

Saé 2.01 – Développement d'une application

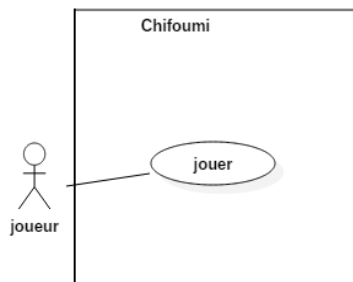
Chifoumi – Dossier d'Analyse et conception

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.

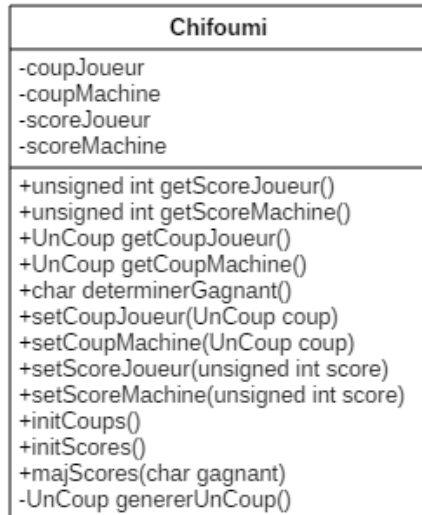


Figure 2 : Diagramme de Classes UML du jeu Chifoumi

(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.

Version v1

6. Classe Chifoumi : Diagramme états-transitions

1. Diagramme états-transitions -actions du jeu

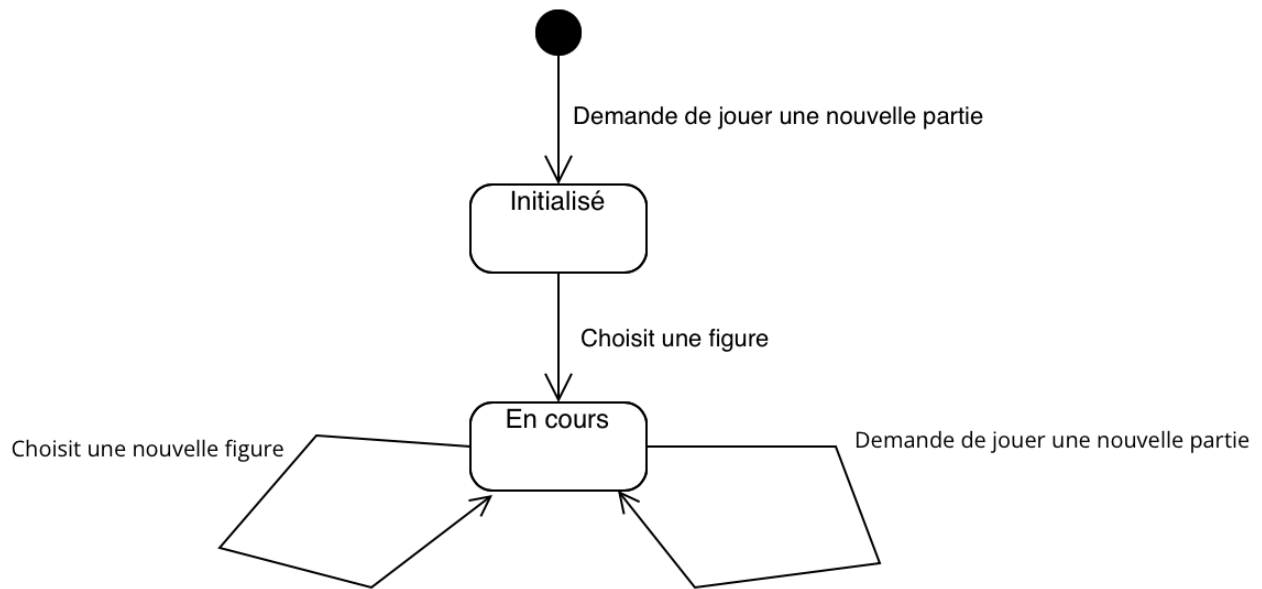


Figure 4 : Diagramme états-transitions

2. 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 découler 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

3. 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

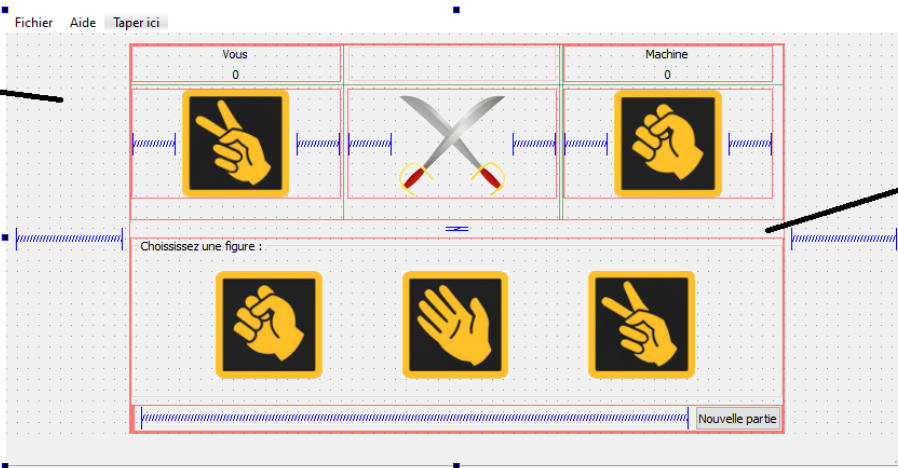
Tableau 6 : Matrice d'états-transitions du jeu chifoumi

