Rapport de projet Bataille Navale

Auteurs :
Charles Éponville,
Nicolas Gille,
Vincent Metton,
Grégoire Pommier,
Sylvestre Vimard

Responsable :
M. Yannick Guesnet

28 avril 2016

Table des matières

Table des figures

Remerciements

Nous remercions :
M.GUESNET pour avoir proposé le sujet, nous avoir soutenu et encouragé tout au long du projet,
M.ANDARY pour avoir répondu à toutes nos question à propos de la manipulation des Threads en java,
M.BEDON pour avoir répondu à toutes nos questions quand au développement Java, serveur côté Java et l'utilisation des Threads,
M.VASSEUR pour avoir répondu à toutes nos réponses liées à la partie réseau du projet,
nous remercions aussi tout ce qui nous ont aidés, apportés leur point de vue sur la tournure du projet, ou encore sur de potentielles idées d'améliorations.

Introduction

De nos jours, il est possible de jouer à des jeux de société même avec des personnes qui ne sont pas dans la même pièce grâce aux différents protocoles réseaux. De plus en plus de jeux sont donc implémentés et utilisent ces technologies pour permettre de jouer en réseau, c'est à dire, de pouvoir jouer entre deux machines distantes.

De cette constatation, le besoin est venu d'implémenter un classique des jeux de société : la bataille navale (ou "touché coulé") qui est un jeu se jouant à deux joueurs dont le but est de couler tous les bateaux de l'adversaire, disposés sur un plateau de jeu. La bataille navale pouvant se jouer avec plusieurs règles, nous avons opté pour "le tour par tour" : chaque joueur joue une et une seule fois puis donne la main à son adversaire.

Ce projet a pour objectif de disposer de deux modes de jeu : un en local permettant de jouer contre une intelligence artificielle et l'autre permettant de jouer contre un adversaire humain en réseau.

Dans ce rapport, nous allons présenter les différents points de l'application, notamment en décrivant le modèle du jeu, ensuite la partie réseau, puis l'interface homme machine permettant à l'utilisateur de jouer dans des conditions correctes, enfin nous exposerons dans une dernière partie les différentes connaissances que ce projet a apporté, les difficultés rencontrées ainsi que l'évolution possible de ce projet.

1 Modèle de jeu

1.1 Le plateau de jeu

Dans le cadre de ce projet, nous avons choisi de donner aux joueurs la possibilité de jouer différemment à la bataille navale, notamment de jouer en trois dimensions.

Pour cela, nous avons réfléchi puis développé un modèle capable de s'adapter aux besoins des joueurs. En effet, celui-ci peut décider de toutes les tailles, longueur, largeur ou encore profondeur, ainsi qu'un nombre de dimensions quelconques.

Le plateau de jeu permet donc de jouer en une dimension (soit sur la longueur ou soit sur la largeur), en deux dimensions (longueur et largeur), en trois dimensions nous ajoutons la profondeur pour enfin pouvoir jouer en plus de trois dimensions.

1.1.1 Définition récursive des Boards

Nous avons uns structure donnée qui se définit récursivement de la manière suivante :

— Une case d'un Board est un Board à zéro dimensions.

— Un Board à N dimensions, dont les tailles dans chaque dimension sont (TN, TN-1, ... , T1) où les Ti sont les tailles, est composé de TN Boards à N - 1 dimensions, dont les tailles dans chaque dimension sont (TN-1, ... , T1).

Notez bien que les Boards à N - 1 dimensions ont exactement les mêmes tailles pour chaque dimension. Ainsi, un Board à deux dimensions sera toujours représentable comme un rectangle, un Board à trois dimensions peut-être visualisé comme un pavé droit et ceux des dimensions supérieures suivent le même principe, bien qu'ils soient plus difficile à représenter.

On peut aussi voir le Board comme un arbre particulier avec les propriétés suivantes :

— La distance (plus court chemin de la racine à une feuille) et la hauteur (plus long chemin de la racine à une feuille) de l'arbre sont égales.

— Tous les nœuds d'un même niveau ont le même nombre de descendants.

Cette structure récursive nous permet d'instancier un Board de n'importe quel nombre de dimensions en lui passant tout simplement un tableau d'entier classique contenant les tailles dans chaque dimension.

