setScoreJ1(int score);

Définit le score du joueur 1.

int getScoreJ2();

Renvoie le score du joueur 2.

void setScoreJ2(int score);

Définit le score du joueur 2.

UnSigne determinerSigne (int operande);

Determine le signe (de type UnSigne) correspondant à l'entier passé en paramètre.

void demandeNouvellePartie();

Ordonne a la vue de se mettre a jour en remettant les scores à zero.

void jouer(int signeJoueur);

Traite la demande de jeu pour le paramètre signeJoueur passé par valeur.
Le type int est une valeur arbitraire attribuée a la signe dans la vue et analysée dans le contrôleur. On peut également directement y attribuer une valeur de type UnSigne.
Ce que fait le contrôleur :

- *Détermine le signe passé en paramètre et modifie signeJ1 en conséquence (1 = pierre, 2 = feuille, 3 = ciseaux).*
- *Détermine de manière aléatoire un chiffre entre 1 et 3 et modifie signeJ2 en conséquence.*
- *Compare les deux signes joués et détermine le gagnant.*
- *Actualise les scores*
- *Ordonne à la vue de se mettre à jour*

int genererNombre(int min, int max);

Génère un nombre dans l'intervalle [1-4[en s'appuyant sur la fonction srand

UnSigne genererUnsigne();

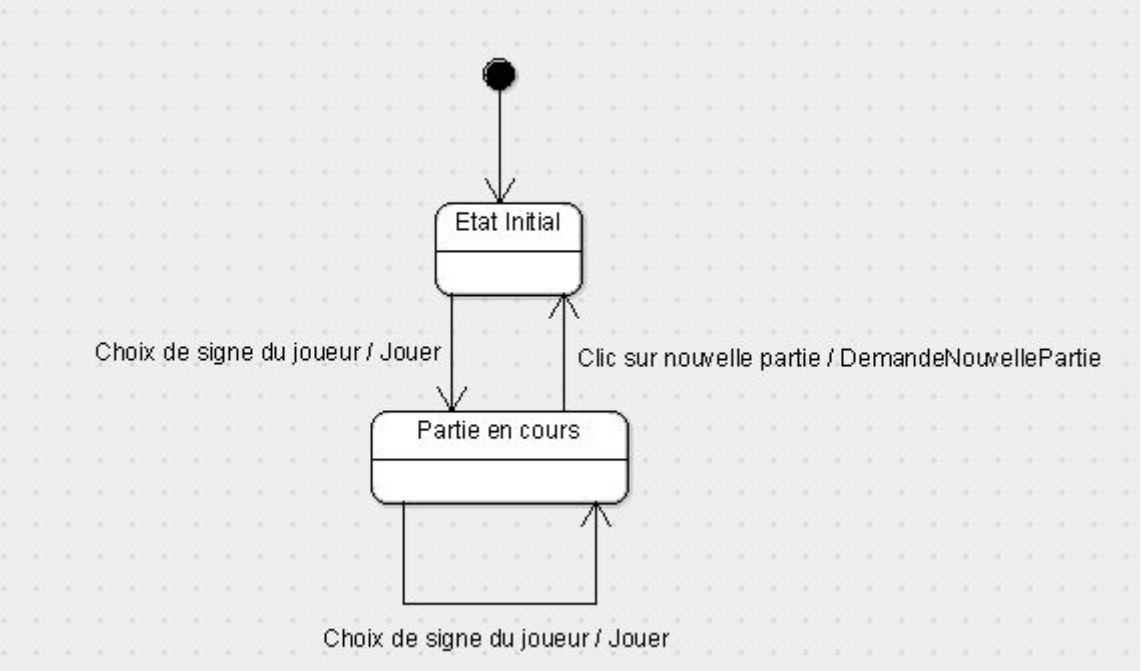
Génère le signe joué par la machine en s'appuyant sur la fonction genererNombre

int determinerLeGagnant(); ***Compare les signes des deux joueurs et détermine le gagnant ou éventuellement l'égalité***
void actualiserScores(int resultat); ***Actualise les scores des joueurs en fonction du résultat retourné par determienrLeGagnant***

Comme on ne retrouve qu'une classe métier dans ce cas, et que son contenu est décrit dans les dictionnaires ci-dessus, la création d'un diagramme de classes UML n'est pas pertinente.

3. Diagramme états-transitions - 1H15 mns :

Diagramme état-transition du système



États du jeu (=du système)

Nom état	*Signification*
`Etat initial`	*L'application attend que l'utilisateur lance la partie.*
`Partie en cours`	*L'utilisateur a lancé la partie en cliquant sur un signe pour jouer. Ensuite, l'utilisateur et la machine jouent tour à tour jusqu'à arrêt de l'application ou lancement d'une nouvelle partie.*

Événements faisant changer le jeu (=système) d'état

Nom événement	*Signification*
`Choix de signe du joueur`	*L'utilisateur lance la partie en cliquant sur un signe pour jouer. La machine joue ensuite.*
`Clic sur nouvelle partie`	*L'utilisateur arrête la partie en cours et en démarre une nouvelle. Le jeu attend alors un clic de sa part sur un signe pour démarrer une nouvelle partie.*

Actions qui accompagnent chaque transition

Nom action	*But*
`Jouer`	***Le système récupère le signe joué par le joueur pour le comparer avec le signe joué par lui même généré aléatoirement. Le système met ensuite les scores à jour en conséquence. Les cartes jouées par le joueur et la machine sont affichées et le système attend une nouvelle action.***
`DemandeNouvellePartie`	***Le système remet les scores à zéro et efface les cartes précédemment jouées. Il attend ensuite une action du joueur pour relancer la partie.***

