

Saé 2.01 – Développement d'une application

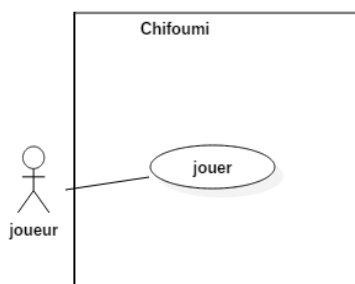
Chifoumi – Dossier d'Analyse et conception

1. Compléments de spécifications externes.

On précise **uniquement** les points qui vous ont semblé flous ou bien incomplets. Rien de plus à signaler dans cette étude.

1.1

2. Diagramme des Cas d'Utilisation



1.2

Figure 1 : Diagramme des Cas d'Utilisation du jeu Chifoumi

3. Scénarios

(a) Exemple Scénario

Cas d'utilisation	JOUER	
Résumé	Le joueur joue une partie.	
Acteur primaire	Joueur	
Système	Chifoumi	
Intervenants		
Niveau	Objectif utilisateur	
Préconditions	Le jeu est démarré et se trouve à l'état initial.	
Postconditions		
Date de création		
Date de mise à jour		
Créateur		
Opérations	Joueur	Système
1	Démarre une nouvelle partie.	
2		Rend les figures actives et les affiche actives.
3	Choisit une figure.	
4		Affiche la figure du joueur dans la zone d'affichage du dernier coup joueur.
5		Choisit une figure.
6		Affiche sa figure dans la zone d'affichage de son dernier coup.
7		Détermine le gagnant et met à jour les scores.
8		Affiche les scores. Retour à l'étape 3.
Extension		
3.A	Le joueur demande à jouer une nouvelle partie.	
3.A.1	Choisit une nouvelle partie	
3.A.2		Réinitialise les scores.
3.A.3		Réinitialise les zones d'affichage des derniers coups.
3.A.4		Retour à l'étape 3.

Tableau 1 :
Scénario
nominal

(b) Remarques :

- *Le scénario est très simple.*
- *L'objectif est de mettre en évidence les actions de l'utilisateur, celles du système, sachant que ces actions sont candidates à devenir des méthodes du système*

1.3

4. Diagramme de classe (UML)

(a) Le diagramme de classes UML du jeu se focalise sur les classes **métier**, cad celles décrivant le jeu indépendamment des éléments d'interface que comportera le programme.

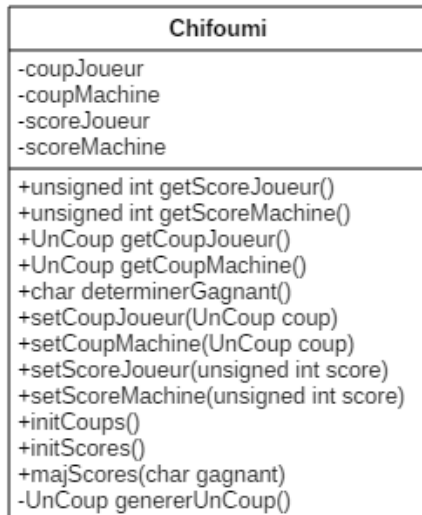


Figure 2 : Diagramme de Classes UML du jeu Chifoumi

(b) Dictionnaire des éléments de la Classe Chifoumi

Nom attribut	Signification	Type	Exemple
scoreJoueur	Nbre total de points acquis par le joueur durant la partie courante	unsigned int	1
scoreMachine	Nbre total de points acquis par la machine durant la partie courante	unsigned int	1
coupJoueur	Mémoire la dernière figure choisie par le joueur. Type énuméré enum unCoup {pierre, ciseau, papier, rien};	UnCoup	papier
coupMachine	Mémoire la dernière figure choisie par la machine.	UnCoup	Ciseau