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



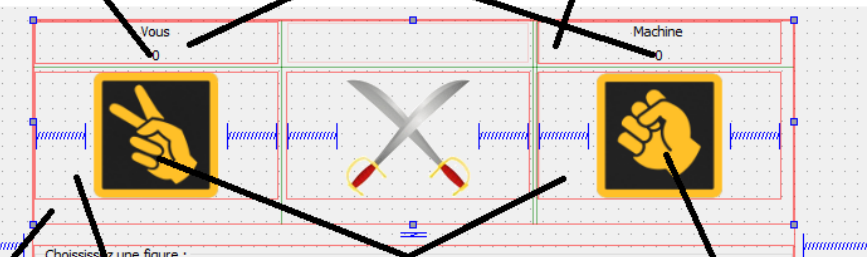
nom : layoutSecondaire
type : layout vertical
Description : layout qui va accueillir les deux parties de l'app, la partie haute (scores et coups) et la partie basse (boutons)

7.2 Éléments du haut de l'écran

nom : labelScoreJoueur/Machine
type : QLabel

nom : layoutScoreJoueur/Machine
type : layout vertical

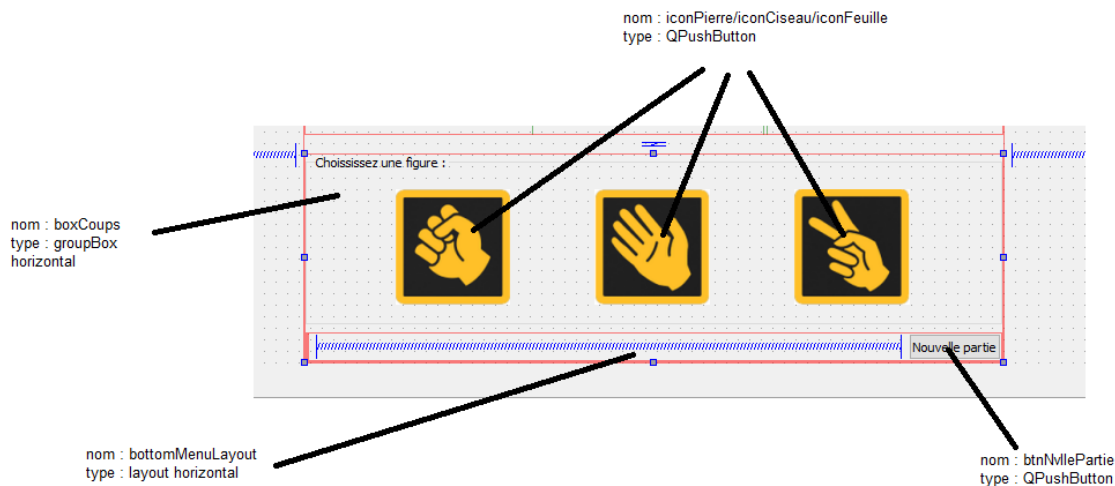
nom : topLayout
type : layout grille



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.

Version v2

9. Classe Chifoumi : Diagramme états-transitions

4. Diagramme états-transitions -actions du jeu

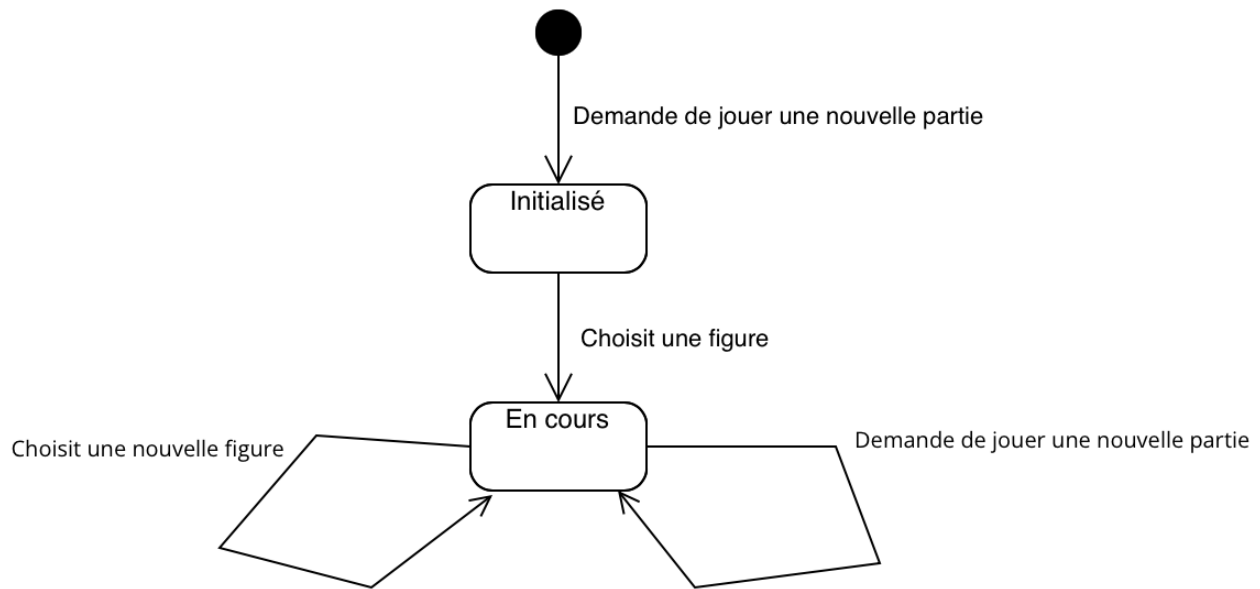


Figure 5 : Diagramme états-transitions

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 chiffoumi sans faire allusion au graphique, c'est une classe qui peut être exécuté dans la console.

