Tom Montbord - Guillian Celle - Oier Cesat

TP 5 - TD III

Saé 2.01 – Développement d'une application

Chifoumi – Dossier d'Analyse et conception

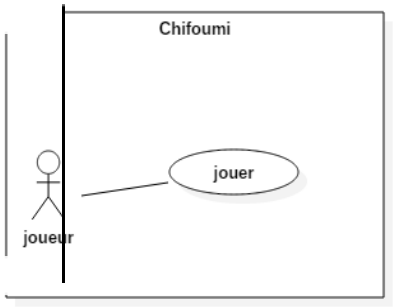
Table des matières :

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.

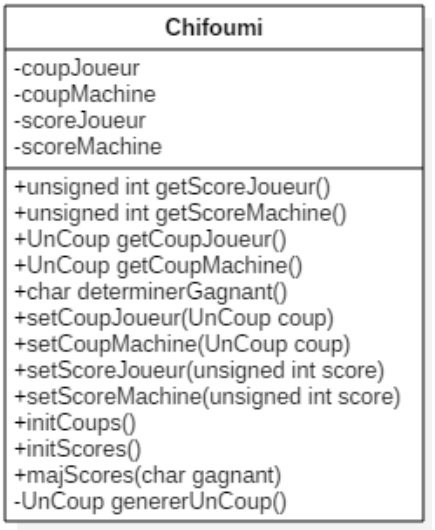


Figure 2 : Diagramme de Classes UML du jeu Chifoumi

(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`	Mémorise la dernière figure choisie par le joueur. Type énuméré `enum unCoup {pierre, ciseau, papier, rien};`	`UnCoup`	`papier`
`coupMachine`	Mémorise 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 4

```
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 4 : 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 :
-main.cpp:
Respectivement spécification et corps de la classe Chifoumi décrite au paragraphe 4.

5.2 Test

Test avec le programme fournit main.cpp

Valeurs fournies / attendues… comme montré dans la ressource R2.03 (partie tests)

Classe	Description	valeur en entrée	valeur en sortie	résultat
getScoreJoueur	récupère la valeur du dernier score du joueur	0	0	ok
getScoreMachine	récupère la valeur du dernier score de la machine	0	0	ok
getCoupJoueur	récupère la valeur du dernier coup de la machine	rien	rien	ok
getCoupMachine	récupère la valeur du dernier coup de la machine	rien	rien	ok
setScoreJoueur	Initialise le score du joueur avec une valeur (p_score)	0	1	ok
setScoreMachine	Initialise le score de la machineune valeur (p_score)	0	2	ok

initScore	Initialise les scores de la machine et du joueur	0	0	ok
setCoupMachine	Initialise le coup du joueur avec une valeur (p_score)	rien	ciseau	ok
setCoupJoueur	Initialise le coup du joueur avec une valeur (p_score)	rien	Pierre	ok

Ajouter une colonne de la valeur attendue

Version v1

6. Classe Chifoumi : Diagramme états-transitions

(a) Diagramme états-transitions -actions du jeu

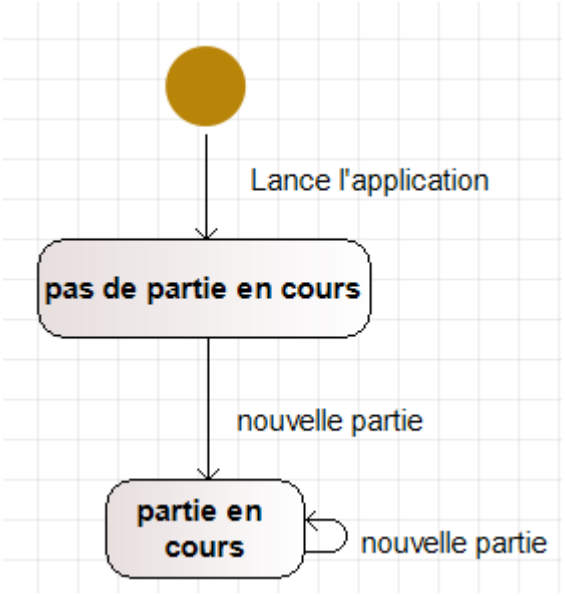


Figure 9 : Diagramme états-transitions

Dictionnaire des états du jeu

nomEtat	*Signification*
`accueil`	La partie n'a pas débuté
`enJeu`	La partie est en cours

Tableau 2 : États du jeu

Dictionnaire des événements faisant changer le jeu d'état

nomEvénement	*Signification*
`partieCommence`	La partie est lancé par le joueur
`partieTermine`	La partie se termine soit par victoire ou plus de temps

Tableau 3 : Evénements faisant changer le jeu d'état

Description des actions réalisées lors de la traversée des transitions

`Clique sur commencer la partie`	l'utilisateur clique sur commencer la partie
`Le score maximal est atteint`	Soit l'utilisateur soit le joueur à atteint le score maximal
`le temps maximal est atteint`	le temps est atteint et la partie se termine

Tableau 4 : 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

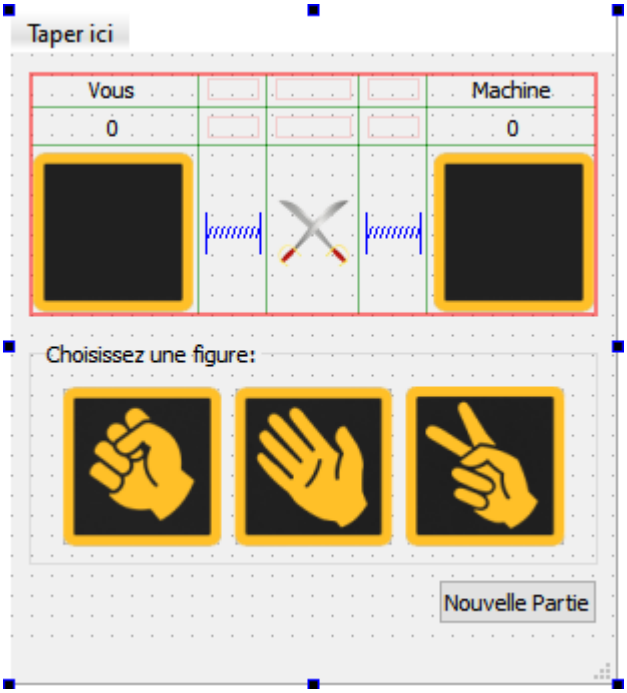
- en *colonne : les **états*** du jeu

Événement **à** *nomEtatJeu*	partie à l'arrêt	partie en cours
lance l'application	1 (nouvelle partie)	0
clique sur "nouvelle partie"	0	1

Tableau 5 : 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



Objet	Classe
MainWindow	QMainWindow
centralwidget	QWidget
horizontalLayout	QHBoxLayout
gridLayout	QGridLayout
horizontalSpacer	Spacer
horizontalSpacer_2	Spacer
labelImageJoueur	QLabel
labelImageMachine	QLabel
labelImageVersus	QLabel
labelIntituleJoueur	QLabel
labelIntituleMachine	QLabel
labelScoreJoueur	QLabel
labelScoreMachine	QLabel
bNouvellePartie	QPushButton
groupBox	QGroupBox
bCiseaux	QPushButton
bFeuille	QPushButton
bPierre	QPushButton
horizontalSpacer_3	Spacer
horizontalSpacer_4	Spacer
verticalSpacer	Spacer
menubar	QMenuBar
statusbar	QStatusBar

L'interface est séparée en deux parties, la première en haut est l'affichage des scores et et le déroulement du jeu, la deuxième étant le choix de la figure.

La première partie étant une partie regroupant l'affichage des scores et l'affichage du coup machine et du coup joueur nous avons fait le choix de tout regrouper sur une grille afin d'avoir un comportement adéquat lors de l'élargissement de la page.
Sur la première colonne, nous avons mis trois label : le premier précisant à qui appartient le score (labelIntituleJoueur), le deuxième le score joueur (labelScoreJoueur) et le troisième l'affichage du choix joueur sous forme d'image (labelImageJoueur).
Les 2,3 et 4èmes colonnes sont dédiées à l'habillage et l'image des épées croisées entre les deux (labelImageVersus).
La dernière colonne est la même que la première à la différence que c'est l'affichage du score machine et du choix de celle-ci (labelIntituleMachine, labelScoreMachine, labelImageMachine).

Dans la seconde partie se déroule le choix des figures jouées par le joueur, il a donc 3 boutons symbolisant les figures jouables (Pierre, Feuille, Ciseaux). Chaque choix est représenté par un bouton contenant l'image du coup à jouer (bPierre, bFeuille, bCiseaux). En dessous de se bloque, on retrouve le bouton de relance d'une nouvelle partie, remettant les scores à 0 (bNouvellePartie).

A faire ici : description sommaire des éléments de l'interface, par exemple, avec une copie d'écran sur laquelle sont nommés les variables/objets graphiques et où les layouts sont positionnés et nommés.

8. Implémentation et tests

8.1 Implémentation

FICHIERS:
mainwindow.h: Il gère l'initialisation du code de la fenêtre de jeu
chifoumi.h: Il gère l'initialisation du code du déroulement de la partie
chifoumi.cpp : Il gère le déroulement des différentes actions décrite dans le chifoumi.h
main.cpp: Lancement de la partie en appelant et la fenêtre et le code de la partie
mainwindow.cpp: Il gère l'affichage direct de la fenêtre de la partie

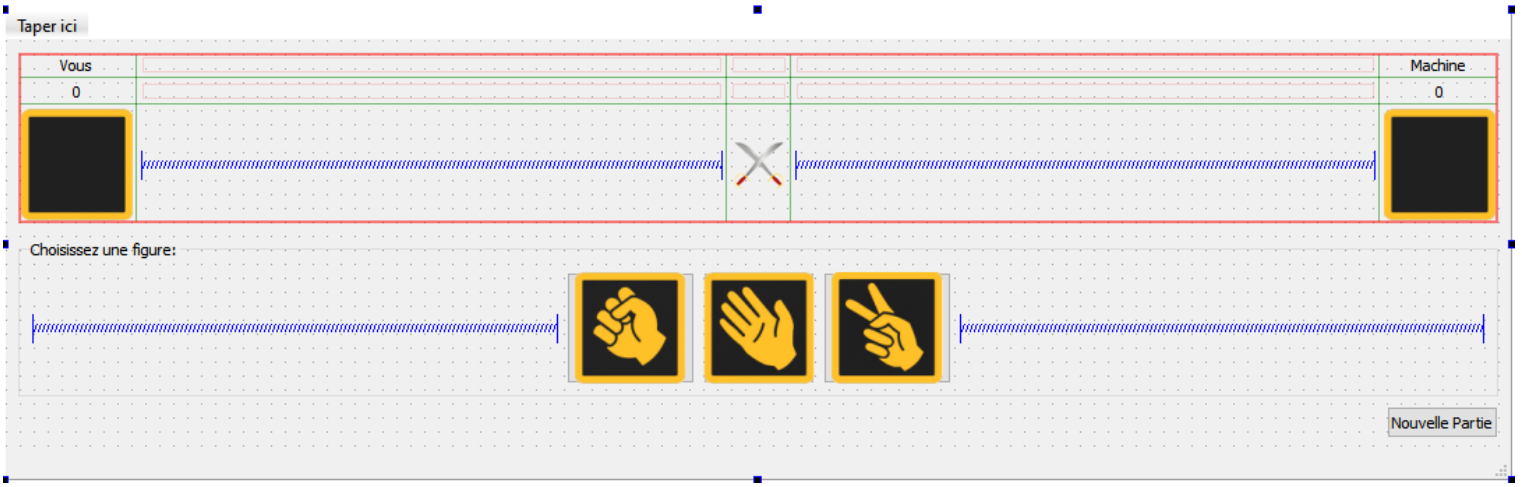
SIGNAL/SLOT:

