# Javass Bonus

**Chat**

Nous utilisons un port différent (5109) pour la communication liée au chat.

Premierement, on a créé un nouveau bean nommé ChatBean qui a trois attributs de type StringProperty : un qui contient tout le texte du chat et deux autres qui fonctionne comme des ArrayBlockingQueue parce qu’ils contiennent les messages qu’on vient de recevoir ou qu’on veut envoyer.

Des deux côtés, serveur et client, on lance un Thread (séparé de la communication pour le jeu) qui écoute continuellement pour des messages et envoi des messages quand besoin.

Finalement, le LocalMain gére la réception des messages des clients et l’envoi des messages des joueurs humains jouant sur le LocalMain.

**Joueur MCTS Parallèle**

Nous joueur MCTS Parallèle utilise aussi l’algorithme MCTS mais au lieu de construire un seul arbre, il en construit plusieurs en parallèle en utilisant des **noyaux différents pour chaque arbre**.

Cela est fait au moyen d’un **ExecutorService** qui permet de créer plusieurs Threads et les exécuter en parallèle. On utilise son méthode **networkStealingPool** parce que nous n’avons pas besoin que les arbres soient construits dans un ordre particulier. A la fin, la carte avec la meilleure ratio points : nombre d’itérations est choisie.

Si le nombre d’arbres à construire n’est pas fourni au constructeur, le nombre de processeurs disponible au JVM est utilisé car ce nombre est aussi le nombre maximal des Threads que le JVM peut vraiment exécuter en parallèle.

**Choix de l’atout**

On a ajouté le méthode chooseTrump(PlayerId chooser, CardSet hand, boolean canPass) dans l’interface Player qui est appelé au début du chaque tour pour tous les joueurs dans JassGame avec le premier joueur du tour comme le « chooser », le main correspondant au joueur et le canPass = true (signifiant que le joueur peut annoncer chibre). Si la valeur retournée par le « chooser » est une couleur (càd il n’est pas null), l’atout est choisi et le tour commence. Sinon, le méthode chooseTrump est appelé de nouveau pour tous les joueurs mais cette fois avec le canPass = false. Cette fois le « chooser » ne peut pas annoncer chibre et donc l’atout est choisi et le tour commence.

**Joueur Humain**: Le text « Choix de l’atout en cours » est affiché si ce n’est pas nous qui choisissons, sinon nous voyons nos choix. On a décidé d’utiliser de nouveau un ArrayBlockingQueue<Color> (de taille 2) pour la communication entre le GraphicalPlayer et le GraphicalPlayerAdapter. Le protocole de communication de cette queue est le suivant :

1. Si un atout est choisi, il est ajouté deux fois à la queue.
2. Si le choix est passé, deux couleurs différents sont ajoutées à la queue.

Pour la communication du « chooser » et « canPass », des attributs observables ont été ajoutés au TrickBean.

**Joueur simulé** : On choisit juste la couleur mode (dont on a le plus grand nombre de cartes) :

1. Si le nombre de carte de cette couleur est inférieur à 4\*, il passe le choix s’il a la possibilité.
2. Sinon, cette couleur est choisie comme l’atout.

\*Par le principe des tiroirs, tous les mains possibles auront au moins 3 cartes de la même couleur (9 cartes et 4 couleurs => ceiling(9/4) = 3). Alors, Si on que 3 cartes de la même couleur, c’est toujours mieux de passer le choix car il ne peut pas faire pire !

**Indice - Meilleure carte à jouer**

Pour ajouter cette fonctionnalité, on a dû ajouter trois choses au projet :

1. Un joueur simulé du type ParallelMCTSPlayer comme attribut à l’interface graphique
2. Un **ArrayBlockingQueue<TurnState>** dans le GraphicalPlayerAdapter qui est passé aussi au GraphicalPlayer correspondant.
3. Des halos autour de tous les cartes de la main, invisibles initialement.

Lorsque le méthode cardToPlay du GraphicalPlayerAdapter est appellé, il ajoute le turnState reçu comme argument au (2). Si le joueur humain appui sur le bouton d’indice, le GraphicalPlayer prend le turnState du (2) et reconstruit le Hand avec son HandBean. Puis il utilise (1) avec ces éléments pour déterminer le meilleur carte à jouer et rend le halo correspondant à ce carte visible. Quand ce n’est plus à ce joueur à jouer, tous les halos sont rendus invisibles.

Le GraphicalPlayerAdapter vide toujours le (2) pour gérer le cas où le joueur n’utilise pas un indice et donc le GraphicalPlayer ne prend pas le turnState.

**Chronomètre**

Un compte à rebours commence quand c’est à nous de jouer. Une carte aléatoire est jouée si le temps s’écoule.

**Blague**

Quand un joueur écrit un message contenant « /blague » dans le chat, une nouvelle blague (prise d’un page web) est envoyée dans le chat.

**Plis valant 0 points**

Nous montrons un « (+0) » pour les plis valant 0 points.

**Sons**

Plusieurs sons sont joués : utilisation d’un indice, temps écoulé, message **reçu**.