chifoumi.h (la vue)

```
#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 nvllleConnexion(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 utile au modèle MVP :

- nvllleConnexion : 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 nvllePartieDemandee();
};

#endif // PRESENTATION_H
```

La présentation fait office de lien entre la vue et le modele donc on à 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.

Version v3

12. Classe Chifoumi : Diagramme états-transitions

5. Diagramme états-transitions -actions du jeu

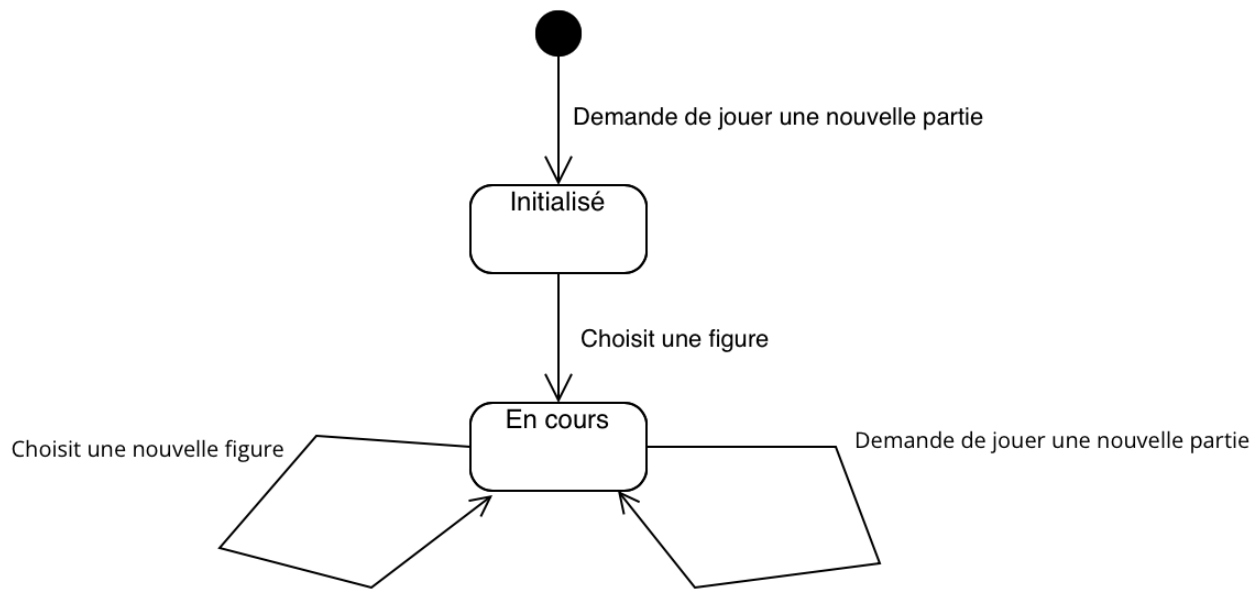


Figure 6 : Diagramme états-transitions

13.Éléments d'interface

Dans le répertoire VI_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.

14.Implémentation et tests

8.1 Implémentation

Fichier .h modifié :

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 nvllePartieDemandee();
    void aProposDe();
};