1.1.2 Remplissage des Boards

Pour remplir un tableau classique en programmation, on utilise généralement autant de boucles imbriquées qu'il y a de dimensions au tableau. Dans le cas du Board, ce n'est pas possible à notre connaissance, puisque le nombre de dimensions est inconnu à la compilation et qu'il faudrait un nombre variable de boucles imbriquées dans le code.

Nous avons donc créé un itérateur de Boards à nombre de dimensions zéro, implémenté dans la classe interne de Board, DimZeroIterator qui donne un par un les Boards à zéro dimensions qui sont les cases du Board.

Cet itérateur est également construit de manière récursive. Il contient les DimZeroIterator correspondants aux Boards fils du Board qu'il itère et les méthodes next() et hasNext() appellent leurs homologues sur l'itérateur fils en cours de parcours.

Si on voit le Board comme un arbre, cet itérateur donne les cases dans l'ordre où un parcours en profondeur donnerait les feuilles.

1.1.3 Accès aux cases

L'accès en lecture et en écriture d'une case en particulier se fait grâce aux objets Coordinates. Ces objets sont techniquement des tableaux d'entiers non mutables, dont on peut tester l'égalité avec la méthode equals() redéfinie. L'accès à une case de ce tableau se fait avec la méthode get(int index), à la façon des listes, mais il est itérable comme un tableau classique.

Lors d'un appel à getItem() ou setItem() sur un Board, chaque composante de la Coordinates fournie en paramètre sert à déterminer sur quel Board de dimension inférieure il faut effectuer l'appel récursif.

Les objets Coordinates sont également utilisés pour instancier les Boards en fournissant le nombre de dimensions et la taille pour chaque dimension.

1.1.4 Détails de l'implantation

Les algorithmes d'accès aux cases de Board sont récursifs terminaux ; c'est à dire que lors de la compilation, le compilateur (si il y a compilation, ce qui est notre cas ici) peut remplacer ces appels récursifs par une simple boucle itérative. Cela s'explique par le fait que le rappel à soi-même soit la dernière instruction à être évaluée. De ce fait, cela permet d'économiser de la mémoire lors des appels à ces fonctions, car seul le dernier appel à la fonction doit être sauvegardé sur la pile d'exécution.
Structures des objets :

```
Structure Board :
    Champ Object : item;
    Champ Liste de Board : boards;
fin Structure

Structure Coordinates :
    Champ Tableau d'entiers : coord;
    Champ Entier : length;
fin Structure
```

1.1.4.1 Algorithmes de Coordinates

```
Algo Coordinates creerCoordinates(Tableau d'entier t) :
   coord <- copie de t;
   length <- longueur de t;
finAlgo

Algo Entier get(Coordinates c, Entier index) :
   renvoyer c.coord[index];
finAlgo
```

1.1.4.2 Algorithmes de Board

```
Algo Board creerBoard(Coordinates c) :
   renvoyer creerBoard(c, 0);
finAlgo

Algo Board creerBoard(Coordinates c, entier index) :
   Board b;
   b.item <- null;
    Si index = c.length  Alors
       renvoyer b;
    finSi
   b.boards <- nouvelle Liste de Board;
    Pour entier k de 0 à c.get(index)  Faire
       b.boards[k] <- creerBoard(c, index + 1);
    finPour
   renvoyer b;
finAlgo

Algo Object getItem(Board b, Coordinates c) :
   renvoyer getItem(b, c, 0);
finAlgo


Algo Object getItem(Board b, Coordinates c, Entier index) :
    si index = c.length  alors
       renvoyer b.item;
    finSi
   renvoyer getItem(b.boards.get(c.get(index)), c, index + 1);
finAlgo

Algo voidsetItem(Board b, Coordinates c, Object item) :
   setItem(b, c, 0, item);
finAlgo

Algo void setItem(Board b, Coordinates c, Entier index, Object item) :
    si index = c.length  alors
       b.item <- item;
    finSi
   setItem(b.boards.get(c.get(index)), c, index + 1, item);
finAlgo
```

1.2 Les navires en dimensions variables

Les navires sont représentés par la classe Ship. Ils ont une longueur, un nom et contiennent les coordonnées des cases du Board qu'ils occupent lorsqu'ils sont placés.

Dans une bataille navale classique, chaque navire est disposé verticalement ou horizontalement. Dans notre modèle à nombre variable de dimensions, il faut généraliser cette règle pour qu'elle fonctionne dans tous les cas.