Tableau 2 : Dictionnaire des éléments - Classe Chifoumi

(c) Dictionnaire des méthodes : intégrées dans l'interface de la classe : cf Figure 3

```
using namespace std;
class Chifoumi
{
    /*** ---- PARTIE MODÈLE -----
        /*** Une définition de type énuméré
    public:
        enum UnCoup {pierre, papier, ciseau, rien};

        /*** Méthodes publiques du Modèle
    public:
        Chifoumi();
        virtual ~Chifoumi();

        // Getters
        UnCoup getCoupJoueur();
        /* retourne le dernier coup joué par le joueur */
        UnCoup getCoupMachine();
        /* retourne le dernier coup joué par le joueur */
        unsigned int getScoreJoueur();
        /* retourne le score du joueur */
        unsigned int getScoreMachine();
        /* retourne le score de la machine */
        char determinerGagnant();
        /* détermine le gagnant 'J' pour joueur, 'M' pour machine, 'N' pour match nul
           en fonction du dernier coup joué par chacun d'eux */

        /*** Méthodes utilitaires du Modèle
    private :
        UnCoup genererUnCoup();
        /* retourne une valeur aléatoire = pierre, papier ou ciseau.
           Utilisée pour faire jouer la machine */

        // Setters
    public:
        void setCoupJoueur(UnCoup p_coup);
        /* initialise l'attribut coupJoueur avec la valeur
           du paramètre p_coup */
        void setCoupMachine(UnCoup p_coup);
        /* initialise l'attribut coupMachine avec la valeur
           du paramètre p_coup */
        void setScoreJoueur(unsigned int p_score);
        /* initialise l'attribut scoreJoueur avec la valeur
           du paramètre p_score */
        void setScoreMachine(unsigned int p_score);
        /* initialise l'attribut coupMachine avec la valeur
           du paramètre p_score */

        // Autres modificateurs
        void majScores(char p_gagnant);
        /* met à jour le score du joueur ou de la machine ou aucun
           en fonction des règles de gestion du jeu */
        void initScores();
        /* initialise à 0 les attributs scoreJoueur et scoreMachine
           NON indispensable */
        void initCoups();
        /* initialise à rien les attributs coupJoueur et coupMachine
           NON indispensable */

        /*** Attributs du Modèle
    private:
        unsigned int scoreJoueur;    // score actuel du joueur
        unsigned int scoreMachine;   // score actuel de la Machine
        UnCoup coupJoueur;           // dernier coup joué par le joueur
        UnCoup coupMachine;          // dernier coup joué par la machine
};
```

Figure 3 : Schéma de classes = Une seule classe Chifoumi

(d) Remarques concernant le schéma de classes

1. On ne s'intéresse qu'aux attributs et méthodes métier. Notamment, on ne met pas, pour l'instant, ce qui relève de l'affichage car ce sont d'autres objets du programme (widgets) qui se chargeront de l'affichage. Par contre, on n'oublie pas les méthodes `getXXX()`, qui permettront aux objets métier de communiquer leur valeur aux objets graphiques pour que ceux-ci s'affichent.
2. On n'a mis ni le constructeur ni le destructeur, pour alléger le schéma.
3. D'autres attributs et méthodes viendront compléter cette vision ANALYTIQUE du jeu. Il s'agira des attributs et méthodes dits DE CONCEPTION nécessaires au développement de l'application.

1.3.1

Version v0

5. Implémentation et tests

5.1 Implémentation

Liste des fichiers de cette version :

- chifoumi.h :

- chifoumi.cpp :

Respectivement spécification et corps de la classe Chifoumi décrite au paragraphe 4.

5.2 Test

Test avec le programme fourni main.cpp

```
appel du constructeur : construction d'un chifoumi : scores a 0, et coupsJoueurs a RIEN'

teste les methodes get() associees aux attributs 'score'
score Joueur : 0      score Machine : 0

teste les methodes get() associees aux attributs 'coup'
coup Joueur : rien    coup Machine : rien

teste les methodes set() associees aux attributs 'score'
score Joueur : 1      score Machine : 2

teste initScores()
score Joueur : 0      score Machine : 0

teste les methodes set() et get() associees aux attributs 'coup'/'choix'
coup Joueur : pierre  coup Machine : ciseau

quelques tours de jeu pour tester l'identification du gagnant et la maj des scores
coup Joueur : papier  coup Machine : pierre
score Joueur : 1      score Machine : 0

Quitter ? (o/n) n

coup Joueur : papier  coup Machine : papier
score Joueur : 1      score Machine : 0

Quitter ? (o/n)
coup Joueur : ciseau  coup Machine : pierre
score Joueur : 1      score Machine : 1

Quitter ? (o/n) █
```

