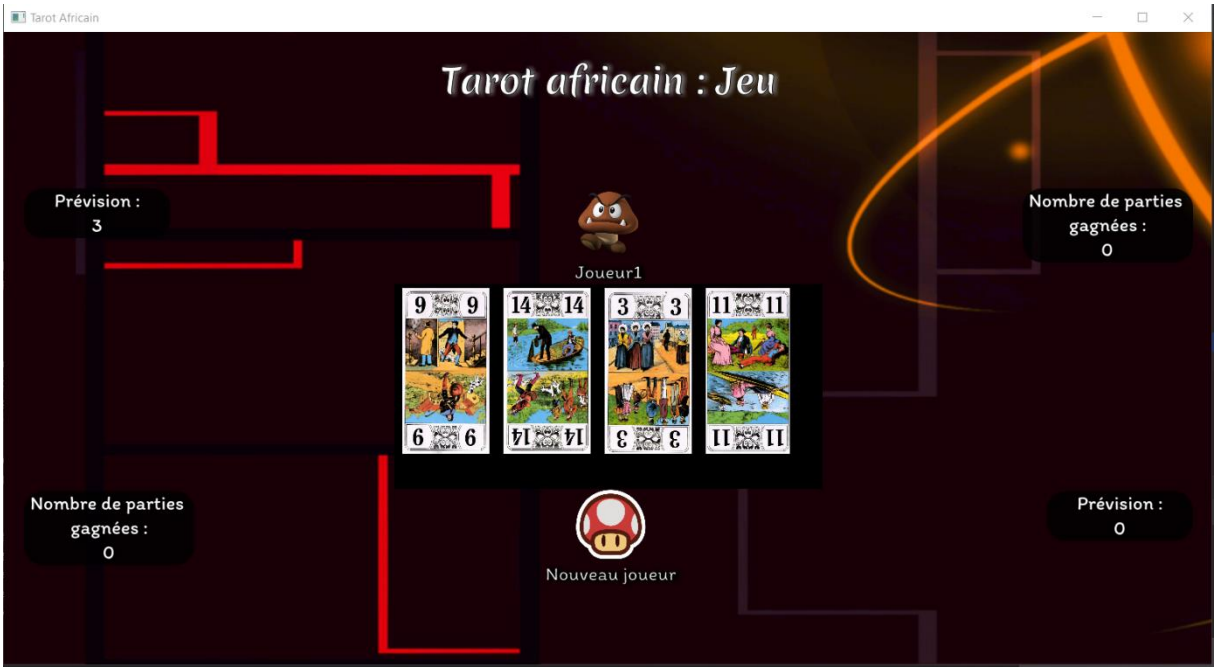
Projet JavaFx 2019
Document de preuves

Le tarot Africain

Membre du groupe :
Chanson Chloé

Sommaire :

I- Description du contexte :

