UFR ST Besançon 2017-2018
Master 1 Informatique

Projet MOIA/SCS - Partie SCS
Jeu Kamisado en réseau

1 Description du projet

Le but de ce projet est l'implantation de la communication permettant de jouer en réseau au jeu Kamisado. Les règles du jeu sont données dans l'énoncé du projet MOIA. La suite définit les spécifications du travail à réaliser pour la partie communication. Le travail est à remettre en binôme - un membre du binôme développera le serveur et l'autre le joueur.

Chaque joueur est à écrire **en langage Java**. Il fait appel à un moteur IA (développé également en **Java**). **Le moteur IA communique avec l'intelligence artificielle du jeu développée en Prolog.** Les parties se jouent à deux joueurs et sont administrées par un processus serveur, arbitre du jeu, écrit **en langage C**. Les processus joueurs communiquent, à travers des sockets, avec le serveur.

1.1 Le joueur

Voici le comportement d'un processus joueur :

1. Il envoie au serveur une requête **PARTIE** pour lui demander de jouer, en précisant son nom et la couleur souhaitée pour son pion. En réponse, le serveur lui donne le nom de son adversaire et lui indique si la couleur transmise lui est validée. En cas de couleur invalidée, le joueur doit jouer avec l'autre couleur disponible. Le joueur souhaitant jouer avec les pions blancs dont la couleur est validée commence la partie.
2. Lorsque c'est à son tour de jouer, le joueur consulte son moteur IA pour connaître le coup à jouer. Il constitue une requête **COUP** qu'il envoie au serveur, puis il en attend la validation.
3. Il attend ensuite des informations sur le coup de l'adversaire, transmises par le serveur. Ce temps peut être utilisé par le joueur pour pré-calculer les prochains coups. Il reçoit dans un premier temps la validation du coup de l'adversaire. Si ce coup est valide et la partie devrait continuer, il reçoit le coup de l'adversaire.
4. Si la partie n'est pas finie, le joueur informe son moteur IA du coup joué par l'adversaire et retourne en 2.
5. Si la partie est finie, il doit afficher le résultat de la partie et relancer une nouvelle partie (à partir de l'étape 2), en laissant le joueur avec les pions noirs commencer en premier.
   Remarque. Les couleurs affectées aux joueurs ne seront pas modifiées.

1.2 Le serveur

Le processus serveur réalise les actions suivantes :

1. Il attend d'avoir deux requêtes **PARTIE** de la part de joueurs. Il répond à chacun, en validant la couleur du joueur dont la requête est reçue en premier et en invalidant la couleur de l'autre joueur si elle est identique.
2. Le serveur doit assurer le déroulement de deux parties - une première où le joueur avec les pions blancs commence et une deuxième où l'autre joueur commence. **Remarque.** Les couleurs des deux joueurs restent inchangées entre les deux parties.
3. Il attend, lors de chaque partie, alternativement des requêtes **COUP** des deux joueurs. Cette attente doit être bornée dans le temps par une constante **TIME_MAX** fixée à 6 secondes. Pour chacune des requêtes reçues il vérifie si le coup joué est valide (voir 1.4 - une bibliothèque vous sera fournie) et donne la réponse au joueur qui a envoyé la requête. Puis il envoie cette même réponse au joueur adverse. Si le coup est valide **et si la partie devrait continuer**, il lui envoie ensuite le coup joué.
4. Une partie est finie si un des joueurs gagne, perd ou s'il y a partie nulle (coup validé par le serveur).

1.3 Protocole d'accès au serveur

Toutes les communications se feront en mode connecté. Vous trouverez sous moodle dans le module SCS - rubrique Projet, le fichier `protocolKamisado.h` qui définit le protocole d'accès au serveur de jeu Kamisado.

Voici les règles retenues pour l'utilisation de ces types :

PARTIE : cette requête permet à un joueur de demander une partie. Le nom à donner est le login LDAP du joueur. Il obtient en retour le nom de son adversaire. Si la couleur envoyée et validée de la part du serveur est `BLANC`, il commence à jouer. Sinon il attend de recevoir le premier coup de son adversaire.

COUP : une requête `TCoupReq` est envoyée par le joueur au serveur pour jouer un coup. Cette requête donne le numéro de la partie en cours (la première étant numérotée par 1), la couleur du pion du joueur ainsi que la caractéristique du pion (`coul`), le type de coup `typeCoup` et, en fonction de ce dernier, des informations sur le coup joué (`deplPion`). La valeur associée à `typeCoup` donne donc le type de coup qui est joué, c'est-à-dire s'il s'agit d'un positionnement de pion (`POS_PION`) ou si le joueur passe son tour (`PASSE`). Lorsque le joueur déplace un pion, il doit initialiser, dans le champ `deplPion`, la position de la case de départ (`caseDep`) et de la case d'arrivée (`caseArr`), du pion déplacé.