Le programme par défaut affiche cela jusqu'à la première demande « Quitter ? (o/n) »

Il y aura une série de test c'est-à-dire que les score de chaque participant doivent être à 0 au début et le coup joué de chacun doit être « rien ».

Ensuite, on test la méthode initScore qui remet à 0 les scores.

Puis, on teste les sets et les gets pour mettre des coups ou les récupérer. On a donné le coup « pierre » au coup joueur et « ciseau » au coup machine.

Par la suite, on teste le jeu final en donnant des coups aléatoires à chacun pour voir si le changement du score équivaut bien aux coups affichés. (C'est le test que nous allons faire afin d'obtenir un gagnant en 3 points pour voir le bon fonctionnement).

Enfin on peut soit quitter avec la touche « o » ou alors continuer de tester avec « n ».

Si les premiers tests ne sont pas bons ou alors que le score ne correspond pas avec les coups joués alors le programme ne fonctionne pas correctement.

6. Classe Chifoumi : Diagramme états-transitions

(a) Diagramme états-transitions -actions du jeu

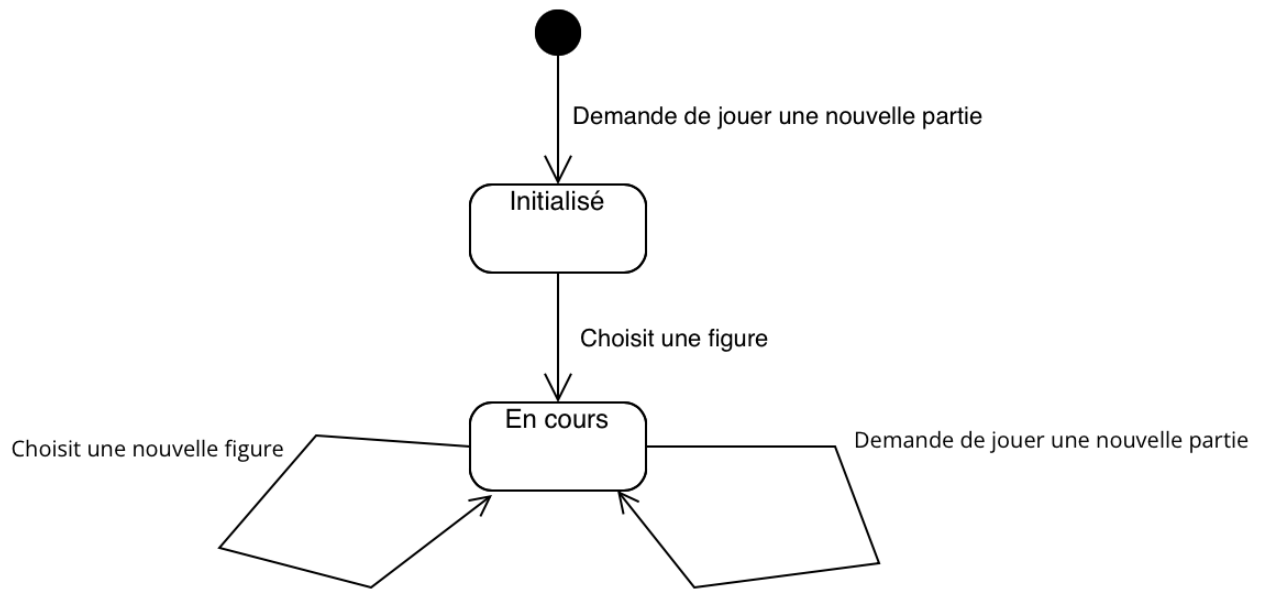


Figure 4 : Diagramme états-transitions

(b) Dictionnaires des états, événements et Actions

Dictionnaire des états du jeu

<i>nomEtat</i>	<i>Signification</i>
Initialisé	Le chifoumi est créé et initialisé : Le score joueur et machine sont mis à 0, les nombres de coup sont mis à 0.
En cours	Le chifoumi est en cours de jeu : Le joueur choisit une figure, par la suite, la machine choisit aléatoirement une figure, sans découle la mise à jour du score et des coups.