#endif // PRESENTATION_H

```

On a simplement ajouté dans la présentation un slot : aProposDe.

Ce slot va simplement ouvrir une messageBox quand elle est appelée.

8.2 Test

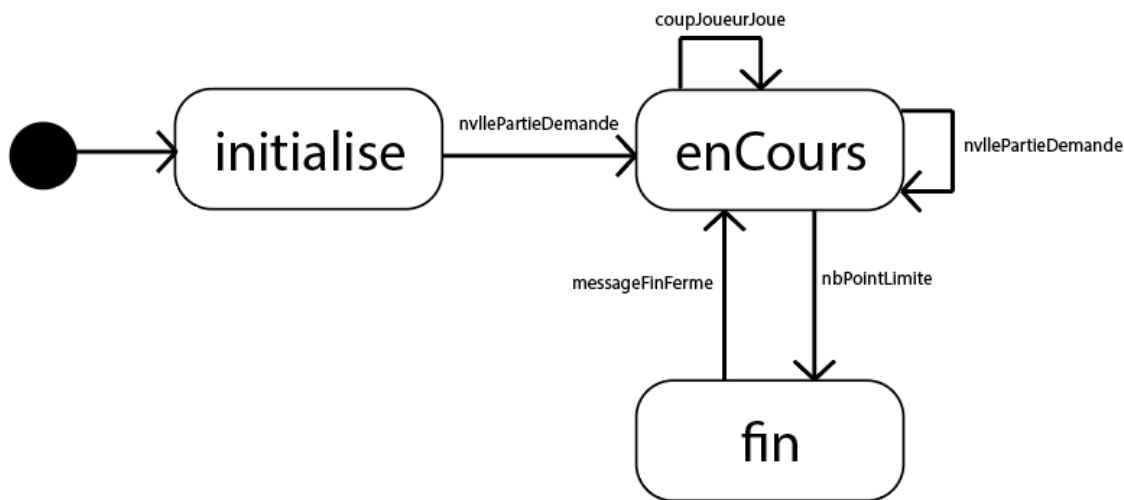
Il y aura 2 tests pour cette version :

- Cliquer sur le menu « Aide » puis « A propos de ... » afin qu'une boîte de message apparaisse avec des informations sur celle-ci (la version de l'application, la date de création et les auteurs).
- Cliquer sur le menu « Fichier » puis « Quitter » en haut de la fenêtre afin de fermer la fenêtre.

Version v4

15. Classe Chifoumi : Diagramme états-transitions

6. Diagramme états-transitions -actions du jeu



16. Dictionnaires des états, évènements et Actions

Dictionnaire des états du jeu

Nom	Description
Initialise	Le chifoumi est créé et initialisé : Le score joueur et machine sont mis à 0, les nombres de coup sont mis à 0.
enCours	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.
fin	Le chifoumi est finit, un message de fin est affiché.

Dictionnaire des événements faisant changer le jeu d'état

Nom	Description
nvlePartieDemande	L'utilisateur demande à lancer une nouvelle partie.
coupJoueurJouer	L'utilisateur clique sur l'un des coups.
nbPointLimite	L'utilisateur ou la machine atteignent le nombre de points limite de la partie
messageFinFerme	L'utilisateur ferme le message de fin

Description des actions réalisées lors de la traversée des transitions

Un des joueurs atteint le score limite	Lorsque le joueur ou la machine atteint un score de 5 le jeu s'arrête et affiche un message de fin.
Le joueur ferme le message de fin	Une fois que le joueur ferme le message de fin il peut rejouer une partie, la fenêtre de jeu ne se ferme pas.

7. 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

Evènements/ Etat	nvllePartieDemande	coupJoueurJouer	nbPointLimite	messageFinFerme
Initialise	enCours	x	x	x
enCours	enCours	enCours	fin	x
fin	x	x	x	EnCours

17.Éléments d'interface

Dans le répertoire VI_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.

18.Implémentation et tests

8.1 Implémentation

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, fin};

    /** 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 */
    unsigned int getNbPointsRequis();
    /* retourne le nombre de points requis pour gagner la partie */

    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 */

```

```

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
        du paramètre p_score */
    void setEtat(Etat e);
        /* initialise l'attribut etatJeu avec la valeur
        du paramètre e */
    void setNbPointsRequis(unsigned int nbPoints);
        /* initialise le nombre de points requis avec le paramètre nbPoints*/

    // 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
    unsigned int nbPointsRequis; // Nb de points requis pour gagné la partie

```

chifoumi.cpp

Nous avons ajouté un état dans le modèle : fin. Nous avons également ajouté une variable (avec son setteur et son getteur) : nbPointRequis, qui est le nombre de point requis pour finir la partie

```

case modele::fin:
    if(scoreJ > scoreM){
        QMessageBox::information(this, "Fin de la partie gagnant", "Bravo vous gagné la partie avec " + txt.QString::setNum(scoreJ) + " points");
    }
    else{
        QMessageBox::information(this, "Fin de la partie perdant", "Dommage vous avez perdu la partie avec " + txt.QString::setNum(scoreJ) + " points");
    }

```

chifoumi.h

Dans la classe chifoumi (la vue), dans l'actualisation nous avons rajouter le cas où l'état est à Fin.

```
void presentation::coupJoueurJoue(){
    _leModele->setCoupMachine(_leModele->genererUnCoup());
    _leModele->majScores(_leModele->determinerGagnant());
    if(_leModele->getNbPointsRequis() <= _leModele->getScoreJoueur() || _leModele->getNbPointsRequis() <= _leModele->getScoreMachine()){
        //Le score requis est atteint
        _laVue->actualisation(_leModele->getCoupJoueur(), _leModele->getCoupMachine(), _leModele->getScoreJoueur(), _leModele->getScoreMachine(), _leModele->getEtat(), _leModele->getNbPoint
        _leModele->setEtat(modele::Etat::fin);
        _laVue->actualisation(_leModele->getCoupJoueur(), _leModele->getCoupMachine(), _leModele->getScoreJoueur(), _leModele->getScoreMachine(), _leModele->getEtat(), _leModele->getNbPoint
        _leModele->setEtat(modele::Etat::enCours);
        nvlePartieDemandee();
    }
    else{
        _laVue->actualisation(_leModele->getCoupJoueur(), _leModele->getCoupMachine(), _leModele->getScoreJoueur(), _leModele->getScoreMachine(), _leModele->getEtat(), _leModele->getNbPoint
    }
}
```

Dans la méthode de la présentation coupJoueurJoue on ajoute une condition, si le joueur ou la machine atteignent le score requis alors l'état est mis à fin.