1.2.1 Définition d'un bon placement

Pour être bien placés, les Ship doivent respecter les conditions suivantes :

— Bon alignement : les cases occupées par un Ship doivent avoir des coordonnées dont toutes les composantes sauf une sont égales, et les composantes qui ne le sont pas doivent se suivre.
Exemple : Dans un Board en quatre dimensions, un porte-avion (de longueur 5) occupe les cases (0,2,1,3), (0,2,2,3), (0,2,3,3), (0,2,4,3), (0,2,5,3). Ce placement respecte la condition de bon alignement car les premières, deuxièmes et quatrièmes composantes des coordonnés sont respectivement égales entre elles, et que les troisièmes composantes se suivent.

— Non dépassement : les cases occupées par un Ship sont celles contenues dans un Board.
Exemple : Dans un Board de dimensions (5,5,5,5), le porte-avion placé précédemment dépasse par sa dernière coordonnée, dont la troisième composante n'est pas comprise entre 0 et 4.

— Bonne longueur : un Ship doit occuper exactement le même nombre de cases que sa longueur.

— Non-Chevauchement : une case ne peut être occupée que par un seul Ship.

1.2.2 Tir sur les Ships

Les tirs sur les Ships se font via le Board sur lequel ils sont disposés. Pour cela, le type générique Board est paramétré soit par le type Case, soit par le type State.

Les Board<Case> de chaque joueur sont ceux sur lesquels ils placent

leurs navires, et les Board<State> représentent la grille de tir des joueurs, là où ils peuvent voir où ils ont tirés et pour quel résultat.

State est une énumération représentant le statut de tir de chaque case d'un Board<State>. Il existe quatre états possibles :

— NOT_AIMED : cette case n'a pas été visée par un tir.
— MISSED : il y a eu un tir sur cette case, mais il n'y avait pas de Ship adverse sur la case correspondante du Board<Case> du joueur adverse.
— HIT : Il y a eu un tir sur cette case, et un Ship adverse a été touché.
— SUNK : Même chose que HIT, avec en plus l'information que le Ship touché a été coulé par ce tir.

Les objets Case ont pour rôle de stocker deux informations : la présence d'un Ship et si la case a déjà reçu un tir. Ces deux informations suffisent à faire la distinction entre les états NOT_AIMED, MISSED et HIT/SUNK.

Un objet Case sert également à relayer un tir jusqu'au Ship qui l'occupe, s'il y en a un, et à s'assurer qu'on ne tire pas deux fois au même endroit. Cette condition rend non-nécessaire le stockage dans les Ship des endroits précis où ils ont été touchés : un Ship qui a été touché un nombre de fois égal à sa longueur a forcément été touché sur toutes les cases qu'il occupe, et on sait ainsi qu'il a été coulé.

Voici ce qui conclut la partie sur le modèles en tant que tels, vous avez vu comment est généré notre plateau de jeu ainsi, la façon dont on place nos navires sur cette grille de jeu ainsi que la façon dont se passent les tirs. Nous allons donc maintenant voir les intelligences artificielles développées dans le cadre du projet.

1.3 Les intelligences artificielles

Il nous a été demandé de pouvoir jouer à la bataille navale en local. De ce fait, nous avons du développer une Intelligence Artificielle (IA), car jouer à deux sur un même ordinateur présente de nombreux risques de triches.
Notre IA se compose de 3 niveaux de difficultés différents, allant de facile à difficile. Nous allons donc vous expliquer le fonctionnement de chacune de ces 3 IA.

1.3.1 IA Facile

L'IA facile fut la première intelligence artificielle programmée. Son cheminement est simple, elle consiste simplement à choisir aléatoirement une case, en vérifiant bien entendu que celle-ci n'ait pas déjà été visée au préalable, puis de tirer dessus, qu'importe l'avancement de la partie.
Le choix de l'aléatoire est fait de la façon suivante :

```
Fonction EasyAdvisor :
Entrée : Board un tableau de state contenant les resultats de tous les tirs.
         Dim le tableau des tailles des dimension de Board.
    Sortie : Coord une Coordonnée d'une case non touchée.
    Début
        Faire
           Entier distance = Alea(0, taille(Board)) ;
           Entier longueur = taille(Board) ;
            Pour (n allant de 0 à taille(Dim))
               longueur = longueur / Dim[taille(dim) - 1 - n) ;
               Ajout(Coord, taille(dim) - 1 - n, distance / longueur) ;
               Distant = distance % longueur
            finPour
        Tant que (Board[Coord] != NON_CIBLÉ) ;
       Retourner Coord ;
    Fin
Fin Procedure
```