Tableau 3 : États du jeu

Dictionnaire des événements faisant changer le jeu d'état

<i>nomÉvénement</i>	<i>Signification</i>
Choisit une figure	Le joueur choisit une figure : Pierre, Feuille, Ciseau, la partie reste donc en cours.
Demande de jouer une nouvelle partie	Le joueur clic sur le bouton Nouvelle Partie, l'état est donc mis en état initiale.

Tableau 4 : Événements faisant changer le jeu d'état

Description des actions réalisées lors de la traversée des transitions

Le joueur joue	Lorsque la transition « choisit une figure » est lancée, le joueur choisit sa figure.
La Machine joue	Une fois que le joueur a fini de choisir sa figure, la machine choisit aléatoirement sa figure et l'affiche.
Le jeu est réinitialisé	Lors de la demande de jouer une nouvelle partie, le chifoumi est remis à 0.

Tableau 5 : Actions à réaliser lors des changements d'état

(c) Préparation au codage :

Table T_EtatsEvenementsJeu correspondant à la version matricielle du diagramme états-transitions du jeu :

- en ligne : les *événements* faisant changer le jeu d'état
- en colonne : les *états* du jeu

<i>Événement →</i> <i>nomEtatJeu</i>	coupJoueurJoué	nvlePartieDemandée
Initialisé	En cours	x
En cours	x	Initialisé

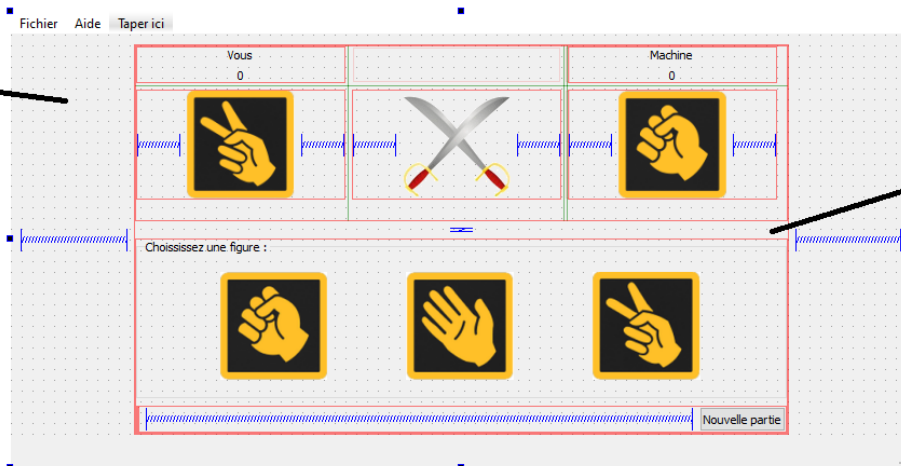
Tableau 6 : Matrice d'états-transitions du jeu chifoumi

L'intérêt de cette vue matricielle est qu'elle permet une préparation naturelle et aisée de l'étape suivante de programmation.