8.2 Test

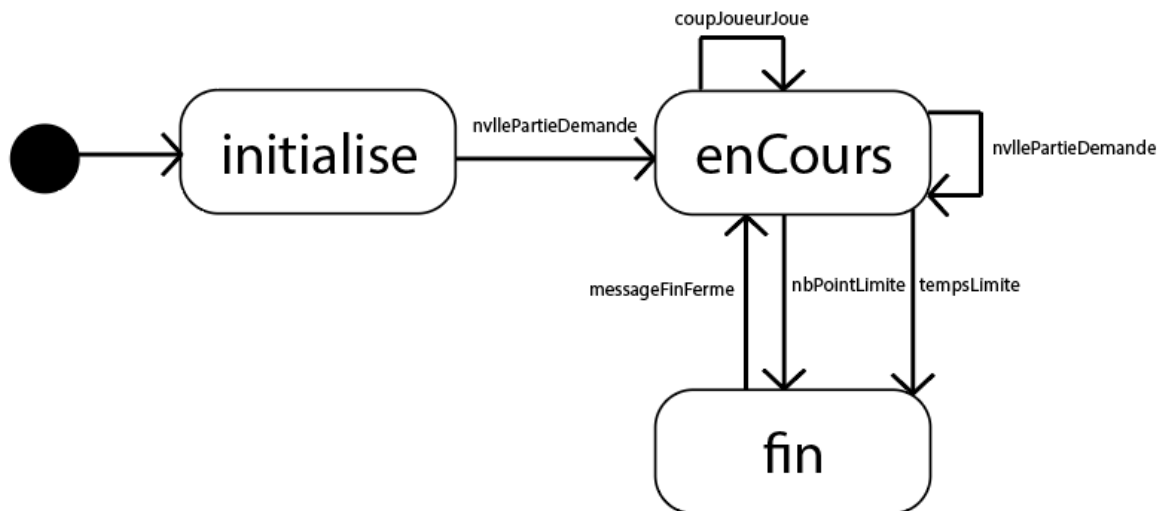
Il y aura 2 tests pour cette version :

- Nous allons jouer jusqu'à ce que la machine ou le joueur arrive à 5 points, à ce stade, la partie s'arrête on doit observer une boîte de message qui nous indique si on a gagné la partie ou alors si on l'a perdue on affiche aussi notre score.
- Lorsqu'on quitte la boîte de message on a l'occasion de jouer une autre partie.

Version v5

19. Classe Chifoumi : Diagramme états-transitions

8. Diagramme états-transitions -actions du jeu



20. Dictionnaires des états, évènements et Actions

Dictionnaire des états du jeu

Nom	Description
Initialise	Le chifoumi est créé et initialisé : Le score joueur et machine sont mis à 0, les nombres de coup sont mis à 0.
enCours	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.
fin	Le chifoumi est finit, un message de fin est affiché

Dictionnaire des événements faisant changer le jeu d'état

Nom	Description
nvlePartieDemande	L'utilisateur demande à lancer une nouvelle partie.
coupJoueurJouer	L'utilisateur clique sur l'un des coups.
nbPointLimite	L'utilisateur ou la machine atteignent le nombre de points limite de la partie

tempsLimite	Le temps de la partie s'est écoulé
messageFinFerme	L'utilisateur ferme le message de fin

Description des actions réalisées lors de la traversée des transitions

Le joueur atteint le temps limite	Lorsque le joueur atteint le temps limite, c'est-à-dire dès qu'il arrive à 0, le jeu se termine.
-----------------------------------	--

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

Evènements / Etat	nvlePartieDemande	coupJoueurJouer	nbPointLimite	tempsLimite	messageFinFerme
Initialise	enCours	x	x	x	x
enCours	enCours	enCours	fin	fin	x
fin	x	x	x	x	Initialise

21.Éléments d'interface

Dans le répertoire VI_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.

22.Implémentation et tests

8.1 Implémentation

chifoumi.h

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

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

Dans la vue nous avons créé une nouvelle méthode majTimer, elle sert à mettre à jours le Timer à l'écran.

presentation.h

```

public slots:
    void boutonFeuille();
    void boutonCiseau();
    void boutonPierre();
    void nvllePartieDemandee();
    void aProposDe();
    void demandePause();

private slots:
    void updateTimer();

private:
    QTimer* _time;
    int tmps;
};

```

Dans la présentation nous avons ajouté deux nouvelles variables : `_time` et `tmps`, `_time` étant un objet Timer et `tmps` étant le temps requis pour finir la partie

Nous avons également ajouté deux slots : `demandePause` qui va permettre en cliquant sur le bouton pause de mettre le temps en pause. Et `updateTimer` qui est la méthode du Timer qui va permettre chaque seconde de décrémenter le Timer et de gérer le fait que le Timer arrive à 0.

8.2 Test

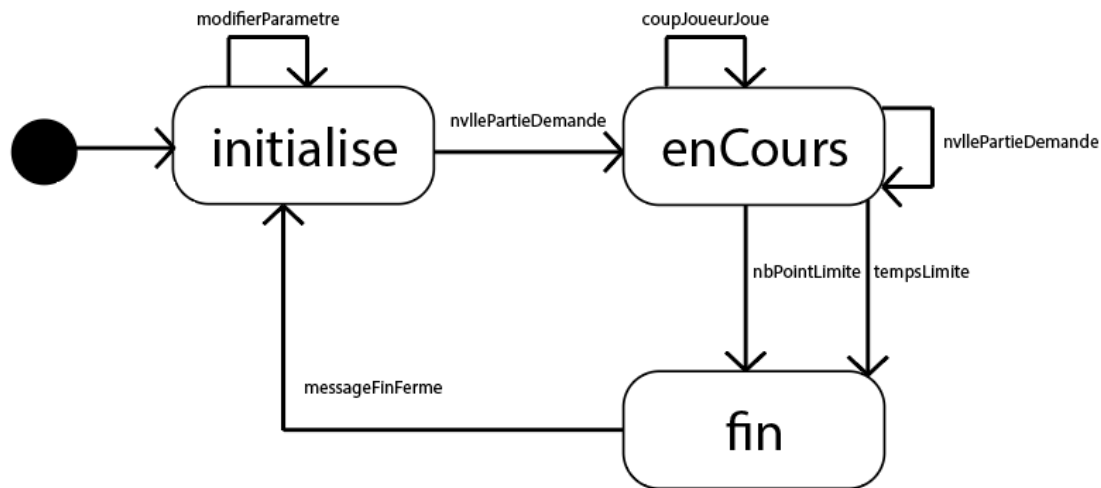
Il y aura 4 tests pour cette version :

