

## Guide développeur

### Description générale du code :

Le Kyudo est un sport japonais qui consiste à pratiquer le tir à l'arc traditionnel japonais.

La compétition de Kyudo se déroule de la manière suivante dans le contexte de notre projet : les archers (3 joueurs) se présentent en équipe pour tirer des flèches sur une cible. Le centre de la cible rapporte 5 points, les zones extérieures rapportent 3 points, et les tirs manqués ne rapportent aucun point. Dans un match entre deux équipes, tous les joueurs de l'équipe A tirent une fois, puis les joueurs de l'équipe B tirent une fois. Chaque joueur a 4 tirs, donc il y a 4 sets au total.

### Les includes :

Les bibliothèques standard : `stdio.h`, `stdlib.h`, `string.h`, `stdbool.h`

Les autres bibliothèques sont : `ctype.h` pour manipuler chaîne de caractère de durée qu'on entre comme argument dans la commande, `pthread.h` et `semaphore.h` pour la gestion de threads et de sémaphores, `unistd.h` pour utiliser la fonction `sleep()` sur linux, `windows.h` pour la fonction `Sleep()` sur windows, et `sys/time.h` pour réaliser des tests avec `time` afin de vérifier si tout le match de tour N commence en même temps et pour réaliser le match par rapport à la durée entre en argument.

Le programme contient plusieurs constantes, telles que les tailles maximales du nombre de joueurs par équipe, du nombre de tirs par joueur, du nombre maximum d'équipes, du nombre maximum de matchs et du nombre maximum des tours possibles. Ces constantes sont utilisées pour initialiser les tableaux de données.

### Les structures :

Le programme contient quatre structures : `Joueur`, `Équipe`, `Match` et `MatchArgs`.

Structure `Joueur` contient les informations générales sur les joueurs tels que : `x` c'est le numéro de joueur (cela peut être remplacé par `nom_joueur`), le score de joueur, et le nombre des tirs qu'il dispose.

Structure `Équipe` contient le nom d'équipe, le score total d'équipe, et le tableau des joueurs.

Pour structure Match on a 3 tableau, table equipe1 et equipe2 pour réaliser la première tour, table vainqueur pour stocker les gagnants et puis faire des match entre eux, ensuite on a start\_time et end\_time pour réaliser les affichages de début et la fin du match et duree\_match pour réaliser les match par rapport à sa. Enfin on a les variables num\_tour et num\_match.

MatchArgs est une structure qui est utilisée pour passer les valeurs du structure Match dans la fonction simuler\_match, on ne peut pas passer directement la structure Match comme paramètres pour simuler\_match car simuler\_match est passée en paramètres pour pthread.

### **Les fonctions:**

Le programme est composé de 12 fonctions:

- ❖ **bool est\_un\_fichier(char \*fichier) :**  
Vérifie si un fichier existe ou non en essayant de l'ouvrir en mode lecture. Elle renvoie **true** si le fichier existe et **false** sinon. Il est utilisé pour vérifier si le fichier passé en argument dans la "command line" est un fichier.
- ❖ **bool est\_time(char \*arg) :**  
Vérifie si une chaîne de caractères passée en paramètre est un nombre entier positif en utilisant la fonction **isdigit()** de la bibliothèque **ctype.h**. Si la chaîne de caractères est un entier positif, la fonction renvoie **true**. Sinon, elle renvoie **false**. Il est utilisé pour vérifier le format de duree\_de\_match qui est passé en argument dans la "command line".
- ❖ **int get\_num\_equipe(char\* fichier) :**  
Calcule le nombre d'équipes dans un fichier en comptant le nombre de lignes dans le fichier. Elle ouvre le fichier en mode lecture, compte le nombre de lignes, puis ferme le fichier et renvoie le nombre d'équipes.
- ❖ **int VerifierPuissanceDeDeux(char\* fichier) :**  
Vérifie si le nombre d'équipes dans le fichier est une puissance de deux en utilisant un opérateur de bit et une opération logique. Si le nombre d'équipes est une puissance de deux, la fonction renvoie **true**. Sinon, elle renvoie **false**.
- ❖ **int get\_num\_match(int num\_equipe) :**  
Calcule le nombre de matchs pour un nombre donné d'équipes en divisant le nombre d'équipes par deux. Il est utilisé pour avoir le nombre de match total par rapport au nombre d'équipes.