Version matricielle du diagramme d'état-transitions

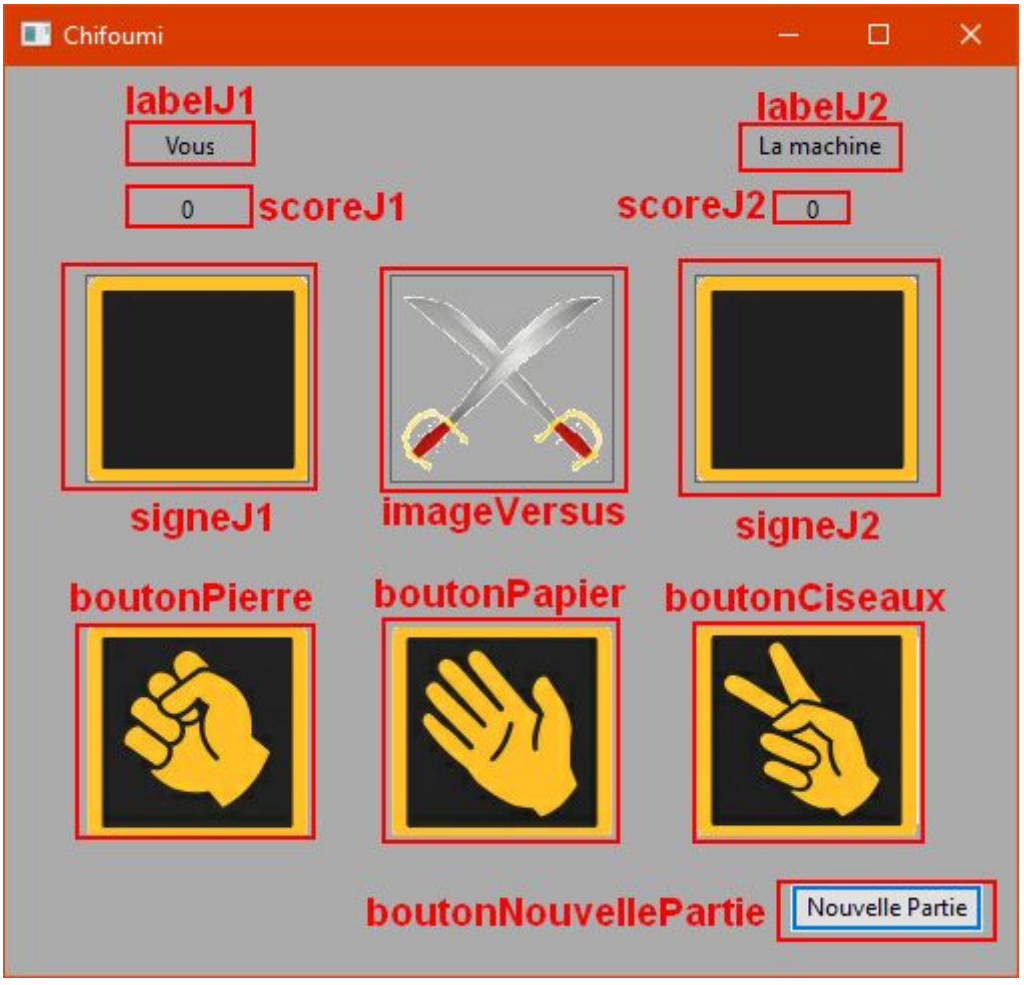
Evénement → / ↓ *nomEtat*	Choix d'un signe	Clic sur nouvelle partie
Etat initial	**Partie en cours**	----
Partie en cours	----	**Etat Initial**

4. *Interface - 2H :*

Version matricielle complétée

Element graphique correspondant	`boutonFeuille` **OU** `boutonPapier` **OU** `boutonCiseaux`	`boutonNouvellePartie`
Evénement → ↓ *nomEtat*	`Choix d'un signe`	`Clic sur nouvelle partie`
`Etat initial`	**Partie en cours**	----
`Partie en cours`	----	**Etat initial**

Maquette de l'interface (<u>sans les sizers</u>)



Etats des éléments graphiques lorsque le système est dans l'état "Etat initial"

Elément d'interface	*Description* *(visible / invisible ; activé / inactif ; focus ; …)*
labelJ1 (wxStaticLabel)	*visible et affiche "Vous"*
labelJ2 (wxStaticLabel)	*visible et affiche "La machine"*
scoreJ1 (wxStaticLabel)	*visible et affiche "0"*
scoreJ2 (wxStaticLabel)	*visible et affiche "0"*
signeJ1 (wxStaticBitMap)	*Visible, n'affiche aucun signe car la partie n'a pas commencé*
signeJ2 (wxStaticBitMap)	*Visible, n'affiche aucun signe car la partie n'a pas commencé*
imageVersus (wxStaticBitMap)	*visible et affiche le logo "versus"*
boutonFeuille (wxBitMapButton)	*Visible, activé et affiche le signe feuille*
boutonPapier (wxBitMapButton)	*Visible, activé et affiche le signe papier*
boutonCiseaux (wxBitMapButton)	*Visible, activé et affiche le signe ciseaux*
boutonNouvellePartie	*Visible et désactivé*