1.3.2 IA Moyen

L'IA moyen, deuxième IA développée, fonctionne de manière similaire à l'IA facile, à ceci près qu'une fois un navire touché l'IA passe en mode "chasse".
Le mode chasse indique à l'IA de tirer sur les cases adjacentes à la case touchée et, si une nouvelle case est touchée, celle-ci part de cette dernière pour ainsi continuer jusqu'à avoir coulé le navire.
Une fois le navire coulé, notre IA reprend sa routine de "ciblage".
L'algorithme est présenté ci-dessous :

```
Procedure MediumAdvisor :
    Entrée : Board un tableau de State contenant le résultats de tous les tirs.
             Dim le tableau des tailles des dimension de Board.
             Coe une liste de case.
    Sortie : Coord une Coordonnée d'une case non touchée.

    Début
        Si estVide(Coe)
           Coodinate c = EasyAdvisor(Board, Dim) ;
            SI (Tir(enemy, c) != MANQUÉ)
               AjouterCasesAdjacentes(Coe, Coord) ;
            FinSi
           Retourner Coord ;
        Sinon
            Si (Board[Coe[0]] == NON_CIBLÉ)
               Coord = Coe[0] ;
               Retirer(Coe, 0) ;
               Retourner Coord ;
            FinSi
           MediumAdvisor(Board, Dim, Coe);
        FinSi
    Fin
Fin Procedure
```

1.3.3 IA Difficile

L'IA difficile, troisième IA développée, fonctionne de manière similaire à l'IA normale, c'est à dire qu'elle effectue sa routine de "ciblage" jusqu'à toucher un navire, puis passe en mode "chasse" une fois un navire touché. La différence avec l'IA normale est que cette IA ne tire pas de case adjacente l'une à l'autre. En effet, suivant la taille minimal du navire ennemi existant au début de partie, notre IA ne tire que toutes les cases séparées de cette taille. Par exemple, dans le cas ou le plus petit navire est un navire de 2, l'IA va donc écarter ses tirs de 2 cases d'écart à chaque fois.

```
Procedure HardAdvisor :
    Entrée : Board un tableau de state contenant les résultats de tous les tirs.
             Dim le tableau des tailles des dimension de Board.
             Coe une liste de case.
             Minsize la taille du plus petit navire présent sur la grille
    Sortie : Coord une Coordonnée d'une case non touchée.

    Début
        Si estVide(coe)
            Faire
               Entier distance = Alea(0, taille(Board)) ;
               Entier longueur = taille(Board) ;
               Entier mod = 0 ;
                Pour (n allant de 0 à taille(Dim))
                   Mod = 0 ;
                   longueur = longueur / Dim[taille(dim) - 1 - n) ;
                   Ajout(Coord, taille(dim) - 1 - n, distance / longueur) ;
                   Distant = distance % longueur
                   Mod = (mod + distance /longueur) % minsize;
                FinPour
            Tant Que(mod != 0 || Board[Coord] != NOTAIMED) ;

            SI (Tir(enemy, coord) != MISSED)
               AjouterCasesAdjacentes(coe, coord) ;
            FinSi
           Retourner Coord ;
        Sinon
            Si (Board[Coe[0]] == NOTAIMED)
               Coord = coe[0] ;
               Retirer(coe, 0) ;
               Retourner Coord ;
            FinSi
        FinSi
```

```
        MediumAdvisor(Board, Dim, Coe) ;
    Fin
Fin Procedure
```

Voici ce qui conclut notre première partie présentant le modèle de notre application, nous avons donc vu dans un premier temps le fonctionnement de notre grille de jeu, qui est donc une grille modulaire à un nombre de dimension infini. Nous avons ensuite expliqué comment fonctionne nos navires, et la façon dont on tire dessus. Puis pour finir, nous avons montré le fonctionnement de nos IA ainsi que les différences entre les 3 niveaux de difficultés. Nous allons donc maintenant passer sur la seconde partie du projet qui est celle du réseau.