7. Éléments d'interface

7.1 Sizers globaux

nom : centralWidget
type : layout horizontal
Description : layout parent de l'application. Elle est en horizontal pour pouvoir mettre des "spacers" pour gérer le redimensionnement de la fenêtre



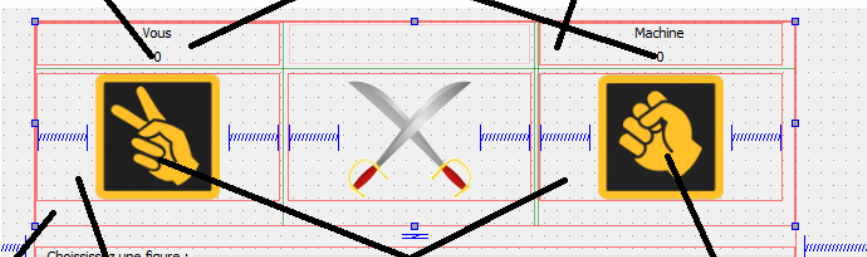
nom : layoutSecondaire
type : layout vertical
Description : layout qui va accueillir les deux parties de l'app, la partie haute (scores et coups) et la partie basse (boutons)

7.2 Éléments du haut de l'écran

nom : labelScoreJoueur/Machine
type : QLabel

nom : layoutScoreJoueur/Machine
type : layout vertical

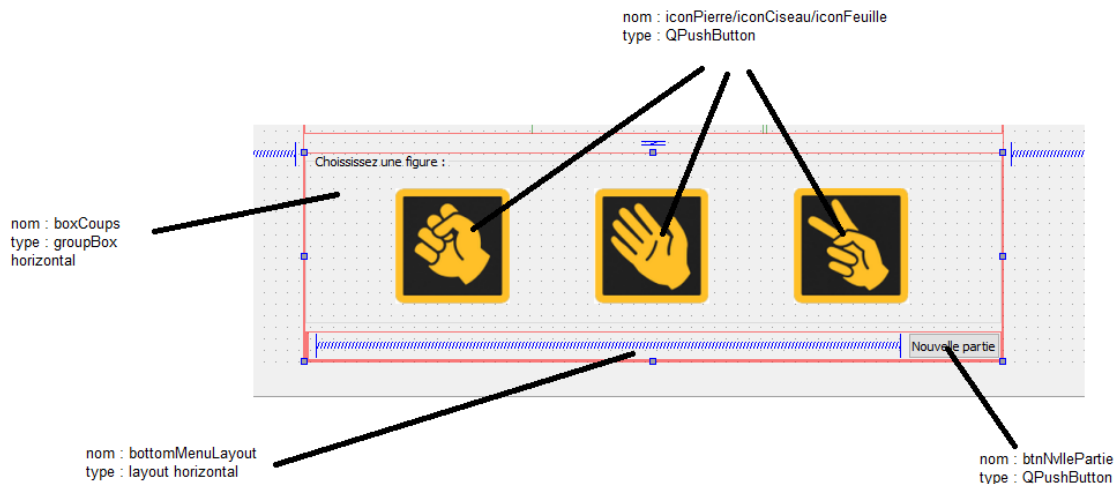
nom : topLayout
type : layout grille



nom : layoutCoupsJoueur/Machine
type : layout horizontal

nom : iconJoueur/Machine
type : QLabel

7.3 Éléments du bas de l'écran



8. Implémentation et tests

8.1 Implémentation

Dans le répertoire VI_SAE-2-01 il y a les fichiers

- *chifoumi.h* ()
- *chifoumi.cpp* ()
- *main.cpp* (Le programme principal qui affiche la fenêtre)

8.2 Test

Tout d'abord nous allons devoir faire 6 tests :

- Voir le changement d'état des boutons des coups lors du clic sur « nouvelle partie »
- Tester le bouton Pierre afin qu'il affiche bien le coup dans la partie du joueur et que le score se met bien à jour pour le gagnant lorsqu'on joue la pierre.
- Tester le bouton Feuille afin qu'il affiche bien le coup dans la partie du joueur et que le score se met bien à jour pour le gagnant lorsqu'on joue la feuille.
- Tester le bouton Ciseau afin qu'il affiche bien le coup dans la partie du joueur et que le score se met bien à jour pour le gagnant lorsqu'on joue le ciseau.
- Cliquer sur le bouton nouvelle partie en cours de partie pour voir si les scores sont remis à 0 et que les zones d'affichage des derniers coups joués soient effacées.
- Tester le redimensionnement de la fenêtre.