- ❖ **void initEquipesEtMatches(char\* fichier, int num\_equipe, Equipe \*equipes) :**  
Initialise les équipes pour le tournoi en lisant le nom de chaque équipe à partir d'un fichier (fichier par défaut ou fichier passé en argument) et en créant un tableau de structures **Equipe** contenant les noms d'équipes et les scores des équipes.
- ❖ **void simuler\_tir(Joueur \*joueur) :**  
Simuler un tir et modifier les données d'un joueur en conséquence. La fonction utilise la fonction **rand()** pour générer un nombre aléatoire entre 0 et 99:
  - ❖ Si le nombre aléatoire est inférieur à 20 alors le joueur a touché le centre (5pts).
  - ❖ Si le nombre aléatoire est entre 20 et 80 alors le joueur a touché le bord (3pts). Sinon, le joueur n'a pas touché la cible(0pts).
 Une fois "touché la cible", le score du joueur est ensuite mis à jour.
- ❖ **void calculer\_score(Equipe \*equipes) :**  
Calcule le score total pour chaque équipe en parcourant tous les joueurs dans chaque équipe et en sommant les scores des joueurs.
- ❖ **void \*simuler\_match(void \*arg) :**  
La fonction **simuler\_match** est responsable de la simulation d'un seul match. Elle prend un pointeur vers une structure **Match** en tant qu'argument et utilise cette structure pour simuler un match entre deux équipes (equipe1 et equipe2). Cette fonction utilise la fonction **simuler\_tir** pour simuler les tirs de chaque joueur de chaque équipe. À la fin du match, cette fonction calcule le score de chaque équipe et détermine le vainqueur. Elle utilise un mutex pour garantir que les résultats du match sont écrits dans le fichier **match\_results.txt** et **afficher** sans problèmes.
- ❖ **void roundOne(int num\_matches, int num\_rounds, pthread\_t threads[NB\_TOUR\_MAX][NB\_MATCH\_MAX], Equipe \*equipes, Match match) :**  
La fonction **roundOne** est responsable de la simulation de la première round du tournoi. Elle prend le nombre de matchs à simuler, le nombre de tours dans le tournoi, un tableau de pointeurs vers des structures **Equipe**, et une structure **Match** en tant qu'arguments. Elle utilise des threads pour exécuter simultanément plusieurs appels à la fonction **simuler\_match** pour simuler tous les matchs en même temps. Elle utilise des sémaphores pour synchroniser le début et la fin de chaque tour pour chaque match.

- ❖ `void rounds(int num_equipe, int num_matches, int num_rounds, pthread_t threads[NB_TOUR_MAX][NB_MATCH_MAX], Equipe *equipes, Equipe *vainqueurs, Match match) :`

La fonction `round` simule les autres rounds du tournoi. Cette fonction fait la même chose que la fonction `roundOne` sauf qu'elle réalise une boucle tout les tours, et elle utilise notamment la table vainqueur qui est mise à jour à chaque début de tour avec les gagnants qui n'ont pas été éliminés.

- ❖ `void existe_fichier(void) :`

Vérifier si le fichier texte "**match\_results.txt**" est rempli ou non. Si elle est remplie, le fichier sera supprimé, sinon il ne se passera rien.

### Main :

Analyse des arguments de ligne de commande La première partie du code analyse les arguments de ligne de commande pour déterminer le nom du fichier de données de tournoi et la durée des matchs. Le programme prend en charge trois options d'argument :

- ❖ **Aucun argument** : utilise le fichier de données par défaut "tournoi.txt" et une durée de match de 1000 millisecondes.
- ❖ **Un argument** : si l'argument est un nom de fichier valide, utilise ce fichier comme fichier de données. Sinon, si l'argument est un nombre, utilise ce nombre comme durée de match.
- ❖ **Deux arguments** : utilise le premier argument comme nom de fichier de données et le deuxième argument comme durée de match. Les arguments doivent être respectivement un nom de fichier valide et un nombre.

Initialisation des équipes et des matchs La deuxième partie du code initialise les équipes et les matchs à partir du fichier de données spécifié. La fonction `get_num_equipe` est utilisée pour déterminer le nombre d'équipes à partir du fichier, et la fonction `initEquipesEtMatches` est utilisée pour initialiser les équipes et les matchs à partir du fichier.

Vérification de la puissance de deux du nombre d'équipes Le code vérifie que le nombre d'équipes est une puissance de deux. Cette vérification est importante car le tournoi est conçu pour être organisé avec un nombre de matchs qui est une puissance de deux. Si le nombre d'équipes n'est pas une puissance de deux, le programme s'arrête et affiche un message d'erreur.

**Création des threads** La troisième partie du code crée les threads pour simuler les matchs. Le premier tour est joué séparément, car il ne peut y avoir qu'une seule série de matchs à ce stade. Pour les tours suivants, le programme divise le nombre de matchs par deux et crée des threads pour chaque paire de vainqueurs du round précédente.

**Utilisation de mutex et de sémaphores** Le code utilise un mutex pour synchroniser l'accès aux variables partagées entre les threads et deux sémaphores pour signaler le début et la fin des matchs.

La finalisation du programme se termine avec la destruction des mutex et des sémaphores à la fin du programme.

**Répartition des tâches :**

	SANTOKI Nihar	ZHANG Victor
Programmation	50%	50%
Guide d'utilisateur	50%	50%
Guide de programmation	50%	50%

**Problématique et éventuelle amélioration possible :**

- Implémentation de la durée (problème de segmentation pour la durée)
- Implémentation d'une interface graphique pour le développement / affichage des parties en cours
- Une version "manuelle" du programme où l'utilisateur décide au clavier de chaque point gagnant
- Une version jouable de la simulation avec le PvP (player vs player), le PvB (player vs bots) comme possibilité