2 Le modèle réseau

2.1 Protocoles

2.1.1 Protocole Réseau

Le protocole réseau [1] utilisé pour que deux joueurs puissent se connecter et jouer en réseau est le protocole TCP [2] qui est un protocole en mode connecté mais également fiable quant à la connexion entre deux machines et pour la réception des paquets envoyés sur le réseau.

Nous avons préféré le protocole TCP au protocole UDP [3] car, premièrement, son implémentation est bien plus aisée. En effet, nous n'avons pas à traiter certains cas d'erreurs, notamment dans le cas d'un paquet qui soit mal formé ou incomplet car TCP assure que celui-ci sera totalement prêt avant de l'envoyer.
Deuxièmement, malgré le fait que le protocole UDP soit plus rapide dans l'envoi des données, utiliser le protocole TCP dans notre cadre n'a pas d'impact significatif puisque, comme il sera exposé un peu plus tard, chaque joueur doit attendre la réception de données avant de continuer son traitement. Ainsi, il n'est pas, en pratique, possible de recevoir des données qui ne sont pas attendues à des moments précis.

2.1.2 Protocole de l'application

L'application doit suivre un certain protocole pour que les deux joueurs puissent communiquer correctement et sans problèmes majeurs. Il a donc été décidé de suivre un type de protocole classique qui est celui du question/réponse. Néanmoins, nous n'avons pas choisi d'implémenter de serveur d'hébergement centralisant toutes les parties et dirigeant toutes les actions des clients, nous avons opté pour un échange de type pair à pair [4], c'est à dire que les deux joueurs doivent jouer les deux rôles : le processus serveur et le processus client, chacun à des temps déterminés et non mutables.
Nous expliquerons dans cette partie la connexion entre les deux joueurs, comment se passe l'échange de données ainsi que le format que les données doivent respectées. Les deux premières parties seront également accompagnées de schémas explicatifs sous forme d'automates disponibles en annexe.

1. Tous les termes demandant explications sont disponibles dans le lexique
2. Transmission Control Protocol : https ://tools.ietf.org/html/rfc793
3. User Datagram Protocol : https ://tools.ietf.org/html/rfc768
4. Voir lexique

2.1.2.1 Connexion des joueurs

Comme dit plus haut, chaque joueur doit jouer deux rôles bien distincts, celui du processus serveur comme celui du processus client. Dans ce cas-ci, il est donc bon de noter l'ordre des différentes opérations effectuées entre un joueur qui va accueillir la partie et celui qui va en être l'invité, nous allons donc expliciter quelles actions font parties du processus client et lesquelles font parties du processus serveur.

Les deux joueurs vont, en premier lieu, créer et lancer leurs serveurs respectifs, en spécifiant l'adresse de la machine en format IPv4, un port d'écoute (nous vérifions qu'il soit compris entre 1024 et 65535), ainsi qu'un nombre de connexions autorisées qu'on nommera "backlog".

Ensuite, le joueur hôte met son serveur en attente d'une demande de connexion via sa socket d'écoute [5] (il est donc considéré en tant que processus serveur) et, de son côté, le joueur invité (considéré comme le processus client à ce moment) va se connecter à l'adresse du joueur hôte. Cela aura pour effet, côté processus serveur, de créer une socket de service [6] par laquelle le joueur passera pour communiquer avec le joueur invité.

À cette étape, il reste à faire la connexion inverse, c'est à dire que le joueur hôte (qui est maintenant le processus client) doit faire une demande de connexion à l'adresse du serveur du joueur invité (devenu le processus serveur) et, une fois la connexion acceptée par la socket d'écoute du processus serveur, une nouvelle socket de service est créée pour que le joueur invité puisse communiqué avec le joueur hôte.

Suite à ces différentes étapes, la connexion entre le joueur hôte et le joueur invité est établie, les échanges de données peuvent donc être effectués.

2.1.2.2 Échanges des données

Mais avant de pouvoir commencer la partie et à échanger des données, chaque joueur doit placer ses navires puis attendre que l'adversaire finisse de placer les siens. Pour nos besoins, nous n'avons pas décidé de laisser l'application choisir au hasard qui commence une partie : ce sera toujours le joueur hôte.