9. Classe Chifoumi : Diagramme états-transitions

(d) Diagramme états-transitions -actions du jeu

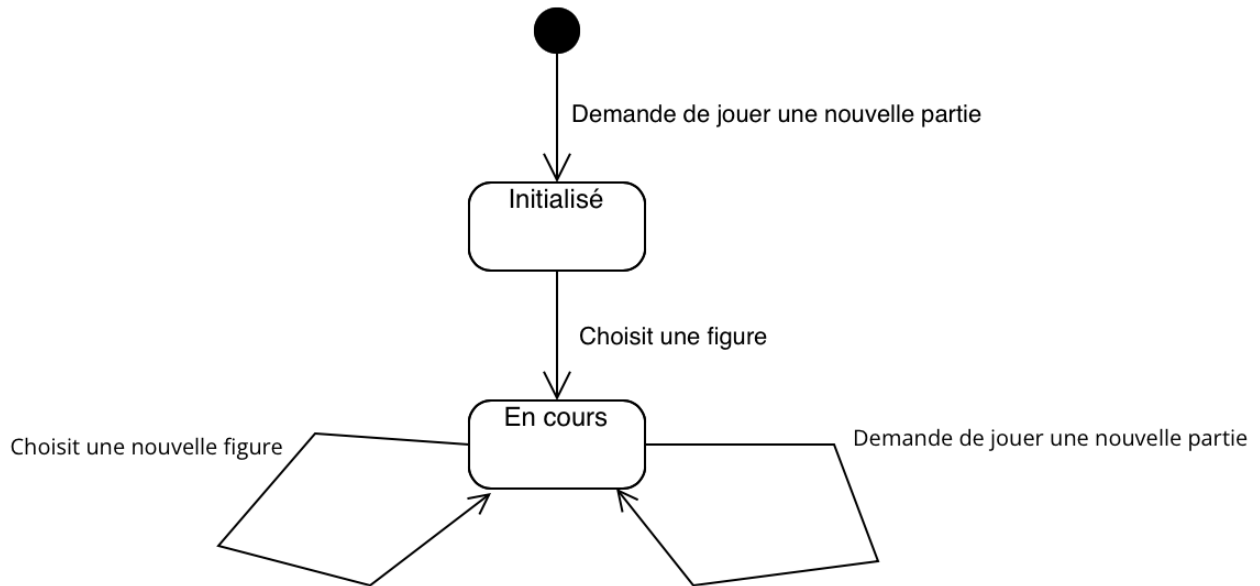


Figure 5 : Diagramme états-transitions

10. Éléments d'interface

Dans le répertoire V1_SAE-2-01 il y a les fichiers :

- presentation.cpp* : Fait le lien entre le modèle et la vue.
- modele.cpp* : C'est le modèle, il contient toutes les primitives du jeu.
- chifoumi.cpp* : c'est la vue, elle sert à effectuer des actions graphiques par rapport au model
- main.cpp* : Lance l'application à l'exécution.

11. Implémentation et tests

8.1 Implémentation

Présentation des .h :