- *Nous lancerons la partie et ferons en sorte d'être gagnant puis d'attendre la fin du timer pour voir si le message comme quoi nous sommes gagnants s'affiche.*
- *Nous lancerons la partie et ferons en sorte d'être perdant puis d'attendre la fin du timer pour voir si le message comme quoi nous sommes perdants s'affiche.*
- *Nous lancerons la partie et ferons en sorte d'être à égalité puis d'attendre la fin du timer pour voir si le message comme quoi nous sommes à égalité avec la machine s'affiche.*
- *Nous ferons tout de même un test pour voir si quand un des joueurs arrive à 5 points et que le timer n'est pas terminé le jeu s'arrête.*

Version v6

1. Classe Chifoumi : Diagramme états-transitions

Diagramme états-transitions -actions du jeu



23. Dictionnaires des états, évènements et Actions

Dictionnaire des états du jeu

Nom	Description
Initialise	Le chifoumi est créé et initialisé : Le score joueur et machine sont mis à 0, les nombres de coup sont mis à 0.
enCours	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.
fin	Le chifoumi est finit, un message de fin est affiché.

Dictionnaire des événements faisant changer le jeu d'état

Nom	Description
nvlePartieDemande	L'utilisateur demande à lancer une nouvelle partie.
coupJoueurJouer	L'utilisateur clique sur l'un des coups.

nbPointLimite	L'utilisateur ou la machine atteignent le nombre de points limite de la partie
tempsLimite	Le temps de la partie s'est écoulé
messageFinFerme	L'utilisateur ferme le message de fin
modifierParametre	L'utilisateur modifie les paramètres de la partie.

Description des actions réalisées lors de la traversée des transitions

Le joueur modifie les paramètres	Lorsque le joueur modifie les paramètres de la partie nous sommes toujours dans l'état initialisé de la partie.
----------------------------------	---

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

Evènements/ Etat	nvlePartieDem ande	coupJoueurJ ouer	nbPointLi mite	tempsLi mite	messageFinF erme	modifierPara metre
Initialise	enCours	x	x	x	x	Initialise
enCours	enCours	enCours	fin	fin	x	x
fin	x	x	x	x	Initialise	x

24.Éléments d'interface

Dans le répertoire VI_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.
- dialogparametre.cpp : Fenêtre de changement des paramètres

25. Implémentation et tests

8.1 Implémentation

dialogparametre.h

```
#ifndef DIALOGPARAMETRE_H
#define DIALOGPARAMETRE_H

#include <QDialog>

namespace Ui {
class DialogParametre;
}

class DialogParametre : public QDialog
{
    Q_OBJECT

public:
    explicit DialogParametre(QWidget *parent = nullptr, QString = "Vous", int=5,int=30);
    ~DialogParametre();
    QString getNom();
    int getNbPtsMax();
    int getTpsMax();
    bool getEnregistrer();

private slots:
    void enregistrer();

private:
    Ui::DialogParametre *ui;
    bool estEnregistrer;
};

#endif // DIALOGPARAMETRE_H
```

Ajout de la classe DialogParametre permettant de gérer la fenêtre de Dialogue pour paramétrer la partie.

presentation.h

```
public slots:
    void boutonFeuille();
    void boutonCiseau();
    void boutonPierre();
    void nvllePartieDemandee();
    void aProposDe();
    void demandePause();
    void parametre();
```

Dans la présentation ajout du slot parametre permettant de lier le bouton à l'ouverture de la fenêtre de Dialogue.

8.2 Test

Il y aura 4 tests pour cette version :

- Nous testerons si en cliquant sur le menu Fichier en haut de la fenêtre puis sur le bouton Paramétrer une fenêtre de dialogue s'affiche.
- Ensuite nous testerons de modifier le nom du joueur pour voir s'il change dans l'endroit dédié lors du jeu.
- Puis nous testerons de modifier le nombre max de points en jouant jusqu'à ce qu'un des deux joueurs l'atteigne, il faudra que la partie s'arrête.
- Enfin nous modifierons le temps maximum de jeu, nous verrons s'il s'affiche et nous le ferons écouler jusqu'à la fin pour voir s'il ne garde pas l'ancien temps.