```
void lancerPartie();
//lance une nouvelle partie en effaçant les données de la précédente et met la couleur bleu sur  le joueur

void tourMachine(Modele::UnCoup,int=0,int=0);
//met la couleur bleu sur la machine, choisit son coup au hasard et l'affiche à l'écran puis lance finPartie()

 void jouerPierre();
 void jouerFeuille();
 void jouerCiseaux();
//change le coup du joueur, l'affiche à l'écran et lance tourMachine()

 void setBlue(char);
//modifie le score et prénom du joueur cible en bleu et change ceux de l'autre joueur en noir
//caractère qui définit la cible: 'J' pour le joueur et 'M' pour la machine

 void definirImageMachine(Chifoumi::UnCoup);
 //définit l'image du joueur machine en fonction de son coup Chifoumi::UnCoup

  void finManche();
  //définis le joueur gagnant la manche, met à jour les scores, les affiches à l'écran et met la couleur bleu sur le joueur humain

SIGNAL(clicked())
//Tout les SIGNAL nécessaire sont des clicked()
```

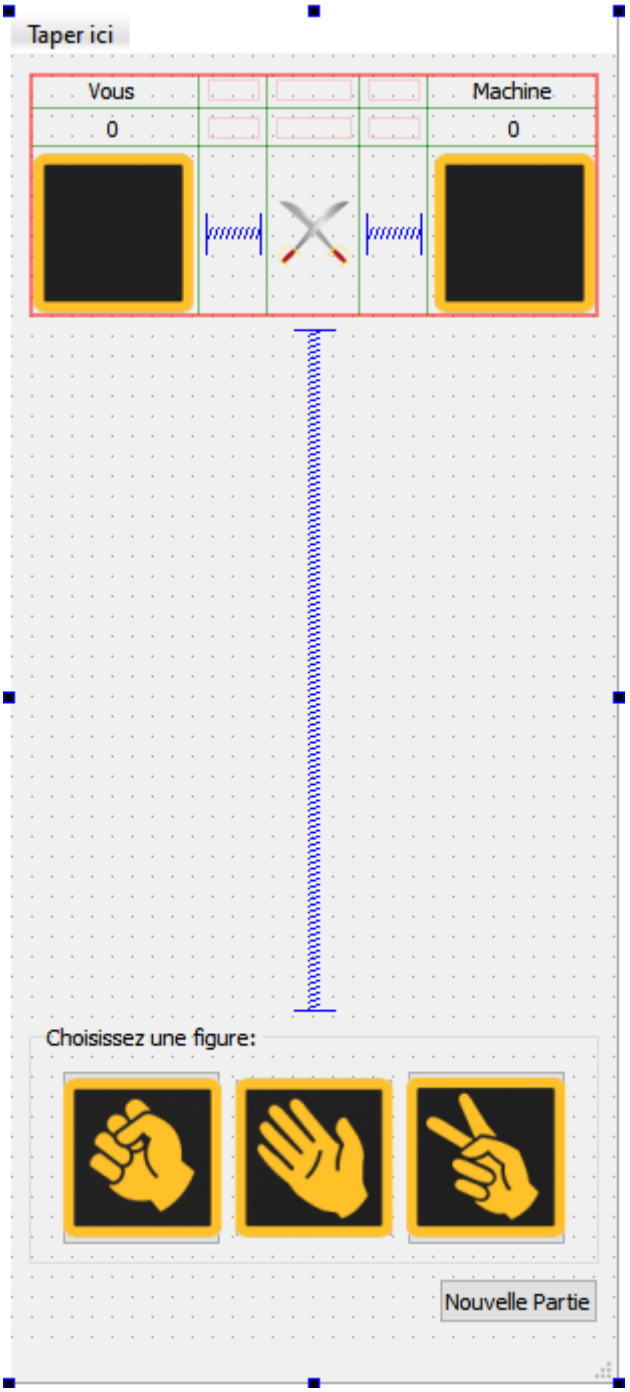
8.2 Test

Tests prévus pour les layout :

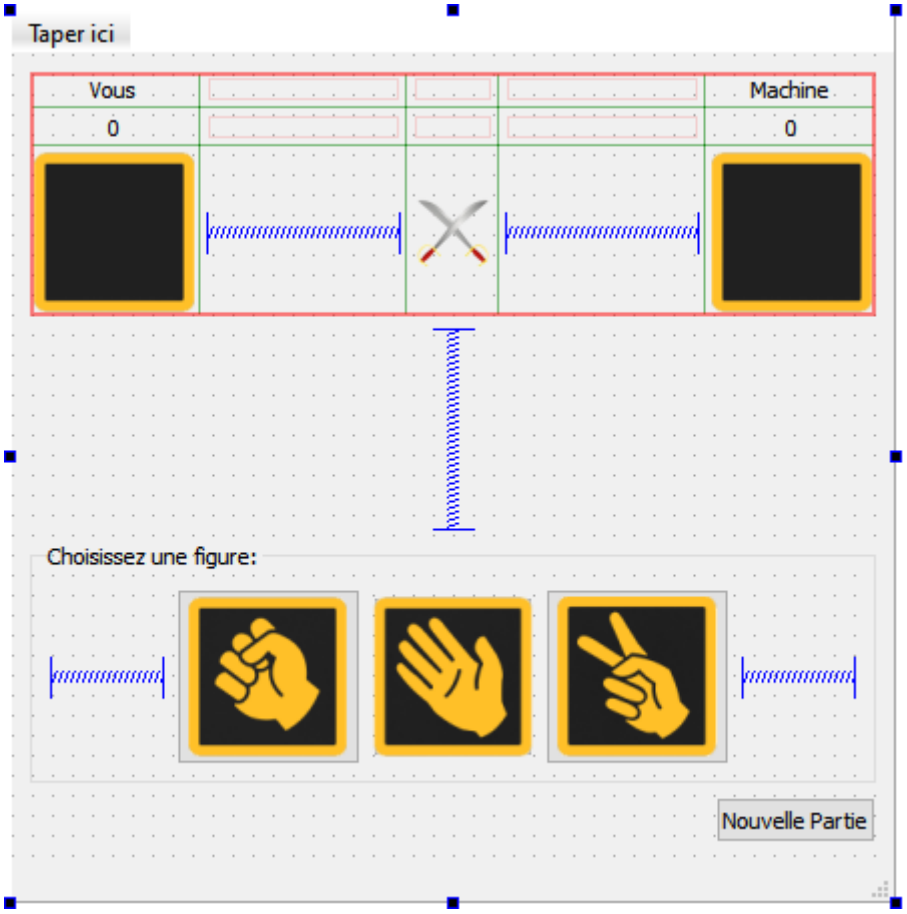
- Élargissement de la page sur la largeur avec les élargissements adéquats.



Élargissement de la page sur la longueur avec les élargissements adéquats.

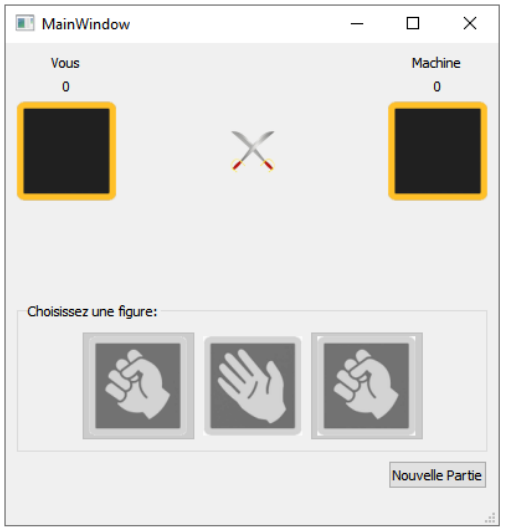


- Élargissement de la page sur la diagonale avec les élargissements adéquats.

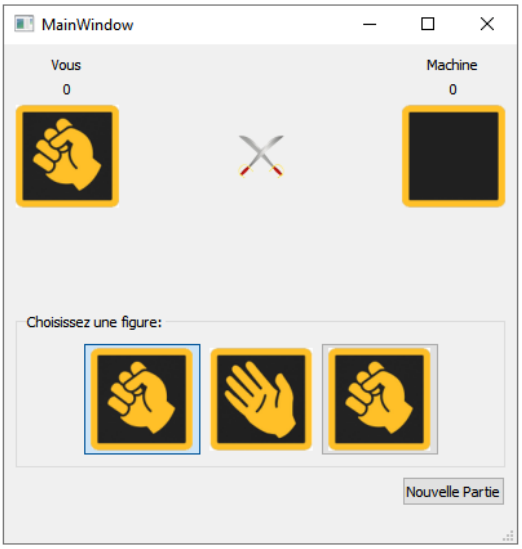


Tests prévu sur l'exécution du programme :

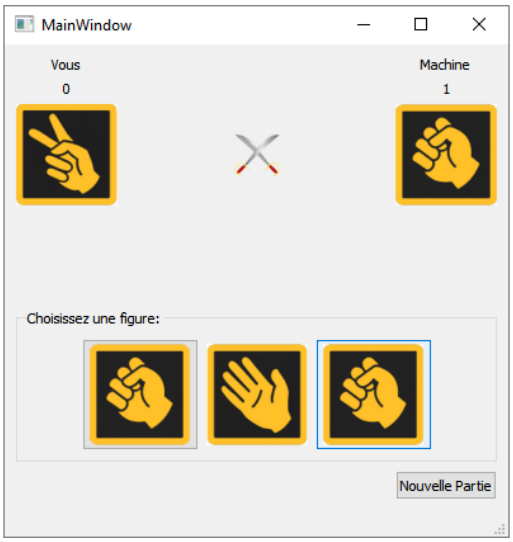
- État initiale de l'application, impossibilité de cliqué sur les boutons figures et affichage nul sur les images et les scores



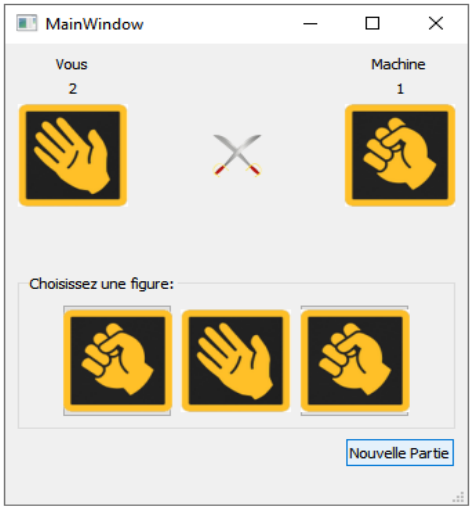
- L'utilisateur clique sur le bouton pierre et la pierre s'affiche sur le labelImageMachine avec le score associé



- L'utilisateur clique sur le bouton ciseaux et la pierre s'affiche sur le labelImageMachine avec le score associé



- L'utilisateur clique sur le bouton feuille et la pierre s'affiche sur le labelImageMachine avec le score associé



Version v2

1. Liste des fichiers sources de cette version (et rôle de chacun)

Fichiers sources .cpp:

chifoumiVue.cpp: Gère l'interface avec laquelle le joueur interagit
modele.cpp: Gère les commandes réalisés par l'app
presentation.cpp: Valide les entrées de l'utilisateur et ce à quoi elles correspondent dans le modèle
main.cpp: Exécute le jeu Chifoumi

Fichiers sources .ui:

ChifoumiVue.ui: Gère le modèle graphique de l'application

2. Présentation des .h de chacune des classes

chifoumiVue.h:

"chifoumieVue.h" s'occupe de l'interface avec l'utilisateur et a donc les classes en rapport avec cela.

void majInterface(): met à jour l'interface en changeant les textes ou les boutons en les mettant **able** ou **unable**
void nvlleConnexion(QObject *c): sert à créer la connexion avec la présentation

void supprConnexion(QObject *c): sert à supprimer la connexion avec la présentation

presentation.h:

