

CROQUEMOUTON

Principe

CROQUEMOUTON est un jeu dont vous êtes le héros : vous êtes le berger devant sauver son troupeau de moutons des griffes d'une horde de loups. Il vous faut donc, armé de votre souris et de votre clavier, commander vos chiens de berger pour guider votre troupeau le plus vite possible à la bergerie en évitant les loups errants.

Ce jeu peut se jouer seul ou bien en réseau, où vous avez alors le privilège de contrôler en sus quelques loups pour aller asticoter le troupeau du voisin.

Le gagnant est alors celui qui a réussi à rentrer le plus grand nombre de moutons à sa bergerie.

Comment lancer le jeu

Le jeu est programmé en Java, et peut donc fonctionner sur la plupart des plateformes courantes.

Après avoir téléchargé le jeu, vous pouvez alors le décompresser en tapant par exemple (sous Linux) :

```
tar -zxvf Croquemouton.tgz
```

Pour le lancer ensuite :

- sous Linux, il vous faut ouvrir une console et taper :

```
./lanceur
```

- sous Windows, il vous faut ouvrir une invite de commandes et taper

```
lanceur.exe
```

La première fenêtre vous permet de définir les divers paramètres de jeu.

- Si vous voulez créer un serveur :

Cochez la case « Créer le serveur ? ». Après avoir créé un serveur et décidé du nombre d'animaux que vous voulez voir sur la carte, cette dernière s'affiche ainsi qu'une fenêtre montrant le nombre de joueurs ayant rejoint votre partie : une fois que vous estimez avoir atteint un seuil raisonnable (de 1 joueur à 22 joueurs), vous pouvez alors lancer le jeu.

- Si vous voulez rejoindre une partie sur un serveur pré-existant :

Dans le cas où vous-même êtes un client qui veut rejoindre une partie ayant lieu sur un serveur, il vous suffit d'entrer le nom de l'ordinateur sur lequel tourne le serveur pour pouvoir commencer la partie dans la case « Nom du serveur ».

N'oubliez pas de sélectionner votre ambiance préférée, Aquatique ou Terrestre !

Comment jouer

Le jeu se joue au clavier et à la souris.

Si jamais, perdu dans le feu de l'action, vous avez oublié quelques commandes, il vous suffit de vous rappeler d'une lettre : « h », qui fera apparaître une magnifique et fort complète fenêtre d'aide.

Seuls les chiens (et les loups dans le mode multijoueur) peuvent se contrôler. La sélection des chiens ou des loups se fait au clavier en entrant leur numéro : ce dernier est alors affiché en rouge au-dessus de la carte de jeu, l'écran se recentre automatiquement sur l'animal sélectionné, qui se retrouve de plus entouré d'un cercle jaune. Dans un souci d'ergonomie, si la sélection par numéro ne vous convient pas, vous pouvez utiliser les touches « d » pour sélectionner le canidé suivant et « x » pour sélectionner le canidé précédent. Une fois le carnivore sélectionné, vous pouvez lui indiquer où aller en cliquant gauche à un endroit de la carte, et même lui définir tout un itinéraire en définissant des « points de route » : en

maintenant la touche « shift » enfoncée, et en cliquant gauche sur la carte, vous aurez le plaisir de voir votre canidé préféré visiter dans l'ordre tous les points de la carte ainsi définis. Vous pouvez de plus observer la carte attentivement puisqu'est implémentée une fonction de zoom : en appuyant sur « z », la carte grossit, et en appuyant sur « e », la carte s'éloigne. Enfin, vous pouvez déplacer la fenêtre de vision sur la carte en cliquant droit sur la carte, en se servant du pavé directionnel de votre ordinateur ou bien encore en cliquant sur la micro-fenêtre de situation en haut à gauche de la fenêtre principale ! Vous aurez gagné lorsque vous aurez mis dans votre bergerie (de la même couleur que vos chiens) plus de moutons que vos adversaires.

Détail de l'implémentation

L'ensemble de l'application est basée sur une architecture « Client-Serveur », du fait de son utilisation en réseau. Les clients sont de simples moteurs d'affichage, allant chercher les images dans le dossier « sprites » selon les instructions que lui transmet le serveur. Tous les calculs du jeu, génération de la carte, déplacements, interactions entre animaux se font sur le serveur. Seules les informations utiles sont ensuite envoyées aux clients sous la forme de « Data ».

Le serveur lui-même peut être décrit à l'aide du paradigme « Client-Serveur » : chaque individu en effet peut alors être vu comme un client qui demande périodiquement des informations au serveur, telles que sa position et ce qui l'entoure (fonction « ouSuisJe »).

Déroulement du jeu : de l'intérieur

La classe **CroqueMouton** est la première lancée : c'est elle qui lance alors la fenêtre qui permet l'entrée des options, de la classe **FenetreEntree**. Imaginons que l'on crée un serveur, car la procédure qui consiste à ne créer qu'un client n'en est qu'une simplification.

Ce sont alors deux éléments qui sont créés : un **Maitre** et un **Client**.

Le **Maitre** est une classe utilitaire dont le rôle est de lancer le serveur (classe **ServeurImpl**, qui implémente l'interface **Serveur**) et de régler les problèmes relatifs au réseau (régler la sécurité, créer le registre...). Le **Client** lui s'initialise (fonction *initClient*, appelée par la fonction *main*) : il va chercher le serveur à l'adresse qui lui a été fournie (fonction *trouveServeur*), se déclare à lui (fonction *declareClient*) et récupère les données lui permettant d'afficher le terrain et les animaux (fonction *recupereInit*, qui initialise les vecteurs « listeTerrain » et « listeIndividus » à partir des vecteurs « terrain » et « occupants » du serveur). Notons que les exceptions dues à une mauvaise adresse sont ici traitées : des fenêtres s'affichent permettant à l'utilisateur de prendre conscience de son erreur (fonction *trouveServeur*).