5. Voir lexique
6. Voir lexique

De ce fait, le joueur hôte devient donc le processus client qui doit alors choisir puis envoyer les coordonnées de son tir, ces dernières étant réceptionnées par le processus serveur, i.e le joueur invité, qui est en attente de ces coordonnées et rien d'autre. En effet, nous verrons plus tard comment ces données sont formatées pour que notre protocole fonctionne correctement (voir 2.1.2.3).

La prochaine étape consiste, pour le joueur invité maintenant devenu le processus client, d'envoyer le résultat du tir qui est un état de la case visée. Cet état peut être de trois natures : "touché", "coulé" ou bien "manqué". Le joueur hôte, quant à lui, attend l'arrivée de ce résultat en tant que processus serveur.

Enfin, les joueurs mettent à jour leur vue respective pour finalement changer le tour de jeu, c'est à dire que le joueur hôte passe la main au joueur invité.

Ce schéma, se répète quand c'est au tour du joueur invité de jouer, il suffit simplement d'inverser les rôles pour obtenir un tour de jeu complet et ce jusqu'à la victoire d'un des joueurs qui fermeront correctement ensuite les différentes sockets [7] mises en jeu.

2.1.2.3 Format des données

Dans le cadre de notre protocole, il a fallu décider d'un certain format que les données doivent respectées pour pouvoir être acceptées et ainsi traitées. De ce fait, tout paquet est formé de cette manière afin de faciliter l'extraction des données utiles au protocole :

"Objet :Valeur"

Ces deux composantes définissent donc le format des données envoyées. Nous allons donc maintenant définir quels sont les types d'objets possible ainsi que leurs valeurs associées.

1. "Coordinates :(X,)*X" ⇒ Ce format permet d'envoyer les coordonnées d'un tir, la valeur est une suite de nombres séparés par des virgules représentant la coordonnée sur un axe donné. Par exemple, si nous recevons "Coordinates :1,2", cela signifie que nous voulons tirer sur la case dont les composantes de coordonnées sont 1 pour la première composante et 2 pour la seconde.
2. "State :X" ⇒ Ce format permet d'envoyer l'état de la case qui a été visée. Les valeurs possibles ont déjà été définies dans la partie 2.1.2.2.

7. Voir lexique

Par exemple, en recevant "State :HIT", cela signifie que la case visée a été touchée.

3. "String :X" ⇒ Un objet String est une chaîne de caractères, la valeur pouvant être n'importe quelle chaîne. Grâce à ce format, nous pouvons envoyer des messages simples, comme le fait d'avoir gagné la partie ou bien un message d'erreur que nous affichons à l'utilisateur.

2.2 Gestion des erreurs

La gestion des erreurs est faite sur plusieurs niveaux d'abstraction dans notre projet. En effet, le premier niveau d'abstraction est le fait d'utiliser le protocole TCP/IP, car celui-ci, en opposition au protocole UDP, en plus de recevoir le paquet fait une vérification afin de valider le paquet avant de le transmettre. De ce fait, une partie des vérifications est déjà effectuée par ce biais.

Le second niveau d'abstraction est lié au langage que nous avons choisi. En effet, Java implémente les protocoles TCP et UDP. Hors, Java effectue aussi des vérifications quand à la validité des données et permet facilement de vérifier si le paquet reçu est valide ou non, par le biais des Exceptions présentes dans l'API de base.

Au final, nous avons implanté notre propre protocole de validité des données, afin d'être sûr de ce que nous recevons. Comme vous avez pu le lire précédemment, chacune de nos données envoyées possède un format propre a notre protocole et si, lors d'un envoi de donnée, nous n'avons pas le bon type de données, alors nous gérons le problème en indiquant qu'un problème est survenue lors de l'envoi des données, ce qui n'est pas normal et qui indique un problème au niveau du programme. De ce fait, nous fermons donc proprement l'application afin d'éviter tout problèmes lors d'une exécution prolongée du logiciel.

Voici ce qui conclut notre présentation de la partie réseau du projet. Nous avons donc vu ensemble le fonctionnement de notre protocole réseau ainsi que la façon dont nous avons géré les potentielles erreurs pouvant survenir dans le cadre d'une partie en réseau. Il nous reste donc plus qu'une seule partie de l'application, il s'agit de l'IHM [8], que nous allons vous expliquer dans la prochaine partie.