Le .h "presentation.h" sert de liant, il y a donc les deux classes "Modele" qui s'occupe de toute la gestion du "Modele.h" et "ChifoumiVue" qui gère "ChifoumiVue.h" gérant la vue dans ce modèle MVP.

Modele* getModele(): getter recevant l'état initial du modele
Chifoumi* getVue(): getter recevant l'état initial de la vue
void setModele(Modele *m): Setter permet de modifier le modele
void setVue(chifoumiVue *v): Setter permet de modifier la vue

SLOTS:

void demanderLancerPartie(): Demande au modèle de lancer une partie suite au SIGNAL clicked de l'utilisateur
void demanderJouerPierre(): Demande au modèle de jouer la pierre suite au SIGNAL clicked de l'utilisateur
void demanderJouerPapier():Demande au modèle de jouer la papier suite au SIGNAL clicked de l'utilisateur
void demanderJouerCiseau():Demande au modèle de jouer la ciseau suite au SIGNAL clicked de l'utilisateur
void demanderQuitterApp(): Demande au modèle de quitter l'application suite au SIGNAL trigerred de l'utilisateur
void demanderInfosApp(): Demande au modèle d'ouvrir la messageBox d'infos suite au SIGNAL trigerred de l'utilisateur

3. Résultats des tests réalisés

N'ayant pas de nouvelle fonctionnalité, nous reprenons simplement les tests précédents et voir s'ils sont toujours efficaces.

Classe	Description	valeur en entrée	valeur en sortie	résultat
getScoreJoueur	récupère la valeur du dernier score du joueur	0	0	ok
getScoreMachine	récupère la valeur du dernier score de la machine	0	0	ok
getCoupJoueur	récupère la valeur du dernier coup de la machine	rien	rien	ok
getCoupMachine	récupère la valeur du dernier coup de la machine	rien	rien	ok
setScoreJoueur	Initialise le score du joueur avec une valeur (p_score)	0	1	ok
setScoreMachine	Initialise le score de la machineune valeur (p_score)	0	2	ok
initScore	Initialise les scores de la machine et du joueur	0	0	ok
setCoupMachine	Initialise le coup du joueur avec une valeur (p_score)	rien	ciseau	ok
setCoupJoueur	Initialise le coup du joueur avec une valeur (p_score)	rien	Pierre	ok

Interfaces graphiques toujours fonctionnel

Version v3

1. liste des fichiers sources

chifoumi.cpp : corps du module chifoumi
mainWindow.cpp : gère l'application en fonction des actions du joueur
main.cpp : créer l'application et l'exécute

2. headers modifiés

Les procédures **quitterApp()** et **infosApp()** ont été ajoutées au fichiers **mainWindow,** ils permettent respectivement de quitter l'application et d'afficher les informations sur la version, la dernière date de modification et le nom des créateurs de l'application.

3. Tests et résultats

Tests pour **quitterApp()** :

- test avec un clique de souris
- test avec les raccourcis (Alt + F3)

Tests pour **infosApp()** :

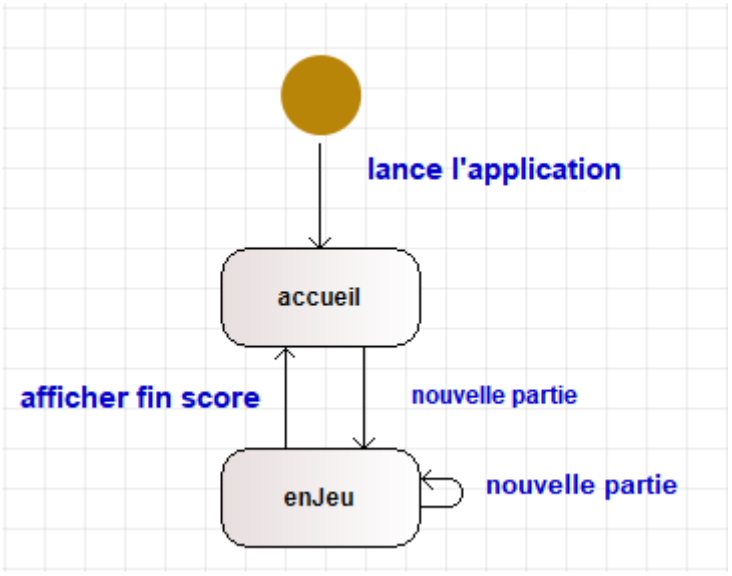
- test avec un clique de souris
- test avec les raccourcis (F1)

Chaque test à été réalisé en mode **accueil** ou **enJeu** et a été validé (la procédure s'est lancée).

Classe	Description	valeur en entrée	valeur en sortie	résultat