Du côté du Client...

Après que le client s'est initialisé, il crée alors des représentants des classes **GestionnaireFenetre**, **GestionnaireImages**, **GestionnaireEntreesSorties**, **GestionnaireTerrain**.

- **GestionnaireFenetre** gère la fenêtre principale du jeu contenant la carte et les scores : elle hérite de la classe Java **JFrame**. Du fait qu'elle implémente l'interface **Runnable**, elle constitue un thread à part entière, ce qui lui permet de relancer la mise à jour des données du client et partant des Gestionnaires divers qu'elle a créés (fonction *paint*, qui appelle la fonction *metaJour* du **Client**).
- **GestionnaireImages** recueille auprès du **Client** qui l'a créé le vecteur d'individus (occupants, au travers de la fonction *metaJour*), qu'il peut alors convertir (fonction

convertiDonnees) pour afficher les « gifs » correspondant aux individus du vecteur. Sa fonction *metaAJour* est appelée par la fonction *metaAJour* du **Client**, et appelle à son tour la fonction *metaAJour* du **GestionnaireFenetre**.

- **GestionnaireEntreesSorties** récupère les clique souris et les commandes entrées au clavier depuis la fenêtre de jeu (**GestionnaireFenetre**) et les traite : le Client lui transmet les clique enregistrés par **GestionnaireFenetre** : il sait alors qu'en faire, et peut transmettre les informations pertinentes (endroit où aller pour un canidé par exemple) au serveur au travers des fonctions *recupereMouvement* et *recuperePointDeRoute*.
- **GestionnaireTerrain** s'occupe d'afficher le terrain, après conversion des données fournies par le **Client** (simplement le vecteur terrain).

La fonction *metaAJour* du **Client** mérite que l'on s'y attarde quelque peu. Elle permet en effet deux comportements différents : soit elle réinitialise totalement le vecteur d'individus « occupants » (le vecteur « terrain » lui est stable), soit elle se contente de faire une mise à jour plus légère de ses données (abscisse et ordonnée des individus). Cet aiguillage de comportement est subordonné à l'état d'un booléen qui se trouve sur le serveur : ce dernier possède en effet pour chaque **Client** une instance de la classe **IdClient** qui possède un booléen « *mettreAJour* » : celui-ci est mis à vrai lorsqu'un changement important (changement dans le nombre d'individus ou changement dans les noms des animaux – mort d'un animal) intervient dans le vecteur d'individus (« occupants ») du serveur. Ce dernier met alors les booléens « *mettreAJour* » des **IdClients** à true au travers de la fonction *declencheMiseAJour* (située dans **ServeurImpl**). Lorsque le **Client** a récupéré le vecteur, il déclenche la fonction *majEffectuee* avec son numéro d'identification comme argument : le booléen de l'instance **IdClient** lui correspondant est alors mis à false.

Du côté du Serveur...

Lorsqu'on crée un serveur, une fenêtre de classe **FenetreJoueur** est créée par le thread **HorlogeJoueur**, et vérifie périodiquement le nombre et l'identité des **Clients** connectés. Lorsque l'utilisateur qui a créé le serveur appuie sur « lancer le jeu », la méthode « *run* » du serveur est appelée, le jeu peut commencer. Ceci n'empêche pas un nouveau client de se joindre à la partie, il est simplement désavantagé par rapport à ses camarades...

Le serveur, à l'initialisation, crée une **Carte** (qui s'appelle *ecosysteme*) et qui a pour rôle d'initialiser le « terrain » et les « occupants » (fonction *init* de **ServeurImpl**). Par la suite, à chaque arrivée d'un nouveau **Client**, le vecteur « occupants » est mis à jour. Du simple fait qu'alors la taille du vecteur « occupants » a changé, une mise à jour drastique sera déclenchée pour chaque client qui était présent auparavant (fonction *metaAJour* du **Client**).

Afin de recommencer périodiquement la procédure de mouvement et d'interaction pour les animaux, la classe **Tour** a été définie. Celle-ci est là-encore un thread, créé dans la méthode *run* de **ServeurImpl**, qui lance pour tous les individus du vecteur « occupants » la méthode *action*.

Comportement des individus

A chaque tour, la méthode *action* (située dans **Vivant**) de chaque animal et de la bergerie est appelée : celle-ci permet à chaque animal de demander ce qu'il y a autour de lui (méthode *ouSuisJe* de **ServeurImpl**), ce qui lui permet de se déplacer (*seDeplacer*) ou de réagir à son environnement (*reaction*). En ce qui concerne la bergerie, cela lui permet de sauver des moutons par exemple.

Le comportement des individus est codé dans la classe **Comportement** : à chaque création d'un individu, celui-ci est entré dans la table de hachage contenue dans

Comportement, en vis-à-vis de sa classe, codée sous forme d'une chaîne de caractères. Ensuite, à chaque tour, la fonction *creeCommentReagir* de **Comportement** crée un objet **ComportementIndividu**. Chaque individu pourra alors y rechercher le comportement à adopter au travers de sa méthode *reaction* (de l'interface **Vivant**).