Chapitre 1

Client Java - Vincent Botbol

Ce client a été réalisé en *Java* (version 7). L'interface graphique utilise la bibliothèque *Swing*. Les extensions implémentés sont : la discussion instantanée, les comptes utilisateurs et le mode spectateur. Les tests ont été réalisés sur le serveur OCaml fourni ainsi que sur un serveur factice java également présent dans l'archive.

Le programme démarre sur une fenêtre de connexion proposant à l'utilisateur d'entrer l'adresse du serveur, son pseudo, son mot de passe (pour les comptes utilisateurs), de choisir le mode spectateur, et de se connecter (protocole "LOGIN" si le mot de passe est fourni, "CONNECT" sinon) ou d'enregister son compte.

Après la réponse du serveur, l'interface graphique du jeu s'affiche. Celle-ci se décompose en trois parties :

- La grille de jeu
- Une zone de texte pour les informations émises par le programme et par le serveur
- La zone de discussion

Pour communiquer avec les autres joueurs (et spectateurs), il suffit d'entrer son texte puis de valider avec "entrée" ou bien de cliquer sur "Envoyer".

La grille de jeu permet les interactions de l'utilisateur et la zone de notification transmet les informations importantes.

Dès la réception du "SHIP", l'utilisateur place son bateau en déplacant la souris sur la grille de jeu (clique droit pour le tourner), et valide sa position avec clique gauche. Le client réalise une vérification interne sur le placement avant d'envoyer la commande "PUTSHIP" au serveur. Les bateaux sont représentés par des rectangles de couleur rouge, grise, verte ou jaune selon leur positions dans la liste de joueurs.

La disposition des bateaux terminée, le client attend le "YOURTURN" avant de redonner la main à l'utilisateur. Lorsque son tour est arrivé, le drone apparait ainsi qu'une zone d'action basée sur les mouvements accordés. L'utilisateur finit son tour lorsqu'il ne lui reste plus de mouvement ou bien qu'il passe son tour (clique droit). En cliquant sur sa position, le joueur active le laser. Si un bateau a été touché pendant ce tour, l'utilisateur peut le constater grâce à une croix verte apparaissant à la position du bateau touché. Si le coup est manqué, un rond magenta est disposé. Si l'un de ses bateaux est touché pendant le tour d'un des adversaires, le client est notifié d'une croix

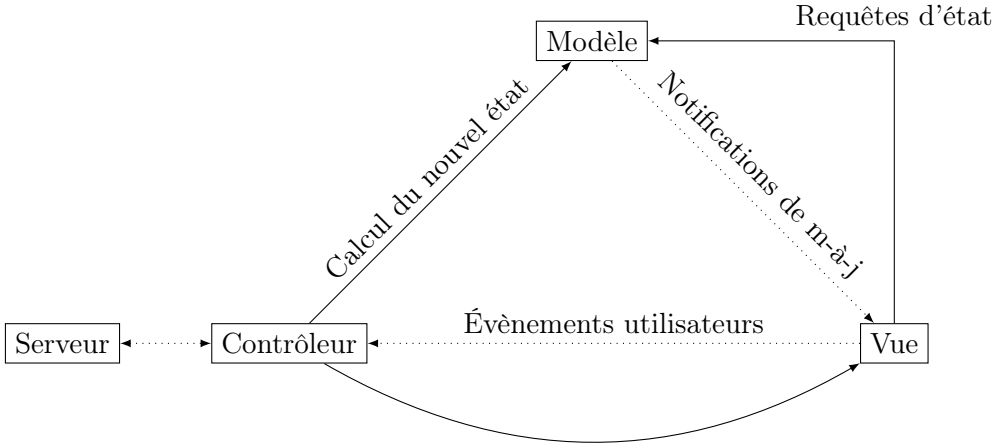
rouge à l'emplacement du bateau détruit.

En mode spectateur, l'utilisateur ne peut intéragir. Il peut uniquement communiquer par le biais de la discussion instantanée. Il est notifié des touches de bateaux par un code couleur propre au joueur ayant effectué l'action.