quitterApp	ferme la fenêtre lorsque actionQuitter est utilisé	fenetre ouverte	fenetre fermée	ok
infosApp	affiche les infos de la version lorsque actionInfosApp est utilisé	menu fermé	menu ouvert	ok

Version v4

1. Diagramme d'état-transitions


lance l'application
accueil
afficher fin score
nouvelle partie
enJeu
nouvelle partie

2. Dictionnaires

2.1 etats :

nomEtat	*Signification*
`accueil`	La partie n'a pas débuté ou le score max a été dépassé par l'un des deux joueurs
`enJeu`	La partie est en cours

2.2 événements :

Signification		
	accueil	**enJeu**
NouvellePartie	lance la partie	relance une partie
afficher fin score	affiche le score une fois la partie finie	X

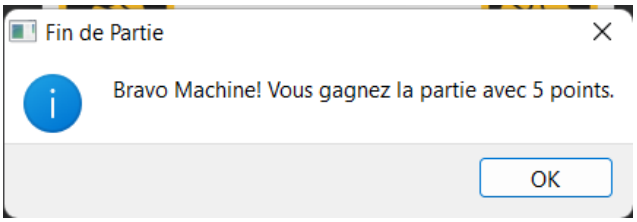
3. Diagramme état transition version matricielle

	accueil	enJeu
lance l'application	1	0
nouvelle partie	0	1
afficher fin score	1	0

Le jeu peut retourner en mode accueil quand un joueur dépasse le score maximal.

4. Nouveaux éléments d'interface

labelIntituleScore : texte fixe qui informe le joueur sur l'utilité de labelScoreTotal.
labelScoreTotal : Nombre (5 par défaut) qui indique le score maximal.



La dialog box "Fin de Partie" indique le gagnant et son nombre de points.

5. Présentation des fichiers sources

main.cpp : créer le modèle mvp et lance l'application
chifoumiVue.cpp : affiche les éléments d'interface à l'utilisateur et réceptionne ses intéractions avec les boutons
modele : stocke les données de l'objet modèle
presentation.cpp : lie la vue et le modèle

6. Présentation des nouvelles procédures

void afficherFinScore(int, QString) (placé dans chifoumiVue) : affiche une fenêtre de dialogue donnant les informations sur la victoire et bloque l'accès aux boutons.

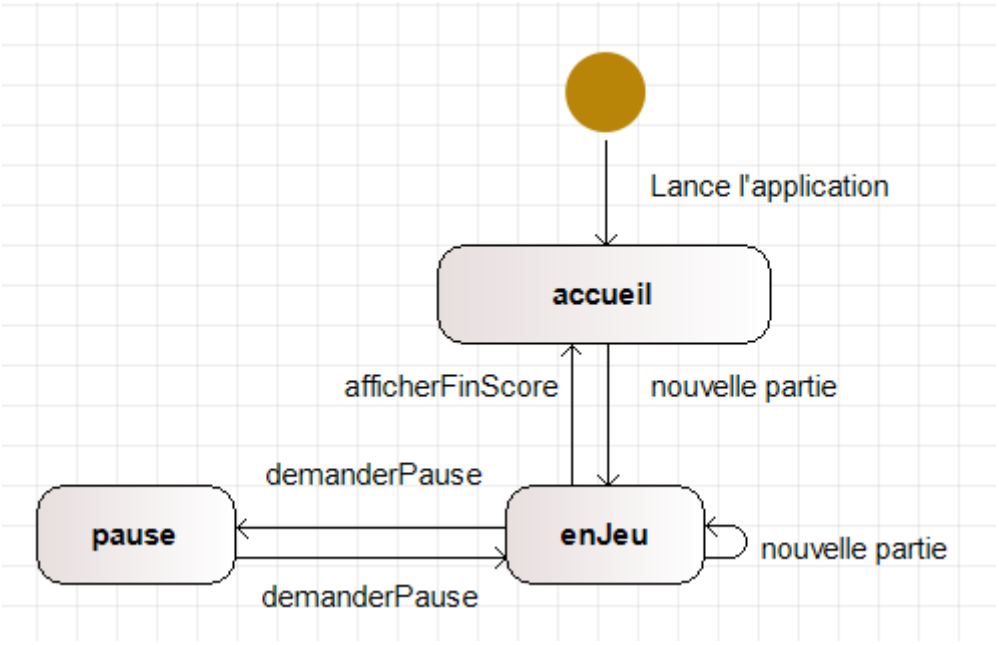
void demanderJouerTour() (placé dans présentation) : a été modifié pour pouvoir appeler **afficherFinScore(int, QString)** si jamais le score d'un joueur dépasse le score maximale.

7. Tests et résultats

Classe	Description	valeur en entrée	valeur en sortie	résultat
afficherFinScore(int, QString)	ouvre une fenêtre de dialogue avec les infos de victoire	boutons actifs pas de fenêtre	boutons inactifs fenêtre ouverte	ok
void demanderJouerTour()	vérifie que le tour peut se jouer	A.Le score max est atteint B.Le score max n'est pas atteint	A.pas de tour joué B.tour joué	ok

Version v5

1. **Diagramme état-transition**



2. **Dictionnaire**
   a. ***Dictionnaire d'état***

nomEtat	*Signification*
`accueil`	La partie n'a pas débuté ou le score max a été dépassé par l'un des deux joueurs
`enJeu`	La partie est en cours
`pause`	La partie est en pause et le chrono arrête de s'écouler

b. Dictionnaire d'événement

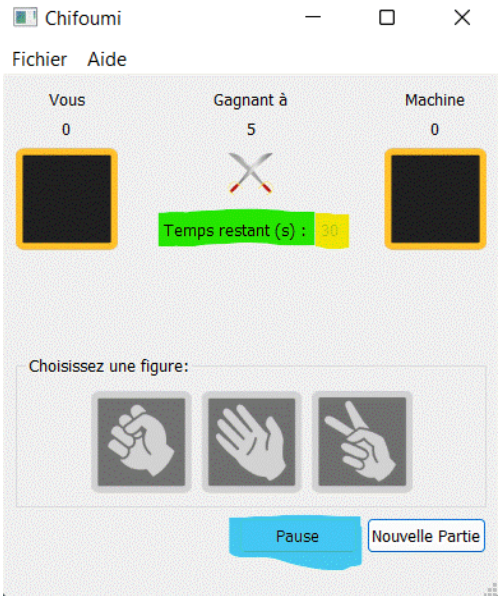
Signification			
	accueil	**enJeu**	**en pause**
NouvellePartie	lance la partie	relance une partie	relance une partie
DemanderPause	X	le jeu s'arrête	le jeu reprend
afficher fin score	affiche le score une fois la partie finie	X	X

