Saé 2.01 – Développement d'une application

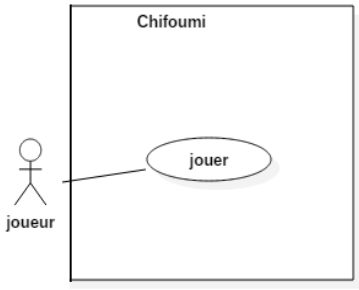
Chifoumi – Dossier d'Analyse et conception

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

2. Diagramme des Cas d'Utilisation



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*

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.

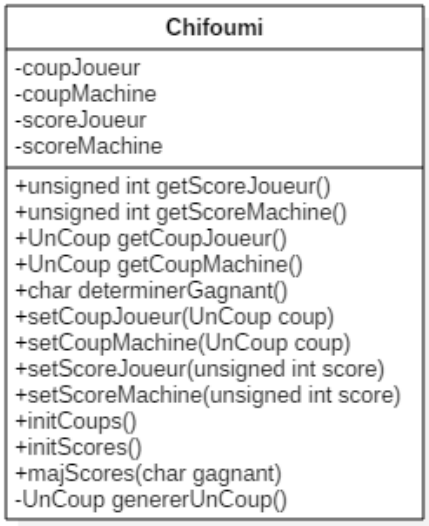


Figure 2 : Diagramme de Classes UML du jeu Chifoumi

(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 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.

4

Version v0

5. Implémentation et tests

5.1 Implémentation

Liste des fichiers de cette version :
- chifoumi.h : Interface de la classe chifoumi
- chifoumi.cpp : Corps de la classe chifoumi
- main.cpp : Programme de test de la classe chifoumi

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

5.2 Test

Méthode get associé au 'score'	***OK***
Méthode get associé au 'coup'	***OK***
Méthode setScore	***OK***
Méthode initScore	***OK***
Méthode initCoups	***OK***
Méthode setCoupJoueur	***OK***
Méthode setCoupMachine	***OK***
Méthode determinerGagnant	***OK***
Méthode majScores	***OK***

		Machine		
Joueur		***Pierre***	***Papier***	***Ciseau***
	Pierre	***Égalité***	***Machine gagne***	***Joueur gagne***
	Papier	***Joueur gagne***	***Égalité***	***Machine gagne***
	Ciseau	***Machine gagne***	***Joueur gagne***	***Égalité***

Test avec le programme fourni main.cpp

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

Version v1

6. Classe Chifoumi : Diagramme états-transitions

(a) Diagramme états-transitions -actions du jeu

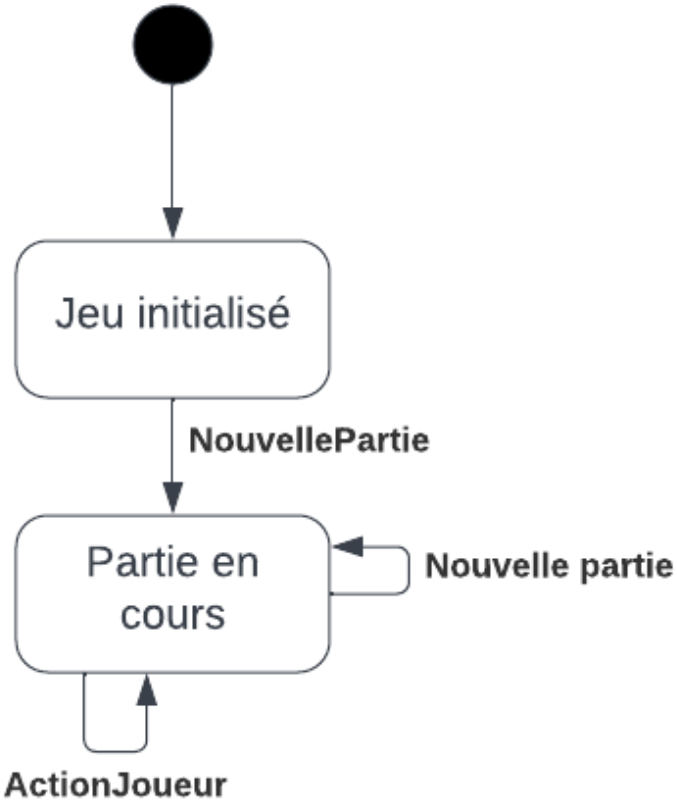


Figure 4 : Diagramme états-transitions

(b) Dictionnaires des états, événements et Actions

Dictionnaire des états du jeu

nomEtat	*Signification*
`Jeu initialisé`	L'application attend que l'utilisateur lance la partie
`Partie en cours`	L'utilisateur a lancé la partie en cliquant sur le bouton nouvelle partie. Ensuite, l'utilisateur et la machine jouent tour à tour jusqu'à arrêt de l'application ou lancement d'une nouvelle partie.