En réponse à la requête, le serveur retourne une validation du coup `TCoupRep` avec une valeur d'erreur initialisée à `ERR_OK` si le coup est joué après l'initialisation de la partie, et le champ `validCoup` précise si le coup est valide, en timeout ou s'il y a triche. En plus, l'information si la partie doit continuer ou si elle est finie (le coup courant étant gagnan, perdant ou engendre une partie nulle) est donnée dans le champs `propCoup`.

Si son coup est valide, le joueur attend ensuite de recevoir un prochain message de la part du serveur. Il reçoit d'abord un message `TCoupRep` qui est la validation du coup de l'adversaire. Si le coup de l'adversaire est valide **et que la partie continue**, il reçoit ensuite le message `TCoupReq` qui a été envoyé au serveur par l'adversaire. Dès qu'il a reçu le deuxième message, il peut s'adresser à son moteur IA pour lui demander le prochain coup à jouer ou pour l'informer de la fin de la partie.

1.4 Validation des coups

Quelques précautions sont prises pour éviter les erreurs répétées, voire moins bien intentionnées.

- Si un coup est erroné (par exemple, le coup est un `PASSE` quand le joueur peut encore jouer un de ses pions ou un mauvais pion est joué) alors la partie est perdue. Le serveur retourne une valeur `TRICHE` dans le champ `validCoup` et en informe simultanément le joueur adverse. Le coup invalidé n'est pas transmis à l'adversaire et la partie s'arrête.
- La recherche dans le moteur IA est limitée dans le temps. Le temps d'attente est calculé à partir de l'instant où le serveur envoie le coup adverse. Si la limite est dépassée, le coup est considéré comme nul et le serveur envoie au joueur et à l'adversaire un message `TCoupRep` avec une valeur `TIMEOUT` dans le champ `validCoup` (l'adversaire ne reçoit pas le coup joué). Dans ce cas, la partie est perdue par le joueur. Attention, un joueur peut donc recevoir un `TIMEOUT` alors qu'il est en train de jouer.

Dans ces deux situations, le coup est considéré invalide par le serveur.

- La fonction (dont le prototype est donné dans le fichier `validation.h`) :
  `bool validationCoup(int joueur, TCoupReq coup, TPropCoup* propCoup);`
  permet d'obtenir la validation d'un coup `coup` joué par un joueur (1 pour le joueur **jouant en premier** et 2 pour le joueur **jouant en deuxième**). Le paramètre de sortie `propCoup` renseigne la propriété du coup.
  Remarque. Le code objet qui implémente la validation d'un coup vous sera fourni. Pour l'utiliser correctement d'une partie à l'autre, il faut appeler, en début de chaque partie la fonction `initialiserPartie` (dans le meme fichier d'entête fourni, `validation.h`).

2 Remise du travail

Pour la remise de ce projet nous attendons le code source des programmes serveur et client et un rapport donnant des explications sur votre travail.

2.1 Développements

Vous devez développer le joueur (ainsi que son moteur IA) et le serveur sur la base du protocole de communication décrit ci-dessus. Les messages seront échangés sur des sockets.

Vous nous fournirez les versions de votre joueur (incluant **le code MOIA**) et de votre serveur en le déposant comme devoir dans le module SCS sur moodle, pour le **2 mai à 8h30**, au plus tard. Le fichier, portant les deux noms du binôme, sera une archive .tar.gz qui contiendra un répertoire, avec le même nom. Dans ce répertoire, mettre tous les fichiers de votre projet (sauf les fichiers temporaires .o, ~, etc.) - moteur IA inclus - et un fichier de script appelé `joueur`.

Remarque sur la programmation : pour que vos programmes soient facilement lisibles, vous respecterez le standard de programmation C/Java.

2.2 Rapport

Pour expliquer votre réalisation, vous nous remettrez un rapport (5-6 pages) qui contiendra obligatoirement les informations suivantes :

— l'utilisation du protocole de communication avec le serveur,
— la mise en place d'un protocole avec le moteur IA,
— les explications sur le code, la structure du joueur et du serveur.

La longueur de ce rapport n'est pas un facteur déterminant, portez plus d'attention à sa clarté et à la structuration de vos explications. Ce rapport est à remettre en même temps avec les sources du projet, au format pdf.

3 Championnat

Les joueurs s'affronteront en championnat (le vendredi 4 mai), en utilisant une version exécutable du serveur que nous vous mettrons à disposition.

Pour faciliter le déroulement de ce championnat vous devez respecter le mode de lancement suivant : un script joueur prend en paramètre le nom de la machine et le numéro de port du processus serveur pour s'y connecter, ainsi que le nom le désignant dans le jeu. La forme exigée pour la commande de lancement du joueur est la suivante :

```
sh joueur.sh hostServeur portServeur nomJoueur [opt]
```

où
`hostServeur` et `portServeur` - désignent le nom de la machine et le port du serveur,
`nomJoueur` - désigne le nom du joueur,
`[opt]` - paramètres supplémentaires qui peuvent être définis.

Il faut impérativement fournir un fichier README contenant la ligne de commande avec la sémantique des paramètres.

Attention. Tout manquement à ces règles peut vous exclure du championnat mais, si vous vous y prenez un peu à l'avance, il sera possible de nous demander une validation (lors de la dernière séance de TP).