Etats des éléments graphiques lorsque le système est dans l'état "Partie en cours"

Elément d'interface	*Description* *(visible / invisible ; activé / inactif ; focus ; …)*
labelJ1 (wxStaticLabel)	*visible et affiche "Vous"*
labelJ2 (wxStaticLabel)	*visible et affiche "La machine"*
ScoreJ1 (wxStaticLabel)	*visible et affiche le score du joueur 1*
ScoreJ2 (wxStaticLabel)	*visible et affiche le score du joueur 2*
signeJ1 (wxStaticBitMap)	*Visible, et affiche le dernier signe joué par le joueur 1*
signeJ2 (wxStaticBitMap)	*Visible, et affiche le dernier signe joué par le joueur 2*
imageVersus (wxStaticBitMap)	*visible et affiche le logo "versus"*
boutonFeuille (wxBitMapButton)	*Visible, activé et affiche le signe feuille*
boutonPapier (wxBitMapButton)	*Visible, activé et affiche le signe papier*
boutonCiseaux (wxBitMapButton)	*Visible, activé et affiche le signe ciseaux*
boutonNouvellePartie (wxButton)	*Visible et activé*

5. *Organisation du code - MVC - 40 mns :*

Dans la classe Jeu :

```
/*************************************************************
 * Name:      Jeu.h
 * Purpose:   Définit le code du jeu de chifoumi
 * Author:    A. Murillo ()
 * Created:   2019-05-28
 * Copyright: A. Murillo ()
 * License:
 *************************************************************/

#ifndef JEU_H
#define JEU_H

#include "Principale.h"

class Principale;

class Jeu {                                                    Modèle
///*Modele
public:
    ///*Signe des joueurs

protected:

    ///*attributs métier
    //Scores, signes

public:

    ///*méthodes métier
    //Constructeur, destructeur, get et set scores et signes



private:

    ///* Méthodes necessaires au fonctionnement
    //determinerSigne, genererNombre, genererSigne,...
///*Controleur                                                  Contrôleur
public:

    ///*Méhodes du controleur
    //Initialisation, gestion des evenements

    //Gestion du lien entre le modèle et la vue

protected:
    ///*Attributs liés au controleur
};


#endif
```

Dans la classe Principale :

```
/***************************************************************
 * Name:      Principale.h
 * Purpose:   Définit la fenetre principale
 * Author:    A. Murillo ()
 * Created:   2019-05-28
 * Copyright: A. Murillo ()
 * License:
 **************************************************************/

#ifndef PRINCIPALE_H
#define PRINCIPALE_H

#include "jeu.h"
#include <wx/wx.h>

class Jeu;
```

Vue

```
class Principale: public wxFrame
{
public:

    ///* Méthodes et attributs de la vue

private:

    ///*Gestionnaires d'evenements
    //Clic sur nouvelle partie, sur un bouton de signe

    ///* Attributs membres
    //Identifiants, objets graphiques (boutons, labels, bitmap,
    //boutons bitmap, sizers


public:

    ///*Méthodes de mises à jour de la vue
    //afficher et effacer scores, mettre a jour et effacer signes


    ///*Méthodes pour faire le lien avec le controleur


private:

    ///*Controleur (attribut)
};

#endif // PRINCIPALE_H
```

6. *Mise en conformité du code - 1H :*

Jeu.h :

```
/*************************************************************
 * Name:      Jeu.h
 * Purpose:   Définit le code du jeu de chifoumi
 * Author:    A. Murillo ()
 * Created:   2019-05-28
 * Copyright: A. Murillo ()
 * License:
 *************************************************************/

#ifndef JEU_H
#define JEU_H

#include "Principale.h"


class Principale;

class Jeu {

///*Modèle
public:
   ///*Type de carte (pour pouvoir y faire référence dans d'autres classes
   enum UnSigne{pierre = 1, feuille, ciseaux};

protected:

   ///*attributs métier
   int scoreJ1; // Score du joueur 1 (utilisateur)
   int scoreJ2; // Score du joueur 2 (machine)


   UnSigne signeJ1; //Carte jouée par le joueur 1
   UnSigne signeJ2; //Carte jouée par le joueur 2

public:

   ///*méthodes métier
   Jeu(); //Constructeur de la classe
   ~Jeu(); //Destructeur de la classe
```

int getScoreJ1(); **//Renvoie le score du joueur 1**

void setScoreJ1(int score); **//Définit le score du joueur 1**

int getScoreJ2(); **// Renvoie le score du joueur 2**

void setScoreJ2(int score); **//Définit le score du joueur 2**

UnSigne getSigneJ1(); **//Renvoie le signe utilisé par le joueur 1**

void setSigneJ1(UnSigne signe); **// Définit le signe utilisé par le joueur 1**

UnSigne getSigneJ2(); **//Renvoie le signe utilisé par le joueur 2**

void setSigneJ2(UnSigne signe); **// Définit le signe utilisé par le joueur 2**

private:

///*Fonction nécessaires au fonctionnement

UnSigne determinerSigne (int); **//Détermine le signe correspondant à l'entier passé en paramètre**

int genererNombre(int min, int max); **// Genère un nombre dans l'intervalle [1-4[en s'appuyant sur la fonction srand**

UnSigne genererUnsigne(); **//Généré le signe joué par la machine en s'appuyant sur la fonction genererNombre**

int determinerLeGagnant(); **//Compare les signes des deux joueurs et détermine le gagnant ou éventuellement l'égalité**

void actualiserScores(int resultat); **// Actualise les scores des joueurs en fonction du résultat retourné par determienrLeGagnant**