1.1 Implémentation

1.2 Modèlisation

L'architecture du client suit un patron de conception MVC strict. Ce schéma illustre la modélisation de l'application :



On distingue donc trois parties principales :

- Le modèle, gérant toute la partie logique de l'application : la grille, le calcul des déplacement, la validité des coups pour la vérification côté client, etc..
- La vue, s'occupant d'afficher les composants graphiques pour communiquer avec l'utilisateur. Cette dernière va s'enregistrer auprès du modèle qui va notifier la vue lorsqu'elle est censée se mettre à jour.
- Le contrôleur, connaissant le modèle et la vue, va permettre de connecter l'interface graphique (boutons, champs textes, ..) et ainsi de traiter les divers évènements que l'utilisateur est susceptible d'envoyer par le biais de l'interface. En réceptionnant ces évènements, le contrôleur établit les calculs à effectuer par le modèle. Le contrôleur s'occupe également de la communication avec le serveur, en traduisant en requêtes les actions de l'utilisateur transmises par la vue et en réceptionnant les réponses du serveur qui démarreront les calculs.

Pour expliciter cet modélisation, on peut détailler un exemple, celui de l'envoi d'un message instantané :

1. Le client saisi son message dans le champ de texte puis clique sur "Envoyer"
2. Le bouton "Envoyer" de la vue étant relié au contrôleur, celui-ci procède à la récupération du texte au travers de la vue. Puis envoie une requête de type "TALK/message/" au serveur

3. Le serveur répond en transmettant la commande “HEYLISTEN”. Le contrôleur traite cette nouvelle commande reçue en appelant le module discussion du modèle qui va ajouter ce nouveau message, prévenir la vue qu'il y a un changement de type “discussion instantanée” et qu'elle doit donc se mettre à jour.
4. La vue étant enregistrée sur le modèle, elle reçoit cet événement, récupère le nouveau message et enfin l'ajoute à sa zone de discussion.

Cette séquence de communication se repète pour la plupart des interactions du programme.

1.3 Concurrence

Le modèle de concurrence employé est préemptif puisqu'il utilise exclusivement l'API native de Java. L'interface graphique est exécutée à l'intérieur de l'“Event Dispatcher Thread” (EDT) pour garantir la fluidité de l'interface graphique même lors de calcul long réalisés par le modèle. Ces calculs sont eux-mêmes exécutés dans des threads [1] dédiés et extérieurs à l'EDT. Des “SwingWorker” ont pour cela été utilisés.

Le principal thread (explicite) de l'application reste la lecture continue des réponses du serveur incluant donc les différents calculs impliqués. D'autres threads secondaires sont également déployés. Notamment pour la mise-à-jour de l'interface graphique ou encore pour effectuer des effets visuels sur la grille de jeu (qui ne sont, certes, pas nombreux).

1.4 Communications Client-Serveur

Comme spécifié dans le sujet du projet, la connexion s'appuie sur un protocole TCP sur le port 2012. Les sockets utilisées sont celles de Java provenant du paquet “java.net” et les lectures/écritures sont réalisés par canaux tamponés. Les envois et récéptions sont textuelles. Les données sont transmises au moyen d'un “DataOutputStream” pour augmenter la portabilité des données envoyées. Les lectures sont quant à elles interprétées directement par un “BufferedReader” lisant les commandes reçues au moyen d'un “readLine()” (puisque les commandes sont préfixés d'un retour à ligne).

Le protocole de communication fourni a été respecté. Le traitement des commandes reçues s'effectue par expression régulière. L'utilisation du *look-behind* a permis de s'abstraire du cas particulier (tels que le “”). Il a donc suffit de séparer la chaine de caractère reçue par l'expression “(?<!\\)/” qui peut-être traduite par : “sépare les champs de la chaine si un slash est rencontré uniquement si il n'est pas précédé d'un anti-slash”.

Les slash saisies par l'utilisateur dans un message instantané, par exemple, sont traités avant l'envoi (en les sécurisant par un anti-slash) et de même pour les anti-slash. La réciproque est employée lors de la réception.

Pour plus de sécurité, les commandes possible à l'envoi sont typés par énumération (ERequest). Les commandes reçues sont traités et coércées vers une deuxième énuméra-

1. Fil d'exécution

tion (EResponse) qui générera une exception si la commande reçue ne fait pas partie de l'énumération. Le tout est encapsulé dans un objet "Command" contenant la commande protocolaire et les arguments reçus ou à envoyer.

1.5 Difficultés

La principale difficulté de ce projet a été d'établir la structure principale de l'application. Une fois cette phase réalisée, l'enchainement de l'implémentation des différentes parties du jeu fut triviale.

Le temps accordé a également été une problématique. J'aurais souhaité améliorer l'aspect visuel et sonore de l'application, ajouter quelques extensions et peaufiner l'implémentation mais la date limite approchant, je n'ai pas eu cette occasion.