Beuque Eric
Etienne Christophe

Master 1 info

Rapport projet de SCS-MOIA
SIAM

Partie SCS

Dans le cadre du module de MOIA du Master SST 1ère année mention Informatique, un jeu de logique est à réaliser. Cette année il s'agit du jeu SIAM.

L'objectif est d'être le premier joueur à sortir un rocher du plateau en le poussant à l'aide de vos animaux : Eléphant ou Rhinocéros.

À votre tour, vous pouvez faire entrer un de vos animaux sur le pourtour du plateau ou en déplacer un s'y trouvant déjà. Vous pouvez pousser des animaux ou des rochers du moment que vous êtes plus nombreux à pousser qu'en face. Lorsque vous poussez, vous utilisez aussi la force des animaux adverses. En cas d'égalité de force, vous êtes bloqués.

Le premier joueur à réussir à sortir un rocher est le gagnant.

Initialisation et création d'une partie :

Tout d'abord, on boucle tant que le tournoi n'est pas fini.

Pour débuter une partie, on envoie une requête d'initialisation a l'aide de la fonction :

```
joueur_siam_init (JoueurSiam *self, char *name, char *machine, short port);
```

Dans un premier temps on initialise à la valeur -1, la socket du joueur, de l'arbitre et de l'adversaire
On copie les valeurs passées en paramètre (le nom du joueur, le nom de la machine et le port), dans l'objet "joueur".
On crée une socket sur le port passé en paramètre. On renvoie une erreur si la socket n'a pas été correctement créée.

Puis, lorsque le joueur est prêt, on envoie une requête partie à l'arbitre pour lui demander de jouer une nouvelle partie.

```
res = joueur_siam_demande_partie (&joueur);
```

On crée deux objets, l'un est du type "TypCoupReq" et l'autre "TypCoupRep". On envoie une requête de demande de partie à l'arbitre avec l'identifiant du joueur.
On reçoit et on vérifie alors la réponse de l'arbitre. L'arbitre nous renvoie le nom de la machine de l'adversaire, son port, son identifiant et le type de l'adversaire (Eléphant ou Rhinocéros).
On affiche alors notre type d'animal.
On appelle alors la fonction "nouvellePartie" et on passe en paramètre les deux joueurs avec en premier le joueur qui débute.

On attends la connexion de l'adversaire avec la méthode
joueur_siam_connexion_adversaire (&joueur);.

Enfin on boucle indéfiniment jusqu'à ce que la partie soit terminée.

```
if (joueur.type_animal == ELEPH) {
    res = joueur_siam_joue_coup(&joueur);
    if (res != 0){
        joueur.fin_partie = VRAI;
    }
}else{
    res = joueur_siam_attend_coup_adverse(&joueur);
    if (res != 0){
        joueur.fin_partie = VRAI;
    }
}
```

Si nous sommes les éléphants, on joue un coup, sinon on attends le coup de l'adversaire.

Dans la fonction joueur_siam_joue_coup, on crée deux objets. L'un est du type "TypCoupReq" et l'autre "TypCoupRep".
Puis dans un second temps, on teste les connexions avec l'arbitre et l'adversaire.
On demande ensuite le nouveau coup a jouer. Ce coup sera donné à l'aide de la fonction "get_next_coup_req(&coup_req)".

Ensuite on regarde si l'arbitre a envoyé un TIMEOUT

```
ioctl (self->arbitre.socket, FIONBIO, &on);
res = recv (self->arbitre.socket, &coup_rep, sizeof(TypCoupRep) + 50, 0);
```

Si nous n'avons pas reçu de "TIMEOUT" de l'arbitre, on lui envoie la requête.

L'arbitre renvoie alors une réponse afin de savoir si notre coup est valide. Si ce coup est valide, on l'envoie à l'adversaire.
L'adversaire renvoie a son tour une confirmation.

Si ce coup est gagnant ou perdant, on arrête la partie.