///*Contrôleur
public:

///*Gestion des etats du système
enum UnEtat {initial, partieEnCours};
void setEtatSys (UnEtat etat); **// Définit l'état actuel du système**
UnEtat getEtatSys (); **//Renvoie l'état actuel du système**

///*Méthodes du contrôleur

//Initialisation
void initialiserControleur(); **//Met le contrôleur dans son état initial**

//Gestion des événements

void demandeNouvellePartie(); **//Ordonne à la vue de se mettre à jour en remettant les scores à zéro.**

void jouer(int valeurSigne); /*
Traite la demande de jeu pour le paramètre carteJoueur passé par valeur.
Le type int est une valeur arbitraire attribuée à la carte dans la vue et analysée dans le contrôleur. On peut également directement y attribuer une valeur de type UnSigne.
Ce que fait le contrôleur :

- **Détermine le signe passé en paramètre et modifie signeJ1 en conséquence (1 = pierre, 2 = feuille, 3 = ciseaux).**
- **Détermine de manière aléatoire un chiffre entre 1 et 3 et modifie signeJ2 en conséquence.**
- **Compare les deux signes joués et détermine le gagnant.**
- **Actualise les scores**
- **Ordonne à la vue de se mettre à jour**

***/**

//Gestion du lien entre le modèle et la vue
Principale* getVue(); **//Renvoie la vue liée au contrôleur (pointeur)**
void setVue (Principale*); **//Définit la vue liée au contrôleur**

protected:
///*Attributs liés au contrôleur

Principale* laVue; **//Vue liée au contrôleur**
UnEtat etatSysteme; **//Etat actuel du système**

};

#endif // JEU_H

Principale.h :

```
/*************************************************************
 * Name:      Principale.h
 * Purpose:   Définit la fenetre principale
 * Author:    A. Murillo ()
 * Created:   2019-05-28
 * Copyright: A. Murillo ()
 * License:
 *************************************************************/

#ifndef PRINCIPALE_H
#define PRINCIPALE_H

#include "jeu.h"

#include <wx/wx.h>

class Jeu;

class Principale: public wxFrame
{
public:

    ///* Méthodes et attributs de la vue
    Principale(wxWindow* parent,wxWindowID id = -1);
    virtual ~Principale();

private:

    ///*Gestionnaires d'evenements
    void OnClickboutonNouvellePartie(wxCommandEvent& event);
    void OnClickboutonCiseaux(wxCommandEvent& event);
    void OnClickboutonPapier(wxCommandEvent& event);
    void OnClickboutonPierre(wxCommandEvent& event);


    ///* Attributs membres

    ///*Identifiants
    static const long ID_LABELJ1;
    static const long ID_LABELJ2;
    static const long ID_SCOREJ1;
    static const long ID_SCOREJ2;
    static const long ID_BOUTONCISEAUX;
    static const long ID_IDSIGNE2;
```

```
    static const long ID_SIGNEJ1;
    static const long ID_LOGOVERSUS;
    static const long ID_BOUTONFEUILLE;
    static const long ID_BOUTONPIERRE;
    static const long ID_BUTTONNOUVELLEPARTIE;


    ///*Objets graphiques

    //Objets
    wxBitmapButton* boutonCiseaux;
    wxBitmapButton* boutonPapier;
    wxBitmapButton* boutonPierre;
    wxButton* boutonNouvellePartie;
    wxStaticBitmap* imageVersus;
    wxStaticBitmap* signeJ1;
    wxStaticBitmap* signeJ2;
    wxStaticText* labelJ1;
    wxStaticText* labelJ2;
    wxStaticText* scoreJ1;
    wxStaticText* scoreJ2;


    //Sizers

    wxBoxSizer* topSizer; //Sizer vertical (tout les autres sizers)

    wxBoxSizer* sizerJ1; // Label J1, scoreJ1, signeJ1

    wxBoxSizer* sizerJ2; // Label J2, scoreJ2, signeJ2

    wxBoxSizer* sizerAffichage; //SizerJ1, sizerJ2, image Versus

    wxStaticBoxSizer* sizerSignes; // Boutons pierre, feuille, ciseau


public:

    ///*Méthodes de mises à jour de la vue

    void afficherScores(); //Affiche le score actuel de chaque joueur

    void effacerScores(); //Remet les cores à 0 pour chaque joueur

    void MajSignes(); //Affiche le dernier signe joué par les joueurs
```

```
    void effacerSignes(); //Efface le dernier signe joué par chaque joueur lors du
démarrage d'une nouvelle partie


    ///*Méthodes pour faire le lien avec le controleur
    Jeu* getJeu(); //Renvoie le controleur utilisé
    void setJeu(Jeu*); //Définit le controleur

private:

    ///*Controleur (attribut)
    Jeu* jeu;
};

#endif // PRINCIPALE_H
```

7. *Finalisation de la programmation - 4H30 :*

Maquette de l'interface (avec les sizers)

Note : SizerSignes est un **wxStaticBoxSizer** (légende : "Pour jouer, cliquez sur un des boutons ci-dessous :"). Tous les autres sizers sont des **wxBoxSizer**.

8. Spécifications internes du programme réalisé :

Il n'était pas nécessaire d'établir les spécifications internes du programme car nous avons pu aisément identifier les attributs et méthodes nécessaire au fonctionnement optimal de la classe Principale (la classe Jeu ne possède pas d'attributs ou méthodes supplémentaires à ceux décrits dans la partie 2).