3. Diagramme état transition version matricielle

	accueil	enJeu	pause
lance l'application	1	0	0
nouvelle partie	0	1	0
afficher fin score	1	0	0
demander Pause (état en jeu)	0	0	1
demander Pause (état en pause)	0	1	0

4. Nouveaux éléments d'interface

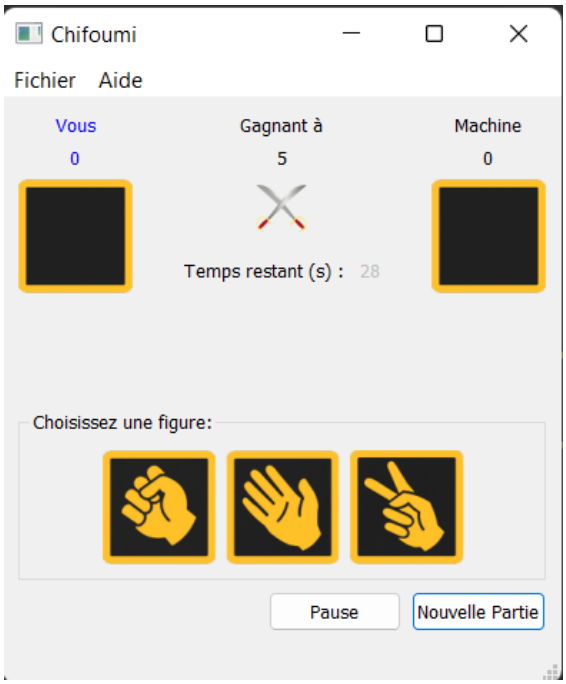
application en mode **accueil** :



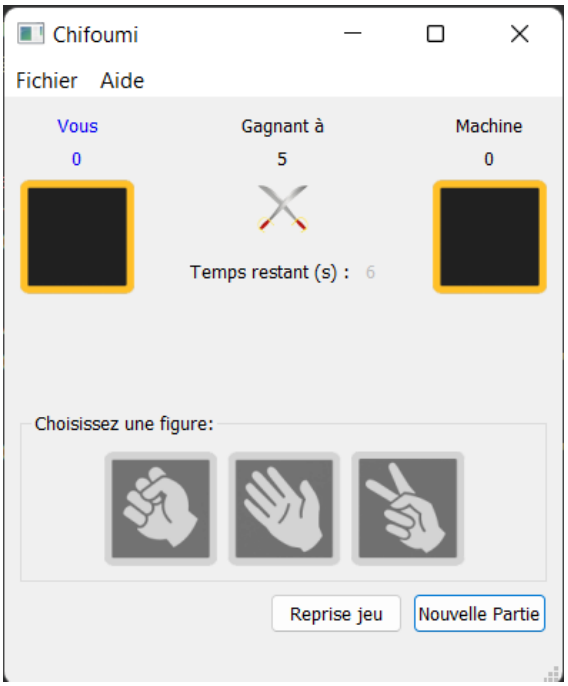
label : texte fixe qui informe le joueur sur l'utilité de labelTimeInt.
labelTimeInt: Nombre (30 par défaut) qui indique le temps restant en seconde.

bPause : Bouton qui permet d'appeler la fonction demanderPause(). Il contient le texte "Pause" quand l'application est en mode **enJeu** ou **accueil** et il contient le texte "Reprise jeu" quand l'application est en mode **pause**.

application en mode **enJeu** :



application en mode **pause** :



5. Présentation des fichiers sources

main.cpp : créer le modèle mvp et lance l'application
chifoumiVue.cpp : affiche les éléments d'interface à l'utilisateur et réceptionner ses intéractions avec les boutons
modele : stocke les données de l'objet modèle
presentation.cpp : lie la vue et le modèle

6. Présentation des nouvelles procédures

Dans presentation.h :

void update() : décrémente le QTimer et l'affiche à l'écran. S'occupe également de la fin de partie si le temps atteint 0.

void demanderPause() : inverse l'état entre **pause** et **enJeu** et demarre ou stop le chronomètre en fonction.

Dans chifoumiVue.h

void afficherFinScore(int, QString) : La fenêtre affiche maintenant le temps restant et un message différent pour chaque fin possible.

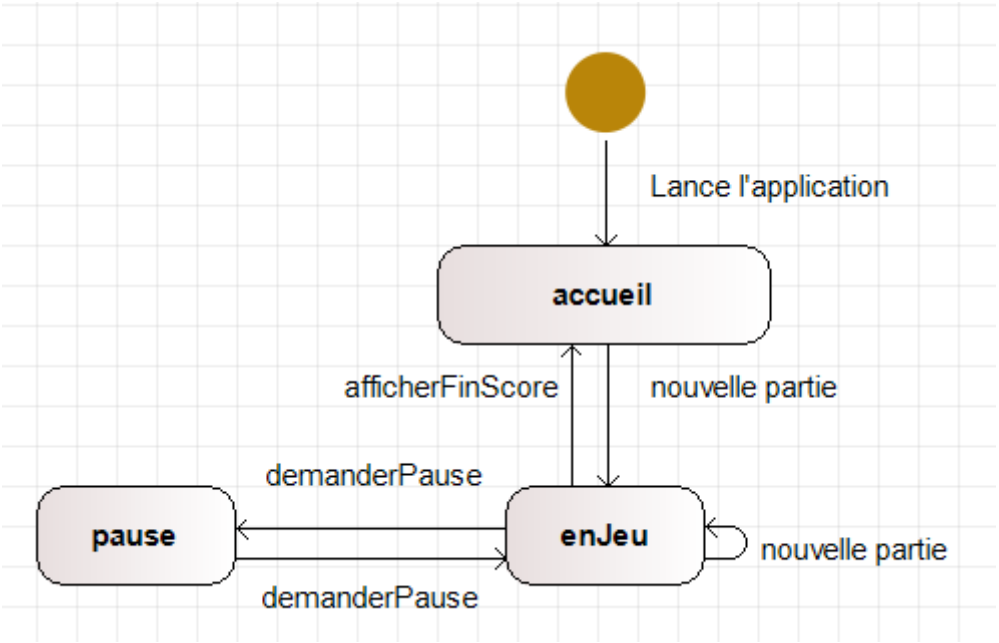
7. Résultat test

Classe	Description	valeur en entrée	valeur en sortie	résultat
void update()	test de l'écoulement du temps en mode enJeu	tempsRestant = 30	tempsRestant = 29	ok

void demanderPause()	passage de **enJeu** à **pause** et arrêt du chrono	etat = enJeu chrono = start	etat = pause chrono = stop	ok
void afficherFinScore(int , QString)	affiche une fenêtre de fin.	scoreJoueur = 5 scoreMachine = 0 tempsRestant = 10	affiche les 3 variables et le gagnant est Joueur	ok
		scoreJoueur = 3 scoreMachine = 0 tempsRestant = 0	affiche que le temps est terminé, gagnant joueur avec 3 points	ok
		score Joueur = 3 scoreMachine = 3 tempsRestant = 0	affiche une égalité de 3 points	ok

Version v6

1. **Diagramme état transition**



Pas de changement car seule différence est le paramétrage de la partie

2. Dictionnaire état événement

a. Dictionnaire d'état

nomEtat	*Signification*
`accueil`	La partie n'a pas débuté ou le score max a été dépassé par l'un des deux joueurs
`enJeu`	La partie est en cours
`pause`	La partie est en pause et le chrono arrête de s'écouler

b. Dictionnaire d'événement

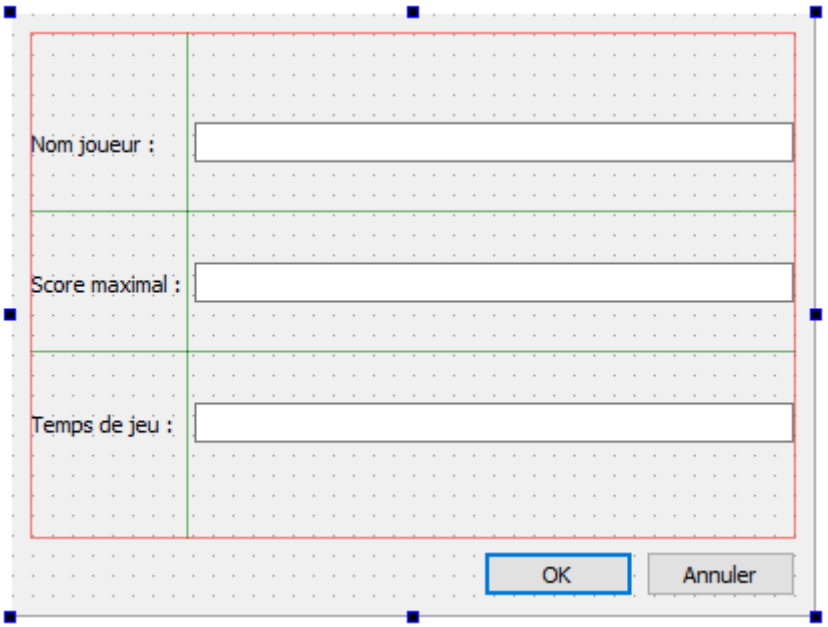
Signification			
	accueil	**enJeu**	**en pause**
NouvellePartie	lance la partie	relance une partie	relance une partie
DemanderPause	X	le jeu s'arrête	le jeu reprend