Dans le cas où c'est à l'adversaire de jouer, on appelle la fonction *joueur_siam_attend_coup_adverse(&joueur).*

Dans cette fonction, on crée deux objets. L'un est du type "TypCoupReq" et l'autre "TypCoupRep".

On crée deux sockets, l'une sur l'arbitre et l'autre sur l'adversaire. Puis nous utilisons la fonction "select" afin de choisir sur quelle socket arrive l'information en premier.

Dans une, on reçoit le coup de l'adversaire envoyé par l'arbitre. Dans l'autre le coup de l'adversaire.
On envoie alors une confirmation à l'adversaire. Puis la validation de l'adversaire à l'arbitre. On reçoit la réponse de l'arbitre. A ce moment là, on ajoute le coup de l'adversaire dans le jeu après l'avoir validé.

Si le coup est gagnant ou perdant, on arrête la partie.

Pour terminer, lorsque la partie est finie, on déconnecte les joueurs.

```
res = joueur_siam_deconnexion_adversaire (&joueur);

if (res<0){
   joueur_siam_destroy (&joueur);
   return -1;
}
```

Partie communication C <-> Java <-> SicstusProlog :

Dans notre projet, nous sommes amenés à programmer une intelligence artificielle en langage Prolog. Ce langage possède de grandes possibilités en terme d'intelligence artificielle. Or, nous devions communiquer avec l'arbitre qui est écrit en langage C. Pour récupérer les coups générés par notre IA, nous avons utilisé le langage Java couplé d'une librairie Sicstus appelée "Jasper". Nous allons vous expliquez les méthodes utilisées afin de réaliser cette transition. Les explications concernant la programmation Sicstus Prolog feront l'objet d'un autre rapport.

Le joueur écrit en langage C doit, pour participer à la partie, générer des coups. Afin de générer un coup, on envoie une requête au fichier "Moteur.java" à l'aide de connexion par "socket" entre le langage C et Java.

Tout d'abord on crée une soket du joueur de la façon suivante :

sock = socketClient ("localhost", port);

Puis on envoie une requête (qui est une chaine) au Moteur.java grâce à la fonction *EnvoiToJava(int sock, char * chaine).* Cette fonction utilise la méthode :

err = send (sock, chaine, strlen(chaine), 0);

Une fois la requête recue dans le moteur java, nous allons pouvoir l'envoyer à notre IA. Cette action est réalisée par une méthode écrite en java qui se nomme :

public static String solutionUnResultatMax(SICStus sp, String saisie, String J, String TC, String L1, String C1, String L2, String C2, String O)

Cette fonction prend en paramètre un objet SICStus, la requête en Prolog ainsi que les noms de toutes les variables que l'on veut récupérer.
Joueur, Type de coup (Entree, sortie..), Position de départ, Position d'arrivée, Orientation.

```
resultat += results.get(J).toString();
resultat += ";"+results.get(TC).toString();
resultat += ";"+results.get(L1).toString();
resultat += ";"+results.get(C1).toString();
resultat += ";"+results.get(L2).toString();
resultat += ";"+results.get(C2).toString();
resultat += ";"+results.get(O).toString();
```

Cette méthode nous retourne le résultat sous la forme d'une chaine du type : "r;sortie;5;5;null;null;null". Cette solution est alors renvoyée au fichier "joueur.c" en utilisant la même socket qui est toujours ouverte.
Une fois la solution récupérée, nous allons en extraire les données qu'elle contient. Pour ce

faire nous avons réalisé une fonction qui extrait les variables en fonction du délimiteur ";".

Ces variables sont alors utilisées dans la fonction *get_next_coup_req (TypCoupReq *coup_req).*

La fonction get_next_coup_req (TypCoupReq *coup_req) :

Dans un premier temps, on se connecte au moteur de l'IA à l'aide d'une socket.
On construit une chaine de caractère contenant le tableau représentant le plateau.
On envoie ensuite cette chaine au moteur de l'IA. En retour, le moteur java retourne un coup a jouer.

Dans ce coup on récupère, les paramètres comme le type de coup, les positions de départ et d'arrivé et l'orientation. Enfin on les ajoute dans la structure coup.