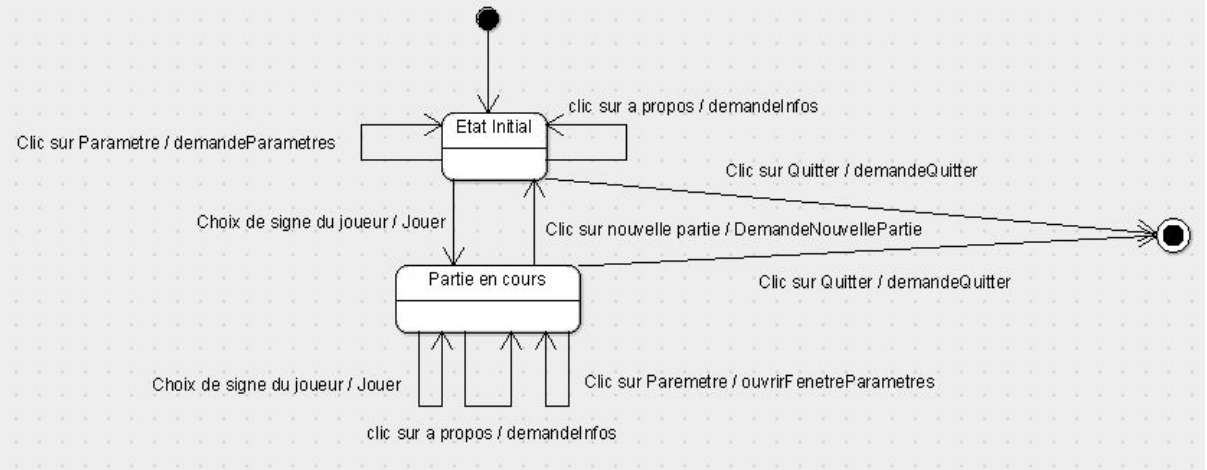
9. *Pour aller plus loin - 5H :*

Nous avons décidé d'entreprendre la mise en place de la version *avecParamètres* du programme.
Pour cela, nous avons repris les éléments suivants :

- Diagramme état transition
- Interface
- Organisation du code - MVC

Diagramme d'état transition :

Nouveau diagramme état-transition du système



Les états du système sont les mêmes que ceux décrits page 6.

Nouveaux événements intervenant sur le jeu (=système)

Nom événement	*Signification*
`Clic sur Parametre`	*L'utilisateur demande à modifier le nom du joueur 1.*
`Clic sur a propos`	*L'utilisateur demande à voir les informations sur le programme et ses auteurs.*
`Clic sur Quitter`	*L'utilisateur demande à quitter le jeu.*

Actions qui accompagnent les nouvelles transitions

Nom action	*But*
`demandeInfo`	***Le système ouvre une boîte de dialogue pour afficher des informations sur le programme et ses auteurs.***
`demandeParamètres`	***Le système ouvre une nouvelle fenêtre (qui est une fenêtre enfant de la fenêtre principale) pour inviter l'utilisateur à saisir le nouveau nom du joueur 1. La partie peut continuer pendant que cette fenêtre est ouverte.***
`demandeQuitter`	***Le système ferme la fenêtre principale.***

Version matricielle du diagramme d'état-transitions

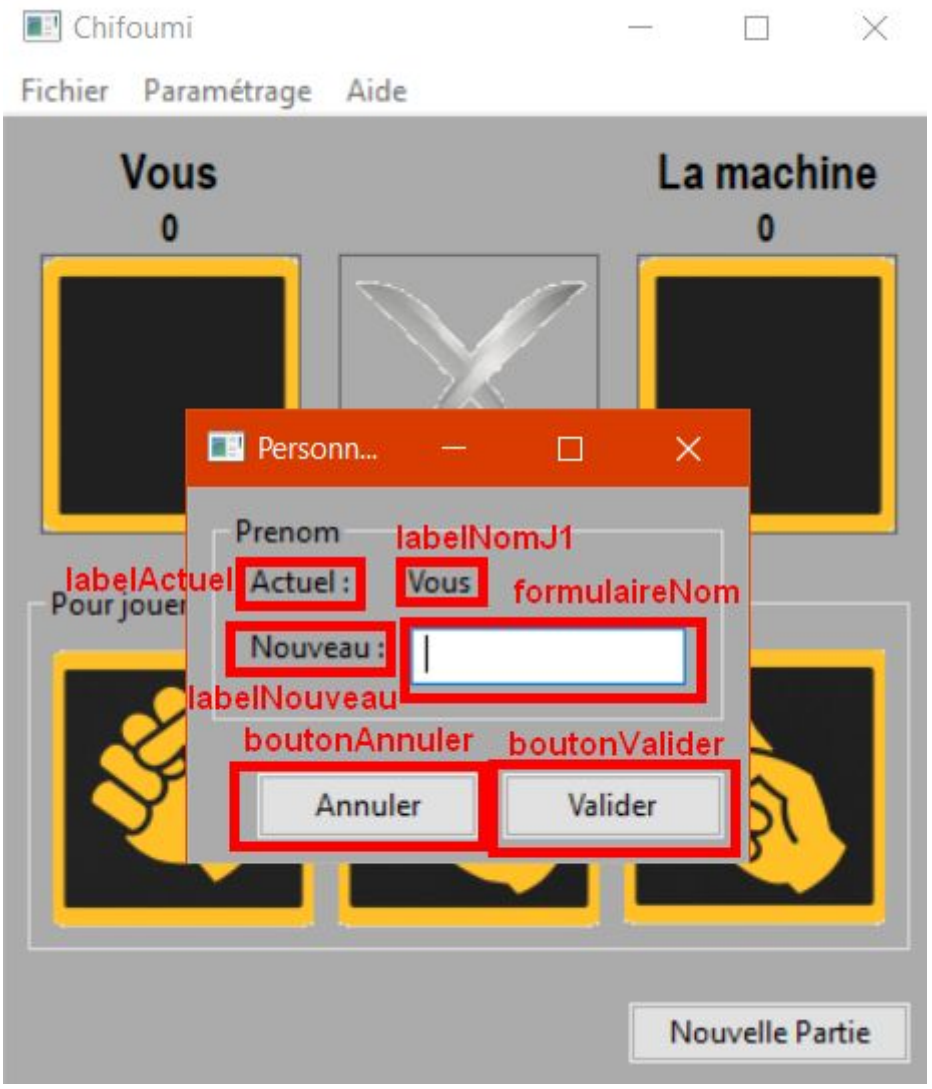
Evénement → / *nomEtat*	Choix d'un signe	Clic sur nouvelle partie	clic sur a propos	clic sur Paramètre	clic sur Qutter
Etat initial	**Partie en cours**	----	----	----	----
Partie en cours	----	**Etat initial**	----	----	----