Version v7

26.Éléments d'interface

Dans le répertoire VI_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.
- dialogparametre.cpp : Fenêtre de changement des paramètres
- database.cpp : Permet la connexion à la base de donnée situé sur Lakartxela
- dialogconnexion.cpp : Fenêtre de connexion

27.Implementation et tests

8.1 Implémentation

database.h

```
#ifndef DATABASE_H
#define DATABASE_H

#include <QtSql/QtSqlDatabase>
#include <QVariantList>

#define DATABASE_NAME "BD_NodenitBdl_Lakartxela"
#define CONNECT_TYPE "QODBC"

class DataBase
{
public:
    DataBase();
    bool openDataBase();
    void clseDataBase();
    bool restorDataBase();
    bool insertJoueur();
private:
    QSqlDatabase mydb;
};

#endif // DATABASE_H
```

database.h

```

#include "database.h"
#include <QSqlQuery>

DataBase::DataBase()
{
}

bool DataBase::openDataBase(){
    this->mydb = QSqlDatabase::addDatabase(CONNECT_TYPE);
    this->mydb.setDatabaseName(DATABASE_NAME);
    return this->mydb.open();
}

void DataBase::closeDataBase(){
    this->mydb.close();
}

bool DataBase::restorDataBase(){
    QSqlQuery query;
    QString insertions="Select * from JOUEUR";
    query.prepare(insertions);

    if(query.exec()){
        qDebug("Table existante");
    }else{
        qDebug("Table inexistante. Création de la table en cours...");
        QSqlQuery query;
        QString insertions="create table JOUEUR (id_joueur integer(3) primary key, login varchar(50), mdp varchar(50));";
        query.prepare(insertions);
        if(query.exec()) qDebug("Create table JOUEUR");
        else qDebug("Create table fail");
        if(insertJoueur()) qDebug("Insertion des premiers JOUEUR");
        else qDebug("Insert JOUEUR fail");
    }
    return true;
}

bool DataBase::insertJoueur()
{
    QSqlQuery query;
    if (!query.exec("insert into JOUEUR values(1, 'root', 'root');")) return false;

    QString insertions="insert into JOUEUR values(?, ?, ?);";
    query.prepare(insertions);

    QVariantList id;
    id << 2 ;
    query.addBindValue(id);

    QVariantList login;
    login << "SAE";
    query.addBindValue(login);

    QVariantList mdp;
    mdp << "1234";
    query.addBindValue(mdp);

    if (!query.execBatch()) return false;

    return true;
}

```

dialogconnexion.h

```

#ifndef DIALOGCONNEXION_H
#define DIALOGCONNEXION_H

#include <QDialog>
#include "database.h"

namespace Ui {
class DialogConnexion;
}

class DialogConnexion : public QDialog
{
    Q_OBJECT

public:
    explicit DialogConnexion(QWidget *parent = nullptr);
    ~DialogConnexion();

    bool getEstConnecter();
    bool getQuitter();

private slots:
    void demandeConnexion();
    void demandeQuitter();

private:
    Ui::DialogConnexion *ui;
    DataBase *db;
    bool estConnecter;
    bool quitter;
};

#endif // DIALOGCONNEXION_H

```

Dialogconnexion.cpp

```

#include "dialogconnexion.h"
#include "ui_dialogconnexion.h"
#include <QSqlQuery>
#include <QDebug>

DialogConnexion::DialogConnexion(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::DialogConnexion)
{
    ui->setupUi(this);
    ui->Erreur->setHidden(true);
    db = new DataBase();
    db->openDataBase();
    db->restorDataBase();

    estConnecter = false;
    connect(ui->Connexion, SIGNAL(clicked()),this,SLOT(demandeConnexion()));
}

DialogConnexion::~DialogConnexion()
{
    delete ui;
}

void DialogConnexion::demandeConnexion(){
    ui->Erreur->setHidden(true);
    if(ui->LineLogin->text() == "" || ui->LineMDP->text() == "" ) ui->Erreur->setHidden(false);
    else {
        //Vérification du mdp et login
        QSqlQuery query;
        QString insertions="select login,mdp from JOUEUR where login = ? and mdp = ?";
        query.prepare(insertions);

        QVariantList login;
        login << ui->LineLogin->text() ;
        query.addBindValue(login);

        QVariantList mdp;
        mdp << ui->LineMDP->text();
        query.addBindValue(mdp);

        if (!query.execBatch()) qDebug() << "Lancement de la vérification fail";
        query.next();
        if (query.value(0).toString()==ui->LineLogin->text() && query.value(1).toString() == ui->LineMDP->text()) {
            estConnecter = true;
            qDebug() << "Login";
            this->close();
        }
        else {estConnecter = false; qDebug() << "Fail";}
    }
}

void DialogConnexion::demandeQuitter(){
    quitter=true;
    this->close();
}

bool DialogConnexion::getEstConnecter(){
    return estConnecter;
}

bool DialogConnexion::getQuitter(){
    return quitter;
}

```

8.2 Test

Il y aura 4 tests pour cette version :

- Nous testerons si en lançant l'application qu'une fenêtre de dialogue s'affiche.
- Ensuite nous testerons la création et l'insertion des joueurs avec login et mdp dans la base de données
- Puis nous testerons de nous connecter avec le bon login et mdp.