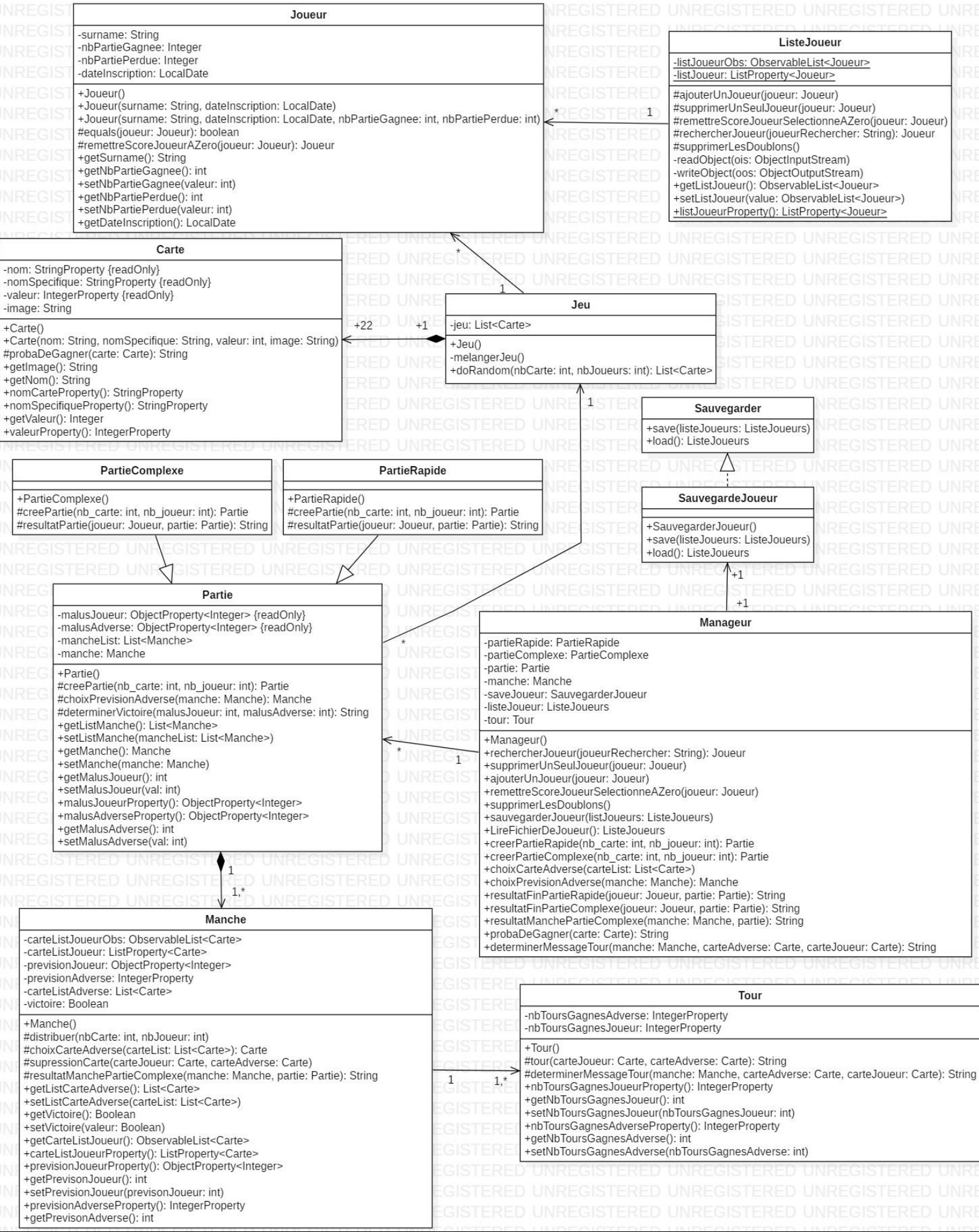


Cette application a pour but de divertir les utilisateurs. C'est un jeu composé des 22 cartes de tarot. Il permet d'utiliser ses facultés d'analyse et de réflexion tout en s'amusant. Ainsi, il faut essayer de lire, et comprendre la façon de jouer de son adversaire afin de gagner.

En résumé, ce jeu demande de bien savoir jauger les situations tout en prenant tous les paramètre en compte. C'est un jeu qui peut se jouer rapidement aussi bien votre pause ou bien que dans un transport en commun. Il peut être joué à tout âge, et sans modération.

Ce jeu est destiné à être joué depuis son ordinateur. En fonction du temps qui vous avez, vous pouvez soit choisir de faire une partie rapide ou bien une partie dite complète. Une partie rapide est une partie qui est composé d'une seule manche. Une partie dite complète et composée de plusieurs manches. A chaque manche nous retirons une carte de la main de chaque joueur jusqu'à qu'ils n'ont plus de carte en mains. C'est alors la fin de la partie. Le gagnant sera celui qui a respecté le plus de fois son contrat, autrement dit celui qui a obtenu le moins de malus (un malus est donné à chaque fois qu'un joueur perd lors d'une partie dite complète). Le gagnant d'une manche est celui qui respecte son contrat c'est-à-dire celui qui réalise le même nombre de partie gagnée que sa prévision. Ce terme de contrat est la base même du jeu. A partir des cartes distribuer au début de chaque manche il faut analyser les cartes afin de savoir combien de partie vous pensez gagner et respecter cette prévision. La particularité est lorsque que les joueurs font leur prévision, le nombre de parties gagnées prédites par les deux joueurs ne doit pas être égale au nombre de cartes dans la main d'un joueur. Sans cette règle il y aurait une possibilité pour que les joueurs respectent tous les deux leurs contrats, et cela ferais perdre tout l'intérêt de ce jeu. Une manche est composée de plusieurs tours (autant de tour qu'un joueur a de carte dans sa main). Le vainqueur d'un tour est celui qui a posé la carte la plus forte.