Interface :

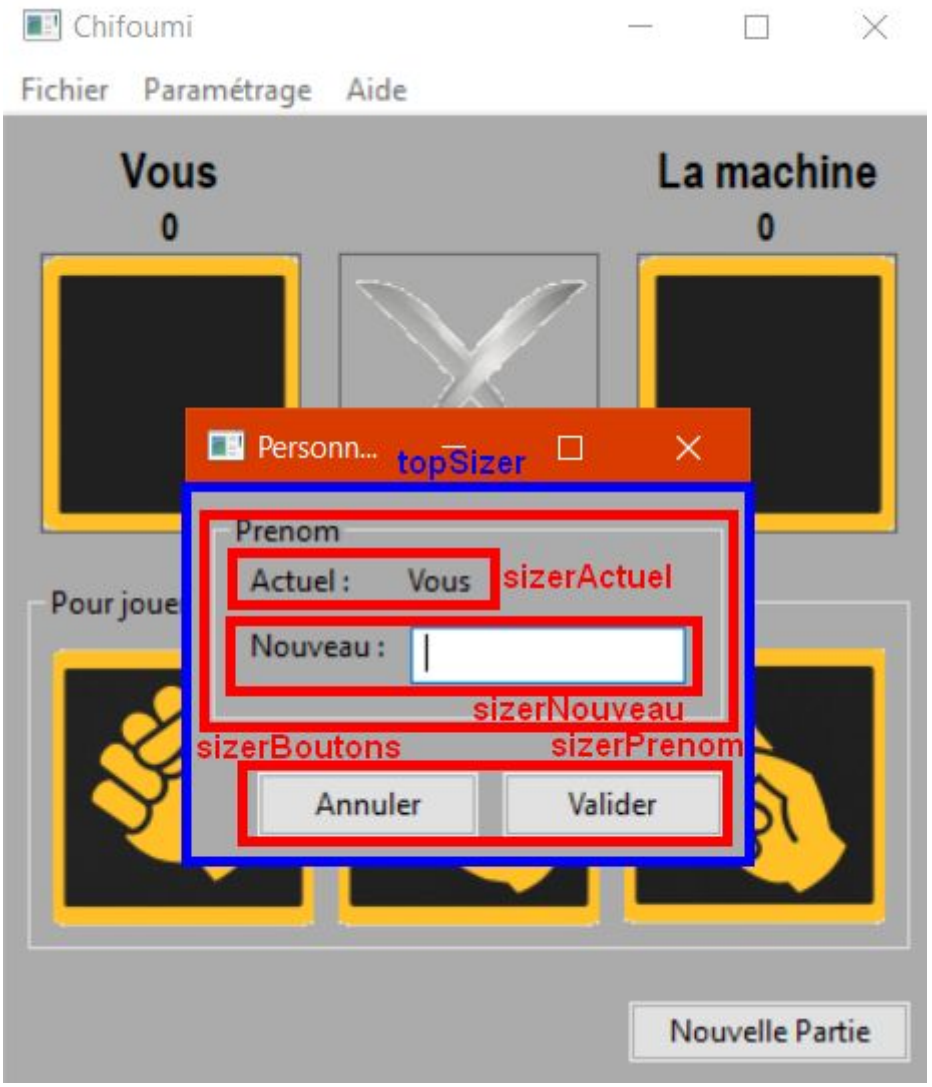
Nouvelle version de la version matricielle de l'état transition complétée :

Element graphique correspondant	boutonFeuille ou boutonPapier ou boutonCiseaux	boutonNouvelle Partie	item de menu "itemAuSujet De"	item de menu "itemJoueur"	item de menu "itemQuitter"
Evénement → / *nomEtat* ↓	Choix d'un signe	Clic sur nouvelle partie	clic sur a propos	clic sur Paramètre	clic sur Qutter
Etat initial	**Partie en cours**	----	----	----	----
Partie en cours	----	**Etat initial**	----	----	----

Nouvelle maquette de l'interface lorsque la fenêtre enfant est ouverte (sans les sizers)



Nouvelle maquette de l'interface lorsque la fenêtre enfant est ouverte (avec les sizers)



Note : sizerPrenom est un **wxStaticBoxSizer** (légende : "Prenom"). Tous les autres sizers sont des **wxBoxSizer**.