modele.h

```
#ifndef MODELE_H
#define MODELE_H

#include <QObject>

class modele : public QObject
{
    Q_OBJECT
public:
    explicit modele(QObject *parent = nullptr);

public:
    enum UnCoup {pierre, papier, ciseau, rien};
    enum Etat {initialiser, enCours};

    /// Méthodes du Modèle
public:
    /// Getters
    UnCoup getCoupJoueur();
    /* retourne le dernier coup joué par le joueur */
    UnCoup getCoupMachine();
    /* retourne le dernier coup joué par le joueur */
    unsigned int getScoreJoueur();
    /* retourne le score du joueur */
    unsigned int getScoreMachine();
    /* retourne le score de la machine */
    Etat getEtat();
    /* retourne l'état du jeu */

    char determinerGagnant();
    /* détermine le gagnant 'J' pour joueur, 'M' pour machine, 'N' pour match nul
    en fonction du dernier coup joué par chacun d'eux */

public:
    /// Setters
    void setCoupJoueur(UnCoup p_coup);
    /* initialise l'attribut coupJoueur avec la valeur
    du paramètre p_coup */
    void setCoupMachine(UnCoup p_coup);
    /* initialise l'attribut coupMachine avec la valeur
    du paramètre p_coup */
    void setScoreJoueur(unsigned int p_score);
    /* initialise l'attribut scoreJoueur avec la valeur
    du paramètre p_score */
    void setScoreMachine(unsigned int p_score);
    /* initialise l'attribut scoreMachine avec la valeur
```

```

// Autres modificateurs
void majScores(char p_gagnant);
    /* Mise à jour des scores en fonction des règles de gestion actuelles :
       - 1 point pour le gagnant lorsqu'il y a un gagnant
       - 0 point en cas de match nul
    */

void initScores();
    /* initialise à 0 les attributs scoreJoueur et scoreMachine
       NON indispensable */
void initCoups();
    /* initialise à rien les attributs coupJoueur et coupMachine
       NON indispensable */

/** Méthodes utilitaires du Modèle
UnCoup genererUnCoup();
/* retourne une valeur aléatoire = pierre, papier ou ciseau.
   Utilisée pour faire jouer la machine */

private:
    unsigned int scoreJoueur;    // score actuel du joueur
    unsigned int scoreMachine;  // score actuel de la Machine
    UnCoup coupJoueur;          // dernier coup joué par le joueur
    UnCoup coupMachine;         // dernier coup joué par la machine
    Etat etatJeu;               // état de la partie
};

#endif // MODELE_H

```

Dans le modèle, on ne change pas grand-chose, on garde juste les primitives du chifoumi sans faire allusion au graphique, c'est une classe qui peut être exécuté dans la console.

```
#ifndef CHIFOUMI_H
#define CHIFOUMI_H

#include <QChifoumi>
#include "modele.h"

QT_BEGIN_NAMESPACE
namespace Ui { class Chifoumi; }
QT_END_NAMESPACE

class Chifoumi : public QChifoumi
{
    Q_OBJECT
public:
    Chifoumi(QWidget *parent = nullptr);
    ~Chifoumi();

    // pour créer une connexion avec la présentation
    void nvllConnexion(QObject *c);
    void supprConnexion(QObject *c);

    void actualisation(modele::UnCoup, modele::UnCoup, int scoreJ, int scoreM, modele::Etat);

private:
    Ui::Chifoumi *ui;
};
#endif // CHIFOUMI_H
```

Ici on aura 3 méthodes utiles au modèle MVP :

- nvllConnexion : Permet à la classe de connecter tous les boutons à la présentation (Objet c)
- supprConnexion : Permet de supprimer les connexions
- actualisation : Actualise l'UI en fonction de l'état du jeu

presentation.h :

```
#ifndef PRESENTATION_H
#define PRESENTATION_H

#include <QObject>
#include "modele.h"

class MainWindow;
class presentation : public QObject
{
    Q_OBJECT
public:
    explicit presentation(modele* m, QObject *parent = nullptr);

    modele* _leModele;
    MainWindow* _laVue;

public:
    modele* getModele();
    MainWindow* getVue();
    void setModele(modele *m);
    void setVue(MainWindow *m);

private:
    void coupJoueurJoue();

public slots:
    void boutonFeuille();
    void boutonCiseau();
    void boutonPierre();
    void nvlePartieDemandee();
};

#endif // PRESENTATION_H
```

La présentation fait office de lien entre la vue et le modele donc on a besoin de les définir à l'aide de setModele et setVue. Comme nous faisons de set nous devons aussi y accéder avec des get : getModel et getVue.

Ensuite les slots auxquelles nous avons connecter nos boutons dans la vue :

- boutonFeuille (pour le bouton Feuille)
- boutonCiseau (pour le bouton Ciseau)

- boutonPierre (pour le bouton Pierre)
- nvlePartieDemandee (pour le bouton nouvelle partie)

8.2 Test

Le test sera d'avoir le même fonctionnement que la version v1 mais avec une organisation de fichier différente. Les tests seront donc identiques à ceux de la v1.