Tableau 3 : États du jeu

Dictionnaire des événements faisant changer le jeu d'état

nomEvénement	*Signification*
`Nouvelle partie`	Le joueur appuie sur le bouton nouvelle partie et lance une nouvelle partie
`ActionJoueur`	Le joueur choisit un signe

Tableau 4 : Evénements faisant changer le jeu d'état

Description des actions réalisées lors de la traversée des transitions

`Jeu initialisé -> Partie en cours (NouvellePartie)`	Les scores sont mis à 0 et les coups sont initialisés en rien. Les boutons sont activés et prêt à être sélectionné par le joueur.
`Partie en cours -> Partie en cours (NouvellePartie)`	Les scores sont remis à 0 et les coups initialisés à rien. Les boutons sont réactivés et prêt à être sélectionné par le joueur.
`Partie en cours -> Partie en cours (ActionJoueur)`	La machine effectue à son tour une action. On compare le coup du joueur et de la machine.

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

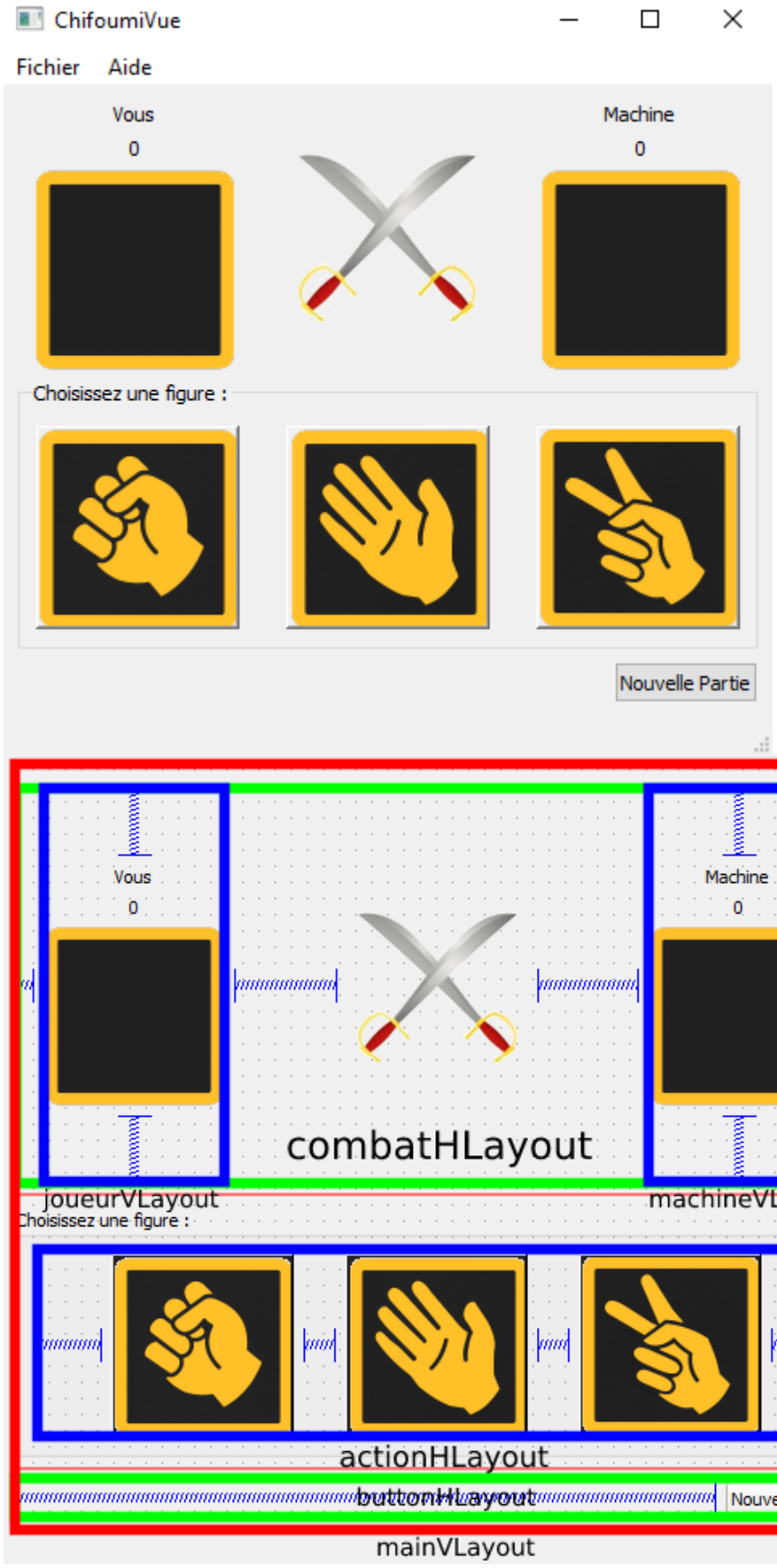
(c) Préparation au codage :