II- Diagramme de conception :



Le digramme UML précédent constitue une représentation statique des éléments qui composent mon application « Tarot Africain » et de leurs relations.

Mon application est un jeu de carte et elle compte 11 classes différentes qui sont : Carte, Jeu, Joueur, ListeJoueurs, Tour, Manche, Partie, PartieRapide, PartieComplexe, Manageur et SauvegarderJoueur. L'application possède aussi 1 interface qui a pour nom Sauvegarder.

Les classes qui sont au plus bas niveau et qui constituent la base de mon application sont :

- La classe « Carte » qui correspond à l'objet (physique) "carte". Cette classe est composé de 4 attributs et de 8 opérations dont 2 constructeurs. La majorité de ces opérations sont des opérations de type « *get* » et « *set* ». Seul l'opération « *String probaDeGagner (Carte carte)* » permet de réaliser une action « concrète » pour l'application. Cette méthode permet de conseiller le joueur en renvoyant un String qui varie en fonction de la valeur de la carte passée en paramètre. Cette méthode est aussi utilisée pour la prédiction du joueur adverse afin que celle-ci soit réaliste.
- La classe « Tour » qui correspond au moment du jeu ou chaque joueur pose une carte. Cette classe est composé de 2 attributs et de 8 opérations dont 1 constructeur. Tout comme la classe « Carte » celle-ci possède 5 opérations de type « *get* » et « *set* ». Les deux autres opérations sont : « *Manche tour(Manche manche, Carte carteAdverse, Carte carteJoueur)* » qui permet de définir quel joueur est le vainqueur du tour et « *String determinerMessageTour(Manche manche, Carte carteAdverse, Carte carteJoueur)* » qui permet de retourner le message adapter au résultat du tour.

- La classe « Joueur » qui représente une personne qui joue au jeu. Cette classe est composé de 4 attributs et de 11 opérations dont 3 qui sont des constructeurs. Parmi elle 6 opérations sont des opérations de types « *get* » et « *set* ». Nous pouvons remarquer que cette classe ne contient pas de variables observables (contrairement aux deux précédentes). Cela permet de pouvoir sérialiser cette classe. Dans cette classe nous faisons aussi une redéfinition de la méthode « *boolean equals(Joueur joueur)* » afin de pouvoir comparer les joueurs. La méthode « *Joueur remettreScoreJoueurAZero(Joueur joueur)* » permet de remettre les scores (le nombre de parties gagnées et perdues) d'un joueur, a zéro.

Nous avons ensuite les classes de second niveau qui utilise les trois classes précédentes. Respectivement nous avons :

- La classe « Jeu » qui correspond à un jeu de cartes, soit, un ensemble matériel complet des cartes nécessaires pour jouer au « Tarot Africain ». Dans notre cas, nous avons un jeu de composé des 22 cartes de tarot. Cette classe ne possède que 3 opérations dont 1 constructeur. Les deux autres méthodes « *void melangerJeu()* » et « *List<Carte> doRandom(int nbCarte, int nbJoueurs)* » permettent respectivement de mélanger le jeu avant la distribution des cartes et de sélectionner les cartes qui vont être utilisées pour une manche.