afficher fin score	affiche le score une fois la partie finie	X	X

3. Diagramme matriciel

	accueil	enJeu	pause
lance l'application	1	0	0
nouvelle partie	0	1	0
afficher fin score	1	0	0
demander Pause (état en jeu)	0	0	1
demander Pause (état en pause)	0	1	0

4. Description nouveaux éléments d'interfaces:

Cette version contient maintenant un fichier "dialog.ui" qui correspond aux paramètres de la partie saisi par le joueur avec son nom, son meilleur score ainsi que la durée de sa partie



5. Liste des fichiers sources

chifoumivue.cpp: gère l'interface avec laquelle le joueur interagit
dialog.cpp: gère les informations saisi du joueur
main.cpp: exécute le programme complet du jeu
modele.cpp: Gère les commandes réalisés par l'app
presentation.cpp: Valide les entrées de l'utilisateur et ce à quoi elles correspondent dans le modèle

6. Fichiers .h modifiés ou créer

chifoumivue.h: declare les différentes classes avec lesquels le joueur va intéragir

-void parametrerPartie(Qstring="vous",int=5,int=30)
cette fonction sert à parametrer le jeu avec le nom du joueur, le temps maximum, ainsi que le score maximal

dialog.h: déclare la nouvelle fenêtre affichant les statistiques du joueur et sa partie

-QString getNom() reçoit le nom du joueur saisi par ce dernier
int getScore() reçoit le score max que le joueur veut atteindre
int getTemps()reçoit le temps maximum de la partie saisi par le joueur

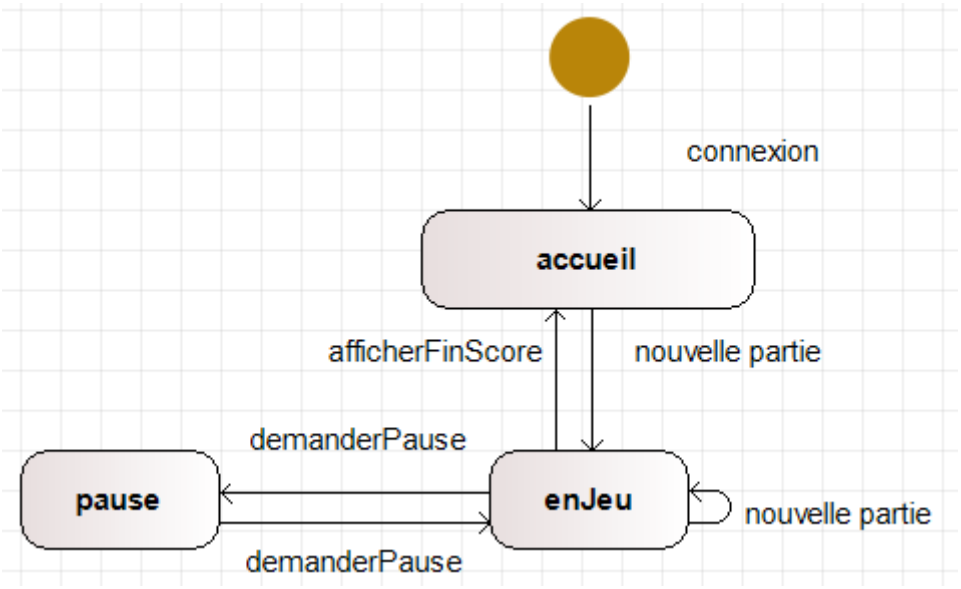
presentation.h: déclare les demandes passer par la vue vers le modèle
-void demanderOuvrirParameres() demande au modèle d'ouvrir la boîte de dialogue des parametres

7. Résultat test

Classe	Description	valeur en entrée	valeur en sortie	résultat
getNom()	Récupère le nom saisi par le joueur	Joueur	NomSaisi	ok
getScore()	récupère le score saisi par le joueur en nouveau score max à atteindre	5	25	ok
getTemps()	récupère le temps saisi par le joueur en nouveau temps maximum	30	60	ok

Version v7

1. Diagramme état transition



2. Dictionnaire état événement

a. Dictionnaire d'état

nomEtat	*Signification*
`accueil`	La partie n'a pas débuté ou le score max a été dépassé par l'un des deux joueurs
`enJeu`	La partie est en cours
`pause`	La partie est en pause et le chrono arrête de s'écouler

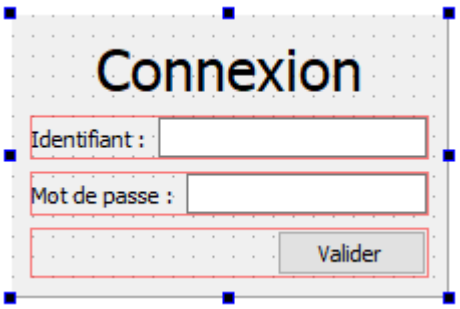
b. Dictionnaire d'événement

Signification				
	initial	**accueil**	**enJeu**	**en pause**
NouvellePartie	X	lance la partie	relance une partie	relance une partie
DemanderPause	X	X	le jeu s'arrête	le jeu reprend
afficher fin score	X	affiche le score une fois la partie finie	X	X
Connexion	connexion avec nom mdp	X	X	X

3. Diagramme matriciel

	accueil	enJeu	pause
connexion du joueur	1	0	0
nouvelle partie	0	1	0
afficher fin score	1	0	0
demander Pause (état en jeu)	0	0	1
demander Pause (état en pause)	0	1	0

4. Description nouveaux éléments d'interfaces:



Cette version contient maintenant un fichier "connexion.ui" qui correspond au éléments nécessaires au joueur pour accéder au jeu avec son identifiant