Nous vérifierons aussi que si un mot de passe est erroné, il demande une reconnexion

Base de données :

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
<input type="checkbox"/> 1	id_joueur	int(3)			Non	Aucun(e)			Modifier Supprimer Plus
<input type="checkbox"/> 2	login	varchar(50)	latin1_swedish_ci		Oui	NULL			Modifier Supprimer Plus
<input type="checkbox"/> 3	mdp	varchar(50)	latin1_swedish_ci		Oui	NULL			Modifier Supprimer Plus

Création de la base de données réussit

		id_joueur	login	mdp	▲ 1
<input type="checkbox"/>	Éditer	Copier	Supprimer	2	SAE 1234

Insertion réussi

Version v8

28.Éléments d'interface

Dans le répertoire VI_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.
- dialogparametre.cpp : Fenêtre de changement des paramètres
- database.cpp : Permet la connexion à la base de donnée situé sur Lakartxela
- dialogconnexion.cpp : Fenêtre de connexion

29.Implementation et tests

8.1 Implémentation

database.h

```
#ifndef DATABASE_H
#define DATABASE_H

#include <QtSql/QtSqlDatabase>
#include <QVariantList>

#define DATABASE_NAME "BD_NodenitBd1_Lakartxela"
#define CONNECT_TYPE "QODBC"

class DataBase
{
public:
    DataBase();
    bool openDataBase();
    void clseDataBase();
    bool restorDataBase();
    bool insertJoueur();
    bool insertScore(int idJ,QString nomJ,int scoreJ, int scoreM);
private:
    QSqlDatabase mydb;
};

#endif // DATABASE_H
```

database.cpp

Ajout de la fonction insertScore

```

bool DataBase::insertScore(int idJ, QString nomJ, int scoreJ, int scoreM){
    QSqlQuery query;
    QString insertions="select * from RESULTAT";
    query.prepare(insertions);

    if(query.exec()){
        qDebug("Table RESULTAT existante");
    }else{
        qDebug("Table inexistant. Création de la table en cours...");
        QSqlQuery query;
        QString insertions="create table RESULTAT (id_resultat integer(6) primary key,horodatage DATE, nomJoueurHumain varchar(50),scoreJoueurHumain integer(6), "
            "nomJoueurMachine varchar(50),scoreJoueurMachine integer(6), id_joueur integer(3),foreign key (id_joueur) REFERENCES JOUEUR(id_joueur));";
        query.prepare(insertions);
        if(query.exec()) qDebug("Create table RESULTAT");
        else qDebug("Create table fail");
    }

    insertions="INSERT INTO `RESULTAT`(`id_resultat`, `horodatage`, `nomJoueurHumain`, "
        "`scoreJoueurHumain`, `nomJoueurMachine`, `scoreJoueurMachine`, `id_joueur`) VALUES (?, ?, ?, ?, ?, ?, ?)";
    query.prepare(insertions);

    QSqlQuery maxID;
    if(!maxID.exec("select MAX(id_resultat) from RESULTAT")) return false;
    maxID.next();

    QVariantList idResultat;
    idResultat << maxID.value(0).toInt()+1;
    query.addBindValue(idResultat);

    QVariantList date;
    date << QDateTime::currentDateTime().toString("yyyy/MM/dd");
    query.addBindValue(date);

    QVariantList nomJoueur;
    nomJoueur << nomJ;
    query.addBindValue(nomJoueur);

    QVariantList scoreJoueur;
    scoreJoueur << scoreJ;
    query.addBindValue(scoreJoueur);

    QVariantList nomMachine;
    nomMachine << "Machine";
    query.addBindValue(nomMachine);

    QVariantList scoreMachine;
    scoreMachine << scoreM;
    query.addBindValue(scoreMachine);

    QVariantList id;
    id << idJ;
    query.addBindValue(id);

    if (!query.execBatch()) {
        qDebug("Insert score Fail");
        return false;
    }
    else qDebug("Insert score réussit");
    return true;
}

```

Presentation.cpp

```

void presentation::demandeActualisation()
{
    _leVue->actualisation(_leModele->getCoupJoueur(),_leModele->getCoupMachine(),_leModele->getScoreJoueur(),_leModele->getScoreMachine(),_leModele->getEtat(),
        tmps, _leModele->getTpsMax(), _leModele->getNom());
    if(_leModele->getEtat() == modele::Etat::fin) {
        _leModele->db->insertScore(_leModele->getId(),_leModele->getNom(),_leModele->getScoreJoueur(),_leModele->getScoreMachine());
    }
}

```




8.2 Test

Il y aura 2 tests pour cette version :





- Nous testerons la création et l'insertion des scores dans la base de données
- Puis nous testerons de finir une partie pour tester l'insertion.
















Base de données :

mheriveau_bd JOUEUR

 id_joueur : int(3)
 login : varchar(50)
 mdp : varchar(50)

mheriveau_bd RESULTAT

 id_resultat : int(6)
 horodatage : date
 nomJoueurHumain : varchar(50)
scoreJoueurHumain : int(6)
 nomJoueurMachine : varchar(50)
scoreJoueurMachine : int(6)
id_joueur : int(3)

<input type="checkbox"/>	 Editor	 Copier	 Supprimer	6	2022-06-06	root	1	Machine	5	1
<input type="checkbox"/>	 Editor	 Copier	 Supprimer	7	2022-06-06	root	5	Machine	4	1
<input type="checkbox"/>	 Editor	 Copier	 Supprimer	8	2022-06-06	root	3	Machine	5	1
<input type="checkbox"/>	 Editor	 Copier	 Supprimer	9	2022-06-06	root	3	Machine	5	1
<input type="checkbox"/>	 Editor	 Copier	 Supprimer	10	2022-06-06	root	5	Machine	3	1