- La classe « Manche » un "morceau" d'un jeu... Une manche est composée de plusieurs tours. Cette classe est composé de 6 attributs et de 16 opérations. Parmi elle il y a 1 constructeur et de 11 opérations de types « *get* » et « *set* ». Les autres opérations sont « *void supressionCarte (Carte carteJoueur, Carte carteAdverse)* » qui permet la suppression de la carte du joueur et du joueur adverse qui vient d'être jouée durant un tour (afin que les joueurs ne jouent pas deux fois la même carte dans une manche). Il y ensuite l'opération

« *Carte choixCarteAdverse(List<Carte> carteList)* » qui permet à l'application de choisir la carte que le joueur adverse joue lors d'un tour. L'opération « *void distribuer(int nbCarte, int nbJoueur)* » qui permet de distribuer les cartes aux deux joueurs. Et enfin l'opération « *String resultatManchePartieComplexe(Manche manche, Partie partie)* » qui permet de déterminer le résultat d'une manche lors d'une partie dite complexe.

- La classe « ListeJoueurs » qui est une classe qui est sérialisable. Elle correspond à l'ensemble des joueurs qui ont joué. Cette classe est composée de 2 attributs statique. Cette classe est composée de 10 opérations dont 3 opérations de types « *get* » et « *set* ». Les opérations « *void readObject(final ObjectInputStream ois)* » et « *void writeObject(final ObjectOutputStream oos)* » permettent de redéfinir les méthodes du même nom pour la sérialisation. Ensuite les autres opérations permettent de réaliser différentes actions sur la liste de joueurs comme ajouter, supprimer, rechercher, remettre les scores d'un joueur,

Il y a ensuite la classe « Partie » et ses deux classes filles « PartieRapide » et « PartieComplexe ».

- La classe « Partie » est composer de 4 attributs et de 14 opérations dont 1 constructeur et de 10 opérations de types « *get* » et « *set* ». Les opérations « *Partie creerPartie(int nb carte, int nb joueur)* », « *Manche choixPrevisionAdverse(Manche manche)* » et « *String determinerVictoire(int malusJoueur, int malusAdverse)* » permettent respectivement de créer une partie, de choisir une prévision pour le joueur adverse et de déterminer qui a gagné une partie.
- Les classes « PartieRapide » et « PartieComplexe » qui héritent de la classe « Partie ». C'est deux classes permettent de redéfinir des comportements particuliers pour chaque « type » de partie. Ainsi c'est deux classes possèdent les classes « *Partie creerPartie(int nb carte, int nb joueur)* » et « *String resultatPartie(Joueur joueur, Partie partie)* ».

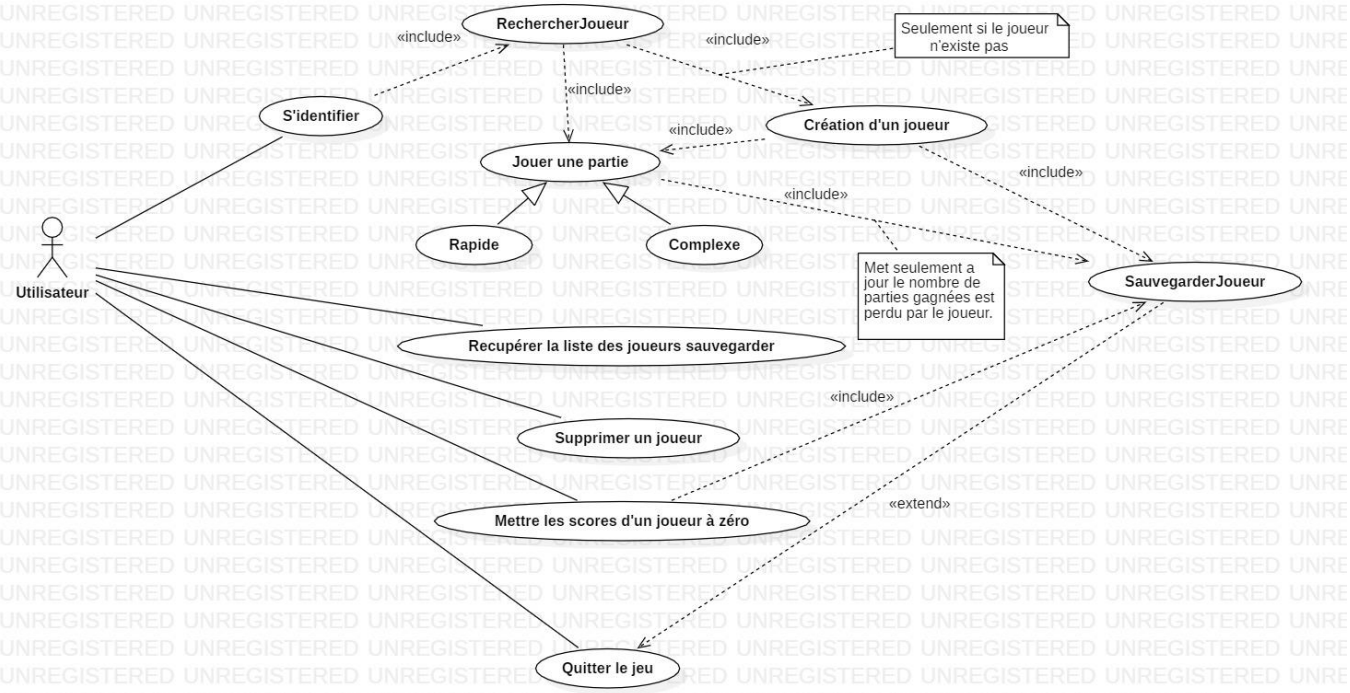
Nous avons ensuite l'interface « Sauvegarder ». Cette interface définie des méthodes destinées à la sauvegarde.