5. Liste des fichiers sources

chifoumivue.cpp: gère l'interface avec laquelle le joueur interagit
dialog.cpp: gère les informations saisi du joueur
main.cpp: exécute le programme complet du jeu
modele.cpp: Gère les commandes réalisés par l'app
presentation.cpp: Valide les entrées de l'utilisateur et ce à quoi elles correspondent dans le modèle
connexion.cpp: vérifie la correspondance entre les infos de connexion saisit et celle stockée en BD

6. Fichiers .h modifiés ou créer

Connexion.h: déclare les setter et getter nécessaire à la connexion par une base de donnée

QString getId(): retourne l'identifiant de l'utilisateur
QString getMDP():retourne le MDP de l'utilisateur
void setMdp(Qstring): configure le mdp de l'utilisateur
void setId(QString): configure l'identifiant

SLOT:

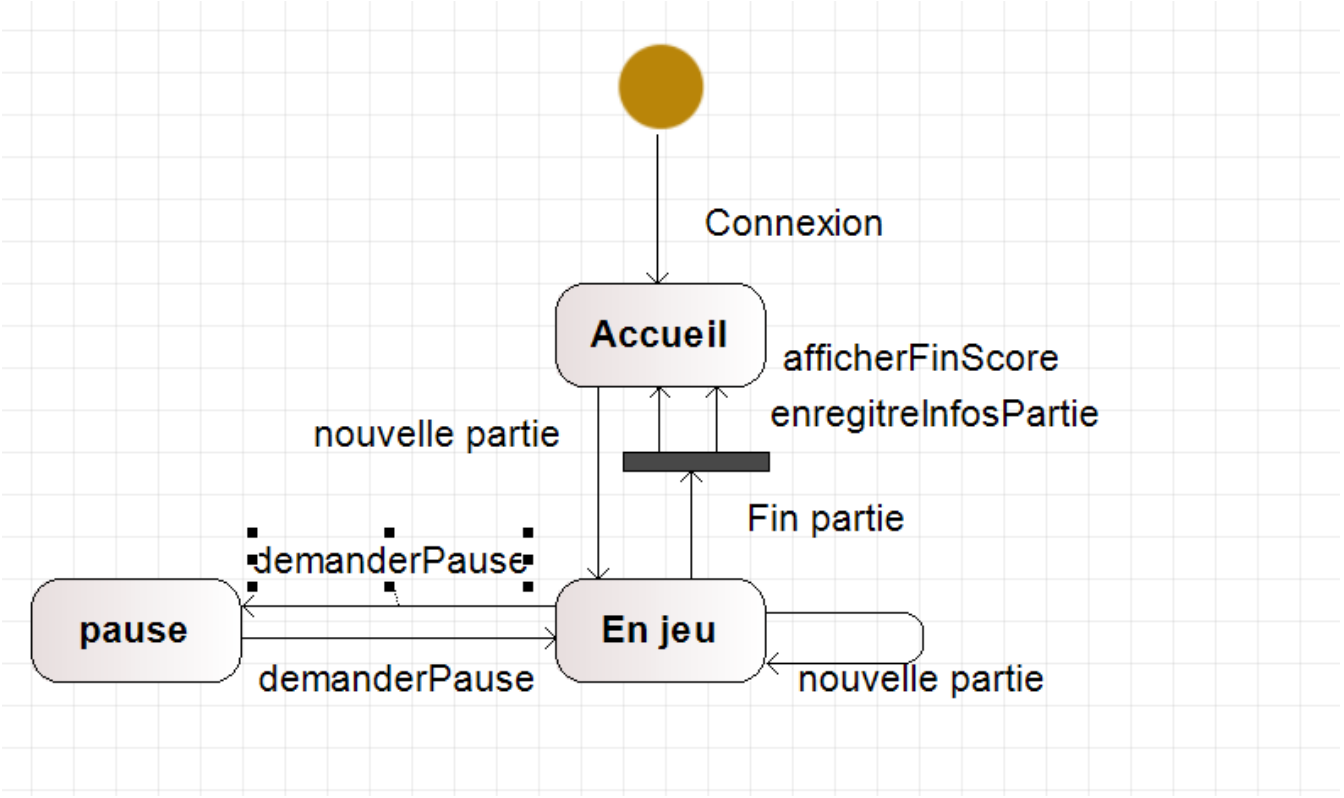
void valider() vérifie l'identifiant et le mot de passe saisi par l'utilisateur au SIGNAL clicked du bouton bValider présent dans connexion.ui

8. Résultat test

Classe	Description	valeur en entrée	valeur en sortie	résultat
valider()	entre un id incorrecte et mdp incorrecte	id saisi mdp saisi	erreur	non ok
valider()	entre un id correct et mdp incorrecte	id saisi mdp saisi	erreur	non ok
valider()	entre un id incorrecte et mdp correct	id saisi mdp saisi	erreur	non ok
valider()	entre un id correct et mdp correct	id saisi mdp saisi	connexion	ok

Version v8

7. Diagramme état transition



8. Dictionnaire état événement

a. Dictionnaire d'état

nomEtat	*Signification*
`accueil`	La partie n'a pas débuté ou le score max a été dépassé par l'un des deux joueurs
`enJeu`	La partie est en cours
`pause`	La partie est en pause et le chrono arrête de s'écouler

b. Dictionnaire d'événement

Signification				
	initial	**accueil**	**enJeu**	**en pause**
NouvellePartie	X	lance la partie	relance une partie	relance une partie
DemanderPause	X	X	le jeu s'arrête	le jeu reprend
afficher fin score	X	affiche le score une fois la partie finie	X	X
Connexion	connexion avec nom mdp	X	X	X
EnregistreInfosPartie	X	enregistre les infos de la partie à la fin de cette derniere	X	X

9. Diagramme matriciel

	accueil	enJeu	pause
connexion du joueur	1	0	0
nouvelle partie	0	1	0
afficher fin score	1	0	0
demander Pause (état en jeu)	0	0	1
demander Pause (état en pause)	0	1	0
enregistement info	1	0	0

10. Liste des fichiers sources

chifoumivue.cpp: gère l'interface avec laquelle le joueur interagit
dialog.cpp: gère les informations saisi du joueur
main.cpp: exécute le programme complet du jeu
modele.cpp: Gère les commandes réalisés par l'app
presentation.cpp: Valide les entrées de l'utilisateur et ce à quoi elles correspondent dans le modèle
connexion.cpp: vérifie la correspondance entre les infos de connexion saisit et celle stockée en BD

11. Fichiers .h modifiés ou créer

Modele.h: initialise les classes manipulées par l'app

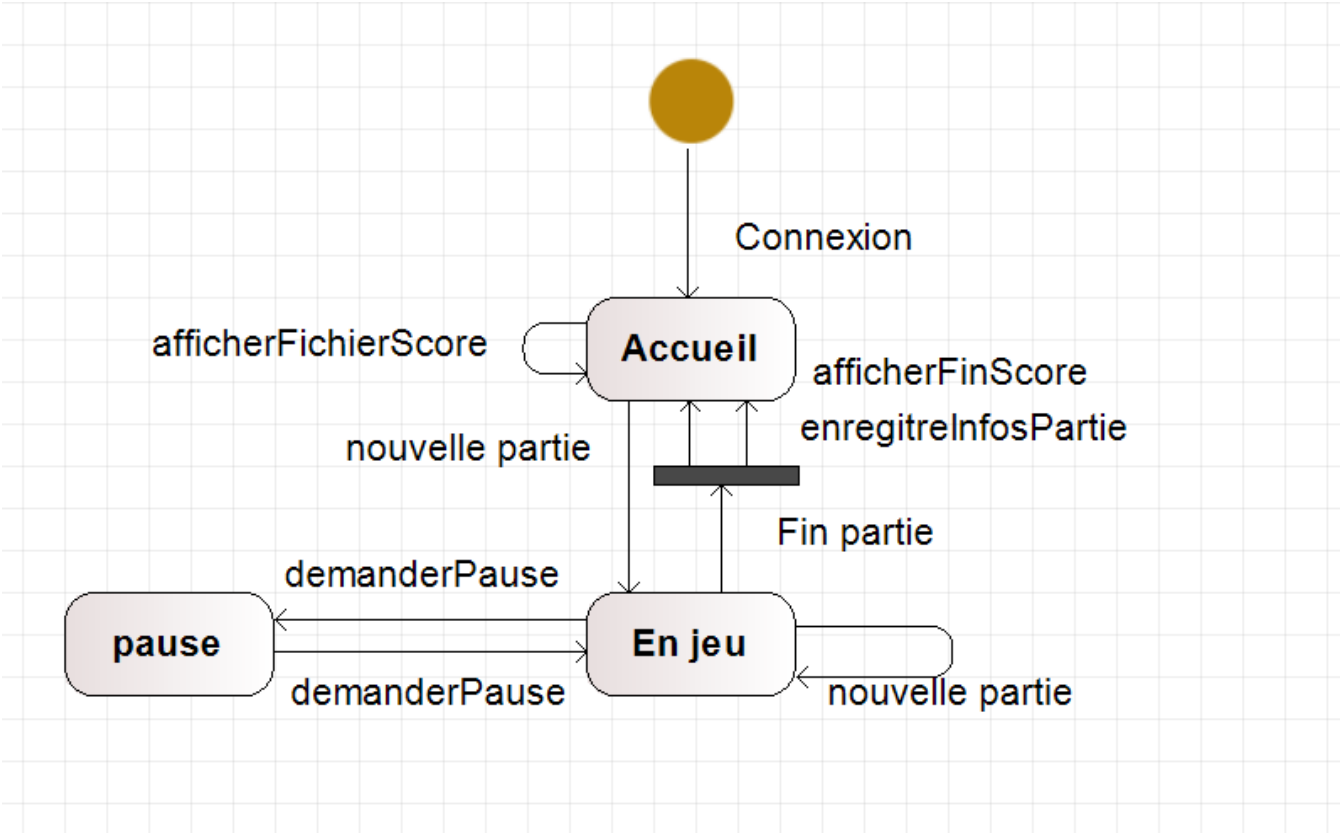