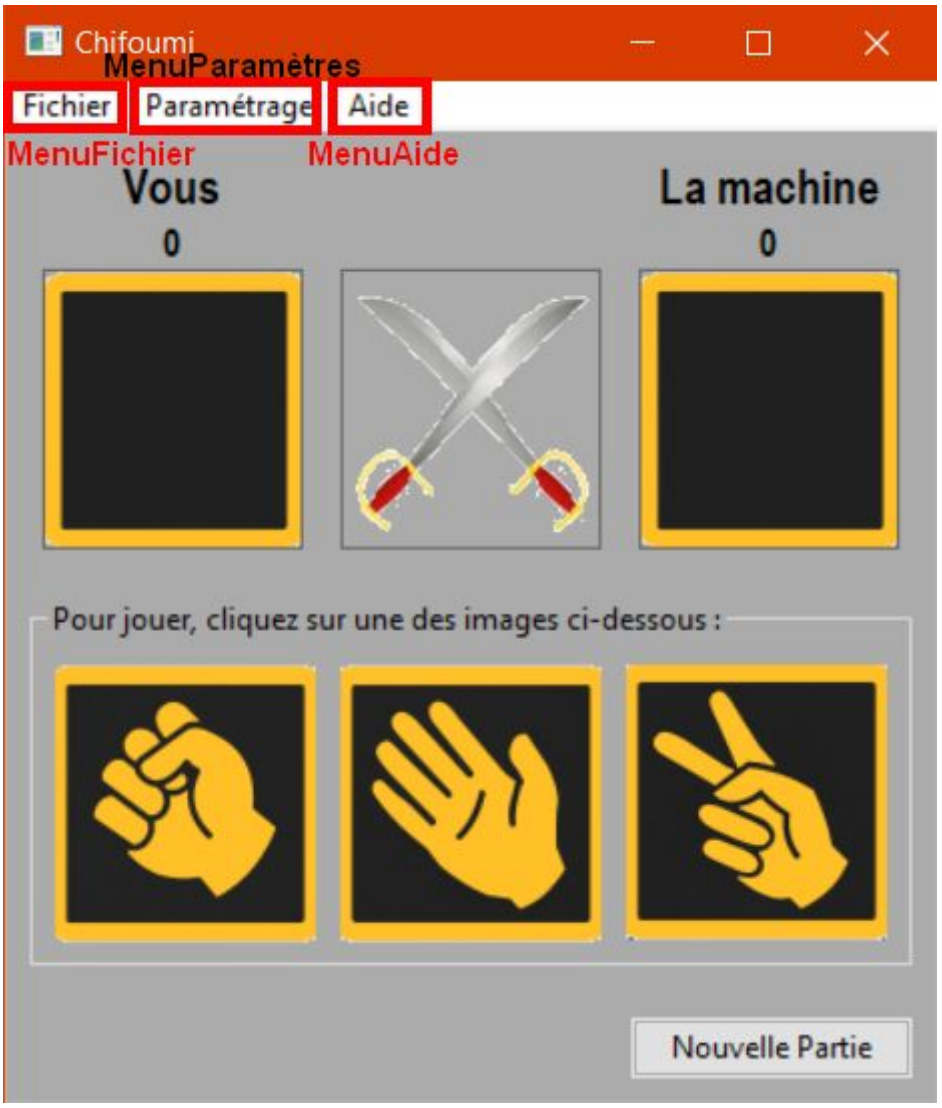
Etats des nouveaux éléments graphiques de la fenêtre enfant lorsque le système est dans l'état "Etat initial"

Elément d'interface	*Description (visible / invisible ; activé / inactif ; focus ; …)*
labelnomJ1 (wxStaticLabel)	*visible et affiche le nom actuel du joueur 1*
labelActuel (wxStaticLabel)	*visible et affiche "Actuel :"*
labelNouveau (wxStaticLabel)	*visible et affiche "Nouveau :"*
formulaireNom (wxTextCtrl)	*visible*
boutonValider (wxButton)	*visible et activé*
boutonAnnuler (wxButton)	*visible et activé*

Etats des nouveaux éléments graphiques de la fenêtre enfant lorsque le système est dans l'état "Partie en cours"

Elément d'interface	*Description (visible / invisible ; activé / inactif ; focus ; …)*
labelnomJ1 (wxStaticLabel)	*visible et affiche le nom actuel du joueur 1*
labelActuel (wxStaticLabel)	*visible et affiche "Actuel :"*
labelNouveau (wxStaticLabel)	*visible et affiche "Nouveau :"*
formulaireNom (wxTextCtrl)	*visible*
boutonValider (wxButton)	*visible et activé*
boutonAnnuler (wxButton)	*visible et activé*

On retrouve sur la fenêtre principale une barre de menu telle que :



La barre de menu est composée de 3 menus (menuFichier, menuParmètres et menuAide) contenant chacun un item, respectivement :

- Quitter (raccourci ALT + F4) : invoque la méthode demandeQuitter.
- Joueur (raccourci ALT + F1) : invoque la méthode demandeParamètre.
- Au sujet de (raccourci F12) : invoque la méthode demandeAide.