La classe "SauvegarderJoueur" quant-a-elle, permet de gérer la sérialisation de la liste de joueur. Cette classe implémente l'interface "Sauvegarder". Cette classe possède 3 opérations dont 1 constructeur et deux méthodes de l'interface « *void save(ListeJoueurs listJoueurs)* » et « *ListeJoueurs load()* ».

Pour finir il y a la classe « Manageur ». Cette classe correspond à une « Façade ». En effet, elle permet de faciliter l'accès au système de classe de l'application. De plus cette classe permet une meilleure encapsulation des données. Elle appelle les différentes méthodes de l'application afin que l'application ne puisse pas accéder directement à ces méthodes. Ainsi cette classe compte 17 opérations dont 1 constructeur est 7 attributs (attributs qui permettent d'accéder aux différentes méthode).

Les différentes classes de cette application possède de nombreuses opérations de types « *get* » et « *set* ». Cela permet de mettre en place une bonne encapsulation des classes.

IV- Diagramme de cas d'utilisation :



V- Description du diagramme de cas d'utilisation :

Un utilisateur peut s'identifier. Lors de cette identifiant une recherche du joueur se fait automatiquement réaliser dans la liste de joueurs de la classe « ListeJoueur ». Cela a pour objectif de permettre à un joueur de continuer de jouer avec un pseudo qu'il a déjà utilisé, mais aussi de permettre que deux pseudos identiques de soit pas créé. Si le pseudo utilisé par le joueur existe déjà et qu'il le sélectionne pour jouer alors, il n'y a pas de nouveau joueur crée. Dans le cas contraire, un joueur est créé. De là, le joueur est automatiquement redirigé vers la fenêtre jouée. Cette fenêtre varie en fonction du type de partie choisi par le joueur. Une fois la partie terminée, le nombre de parties gagné ou perdu par le joueur est mis à jour puis sauvegardé dans le fichier de sauvegarde.

Un utilisateur peut aussi soit supprimer un joueur ou récupérer la sauvegarde des joueur (afin de retrouver les joueurs supprimés). Cela a pour but de lui permettre de récupérer un joueur qu'il a supprimé par erreur. Il peut aussi mettre les scores d'un joueur à zéro ou quitter le jeu. Ce sont deux actions qui entraînent la sauvegarde automatique de la liste de joueurs. Ainsi pour enregistrer des joueurs que l'utilisateur, il suffit que l'utilisateur quitte normalement le jeu.