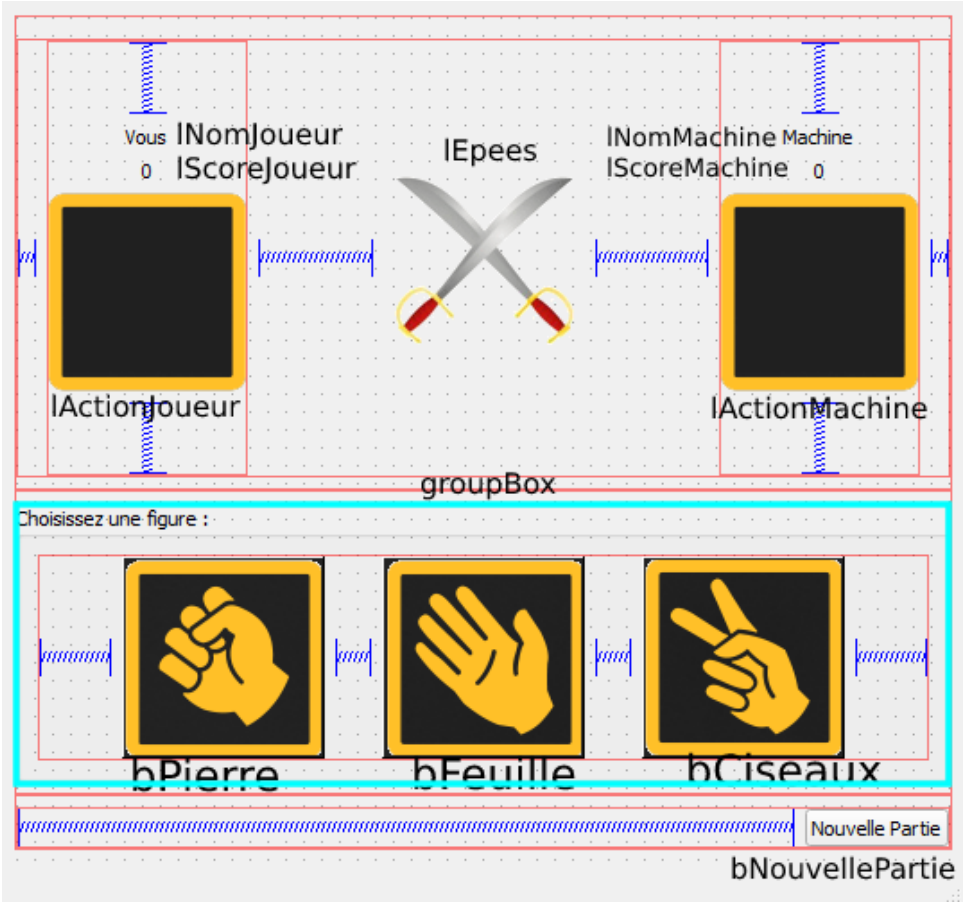
Table `T_EtatsEvenementsJeu` correspondant à la version matricielle du diagramme états-transitions du jeu :
- en *ligne : les **événements*** faisant changer le jeu d'état
- en *colonne : les **états*** du jeu

Événement → *nomEtatJeu*	`Nouvelle partie`	`ActionJoueur`
`Jeu initialisé`	`Partie en cours`	`------`
`Partie en cours`	`Jeu initialisé`	`Partie en cours`

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

7. Éléments d'interface





L'interface est séparée en 2 parties :

-Une partie qui affiche l'état de la partie

-L'autre partie permet au joueur d'effectuer une action ou de lancer une nouvelle partie

La partie du haut est elle-même séparée en 2 parties, la partie joueur et la partie machine.

Dans chacune de ces parties on peut voir l'action jouée par le joueur et la machine ainsi que leurs scores.

La partie du bas est constituée de boutons d'actions (pierre, feuille et ciseaux), une fois que le joueur clique sur un de ces boutons, l'action décrite sur l'image est exécutée.

Le bouton nouvelle partie permet de remettre les scores à 0.

8. Implémentation et tests

8.1 Implémentation

Chifoumi.h interface du modèle du jeu du chifoumi founi en ressource
Chifoumi.cpp corps du modèle du jeu du chifoumi fourni en ressource
Chifoumivue.h interface de la vue de l'application
Chifoumivue.cpp corps de la vue de l'application
Main.cpp programme qui initialise le modèle et la vue du chifoumi

8.2 Test

Quand il clique sur l'action pierre, l'action est bien prise en compte et s'affiche dans la case d'action du joueur. (Idem pour papier et ciseau)
Le bouton nouvelle partie remet tout à 0 et vide les cases d'actions du joueur et de la machine.