void enregistrerPartie(const int&); permet d'enregistrer les informations de la partie

9. Résultat test

Classe	Description	valeur en entrée	valeur en sortie	résultat
enregistrerPartie(const int&)	enregistre les données de la partie a la fin de cette dernière	base de données vides	base de données avec valeur	ok

Version v9

12. Diagramme état transition



13. Dictionnaire état événement

a. Dictionnaire d'état

nomEtat	*Signification*
`accueil`	La partie n'a pas débuté ou le score max a été dépassé par l'un des deux joueurs
`enJeu`	La partie est en cours
`pause`	La partie est en pause et le chrono arrête de s'écouler

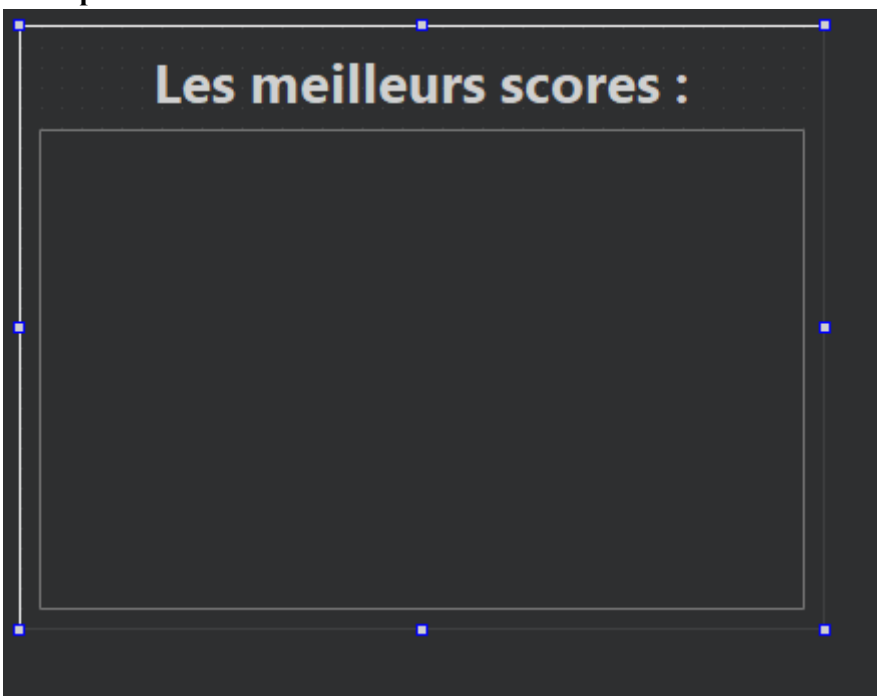
b. Dictionnaire d'événement

Signification				
	initial	**accueil**	**enJeu**	**en pause**
NouvellePartie	X	lance la partie	relance une partie	relance une partie
DemanderPause	X	X	le jeu s'arrête	le jeu reprend
afficher fin score	X	affiche le score une fois la partie finie	X	X
Connexion	connexion avec nom mdp	X	X	X
EnregistreInfosPartie	X	enregistre les infos de la partie à la fin de cette derniere	X	X
AfficherFichierScore	X	affiche les 10 meilleurs scores du joueur	X	X

14. Diagramme matriciel

	accueil	enJeu	pause
connexion du joueur	1	0	0
nouvelle partie	0	1	0
afficher fin score	1	0	0
demander Pause (état en jeu)	0	0	1
demander Pause (état en pause)	0	1	0
enregistement info	1	0	0
afficher fichier score	1	0	0

15. Description nouveaux éléments d'interfaces:



Cette version contient maintenant un fichier "connexion.ui" qui correspond au éléments nécessaires au joueur pour accéder au jeu avec son identifiant

16. Liste des fichiers sources

chifoumivue.cpp: gère l'interface avec laquelle le joueur interagit
dialog.cpp: gère les informations saisi du joueur
main.cpp: exécute le programme complet du jeu
modele.cpp: Gère les commandes réalisés par l'app
presentation.cpp: Valide les entrées de l'utilisateur et ce à quoi elles correspondent dans le modèle
connexion.cpp: vérifie la correspondance entre les infos de connexion saisit et celle stockée en BD
resultats.cpp: gère l'affichage des 10 meilleurs score obtenus face à la machine

17. Fichiers .h modifiés ou créer

resultat.h: s'occupe de l'affichage des 10 meilleurs scores obtenus en créant un resultat.ui contenant un tableWidget et une connexion à la base de données. resultat.h n'a pas de fonction ou de procédure car tout est développer dans le constructeur (aucune interaction avec l'utilisateur n'est attendu).

presentation.h: initialise les demandes entre l'interface utilisateur et l'application.

SLOT: void demanderOuvrirResultats(): ouvre une boite de dialogue contenant les 10 meilleurs résultats lorsque le SIGNAL trigerred est exécuté dans chifoumivue.cpp avec l'actionResultat

10. Résultat test

Classe	Description	valeur en entrée	valeur en sortie	résultat
demanderOuvrirResultat()	demande l'ouverture du fichier contenant les 10 meilleurs scores	fenêtre fermée	fenêtre ouverte avec score affiché	ok