Cette barre de menu est visible et activée dans tous les états du jeu.

Organisation du code - MVC :

L'organisation du code selon l'architecture MVC n'a pas évolué pour les classes Jeu et Principale. On y retrouve cependant des méthodes supplémentaire respectivement dans la partie contrôleur (gestion des événements) :

- void demandeInfos();
- void demandeParamètres();
- void demandeAide();

et dans la partie vue (méthodes de mise à jour de la vue) :

- void ouvrirFenetreParametres(); //Instanciation de la fenêtre enfant puis ouverture
- void afficherInfos(); //Affichage des informations sur le programme et ses auteurs

Par contre, on retrouve une classe suplémentaire, de même type que Principale, représentant la fenêtre enfant. Cette classe est une vue, et son code est organisé comme suit dans le fichier .h :

```
/***************************************************************
 * Name:      fenetreJoueur.H
 * Purpose:   Définition de la fenetre enfant fenetreParametres
 * Author:    A. Murillo et X. Avellan ()
 * Created:   2019-05-28
 * Copyright: A. Murillo et X. Avellan ()
 * License:
 **************************************************************/

#ifndef FENETREJOUEUR_H
#define FENETREJOUEUR_H

#include <wx/wx.h>

class Principale;
```

Vue

```
class fenetreJoueur: public wxFrame
{
public:

    ///* Méthodes et attributs de la fenetre
    //Constructeur, destructeur

private:

    ///*Gestionnaires d'evenements
    //Clic sur valider, clic sur annuler

    ///* Attributs membres
    //Identifiants des objets graphiques, objets graphiques, sizers


public:

    ///* Méthode pour faire le lien avec la fenetre principale
    //Set fenetre principale

private:

    ///*Attribut pour faire le lien avec la fenetre principale
};

#endif // FENETREJOUEUR_H
```

Le fichier .h de cette classe se présente donc comme suit :

```
/***************************************************************
 * Name:      fenetreJoueur.H
 * Purpose:   Définition de la fenetre enfant fenetreParametres
 * Author:    A. Murillo et X. Avellan ()
 * Created:   2019-05-28
 * Copyright: A. Murillo et X. Avellan ()
 * License:
 **************************************************************/
#ifndef FENETREJOUEUR_H
#define FENETREJOUEUR_H

#include <wx/wx.h>

class Principale;

class fenetreJoueur: public wxFrame
{
public:
    ///* Méthodes et attributs de la fenetre
    fenetreJoueur(wxWindow* parent,wxWindowID id = -1); //Constructeur
    virtual ~fenetreJoueur(); //Destructeur

private:
    ///*Gestionnaires d'evenements

    void OnClickboutonValider(wxCommandEvent& event);

    void OnClickboutonAnnuler(wxCommandEvent& event);
```

///* Attributs membres

///*Identifiants des objets graphiques

//De la fenêtre

static const long ID_BOUTONVALIDER;

static const long ID_BOUTONANNULER;

static const long ID_FORMULAIRENOM;

static const long ID_LABELACTUEL;

static const long ID_LABELNOMJ1;

static const long ID_LABELNOUVEAU;

///*Objets graphiques

//Objets

wxButton* boutonValider;

wxButton* boutonAnnuler;

wxTextCtrl* formulaireNom; **//Formulaire dans lequel saisir le nouveau nom**

wxStaticText* labelActuel; **//"Actuel :"**

wxStaticText* labelNomJ1; **//Nom actuel du joueur 1**

wxStaticText* labelNouveau; **//"Nouveau : "**

```
 //Sizers
    wxBoxSizer* topSizer; // Sizer Vertical

    wxBoxSizer* sizerActuel; //affichage Ancien nom

    wxBoxSizer* sizerNouveau; // Saisie nouveau nom

    wxBoxSizer* sizerBoutons; // Boutons annuler et valider

    wxStaticBoxSizer* sizerPrenom; // Zone de saisie du label


public:

    ///* Méthode pour faire le lien avec la fenetre principale
    void setFenetrePrincipale(Principale*);

private:

    ///*Attribut pour faire le lien avec la fenetre principale
    Principale* fenetrePrincipale; //Pointeur sur fenetre principale
};

#endif // FENETREJOUEUR_H
```

10. Bibliographie / webographie :

- ***Documentation wxWidget : https://www.wxwidgets.org/***

- ***Elements de correction*** par Pantxika Dagorret (uniquement pour la mise en place de la fonction *genererUnSigne* dans la classe Jeu).

11. Bilan des activités :

Les différents temps passés sur chaque partie sont renseignés dans les titres des parties correspondantes (ceux-ci sont approximatifs).

Les apprentissages ou pratiques que nous retenons de ce projet sont les suivants :

- L'utilisation de générateurs d'interfaces qui rendent plus rapide la conception des interfaces.
- Prendre le temps de bien rédiger le dictionnaire des méthodes pour poser le cadre de la classe et s'assurer de produire quelque chose de fonctionnel (nous avons du reprendre la classe "Jeu" car nous avions codé celle-ci trop vite et pas assez rigoureusement).
- L'utilisation du modèle MVC rend plus simple les modifications drastiques de la vue (nous avons également dû reprendre cette classe plusieurs fois à cause de problèmes de conception et de bugs de wxSmith).
- L'utilisation de la documentation officielle sur internet car celle-ci nous a aidé pour savoir quelles méthodes utiliser tout au long du projet.