8. Interface Homme-Machine, aussi appelé Graphical User Interface, voir lexique

3 Interface Homme Machine

Dans toute application où les interactions avec l'utilisateur sont fréquentes, et notamment dans un jeu tel que la bataille navale, il est bon que cette application possède une Interface Homme Machine (IHM) [9] qui va prendre en compte les actions de l'utilisateur et modifier ce qu'il voit en conséquence.

Nous avons donc développé une IHM permettant à un joueur de pouvoir jouer aisément et sans difficultés. Notre application se découpe en deux fenêtres principales. La première est la fenêtre de démarrage où l'utilisateur pourra définir les différentes options liées à la nouvelle partie et la seconde est l'interface principale du jeu où toute la partie se déroulera. Nous allons donc exposer ces deux fenêtres dans deux parties distinctes où nous expliquerons comment les fenêtres sont construites et détaillerons les différents actions à effectuer. Enfin, nous en profiterons pour mettre en avant, dans une troisième partie, les composants graphiques permettant l'affichage des Boards.

3.1 La fenêtre de démarrage

Comme dit plus haut, la fenêtre de démarrage est la première fenêtre que verra le joueur au lancement de l'application. Pour nous aider à décrire cette fenêtre, voici une capture d'écran :

FIGURE 1 – Fenêtre de démarrage de l'application

Comme on peut le voir, la fenêtre de démarrage est séparée en plusieurs morceaux. Pour s'y repérer, nous allons d'abord nous intéresser aux différents champs à remplir et actions à effectuer pour pouvoir créer une partie locale contre une IA, puis nous réitérerons cette logique pour soit créer, soit rejoindre une partie en ligne.

3.1.0.1 Une partie en local

Pour effectuer une partie en local, il faut premièrement indiquer, dans le champ de texte en haut à droite symbolisé par un "A", les dimensions

9. Voir lexique

de notre plateau de jeu, chaque dimension étant séparée par le symbole ",". Puis, nous devons choisir, grâce à la liste déroulante symbolisé par un "B" à gauche, le niveau de difficulté choisi pour l'IA qui sont, à l'heure actuelle, au nombres de trois. Enfin, Nous pouvons lancer la partie en cliquant sur le bouton "commencer une partie contre l'IA, qui se situe en bas à gauche symbolisé par un "C".

3.1.0.2 Une partie en réseau

Pour pouvoir jouer en ligne, il faut bien entendu indiquer les dimensions du plateau de jeu, cependant cette information n'est nécessaire que pour le joueur qui hébergera la partie. Comme pour la partie en local, il faut remplir le champ de texte marqué d'un "1" en haut à droite de la fenêtre. Ensuite, chaque joueur doit rentrer son adresse IPv4 dans le champ numéroté "2" et l'adresse IPv4 de l'adversaire dans le champ numéro "3" au centre de la fenêtre.
Il est également possible de récupérer une adresse mémorisée au préalable dans un fichier texte, apparaissant dans les listes déroulantes notées "4" mais aussi de mémoriser l'adresse, qui est écrite dans les champ de textes sus-mentionné, grâce aux boutons sur la droite.
Enfin, le joueur hôte lance la partie en cliquant sue le bouton "créer une partie en ligne", noté "6" en bas au centre de la fenêtre et le joueur invité peut rejoindre une partie après avoir cliqué sur le bouton indiqué par un "7".

3.2 L'interface du jeu

3.2.1 Le placement des navires

Avant de pouvoir jouer, il est nécessaire que les joueurs puissent placer leurs navires. Ce placement est illustré dans la figure suivante :

Le joueur doit avant tout cliquer sur une des cases du plateau de jeu, ce composant graphique un peu spécial sera explicité dans une autre sous-partie. La case cliquée représentera alors la proue du navire("1") et, en cliquant sur une seconde case, nous choisissons la poupe("2") de celui-ci. Il est à noter que, sur la gauche de la fenêtre, il est possible d'entrer les coordonnées des deux cases en format textuel, c'est à dire "X,X"("3").

Il faut ensuite cliquer sur le bouton "Placer ce navire"("4"), en dessous des champs de texte permettant de renseigner les coordonnées. Le fait de cliquer sur ce bouton a pour effet de faire passer le navire à placer au suivant dans la liste déroulante en haut à gauche("7"). Naturellement, il

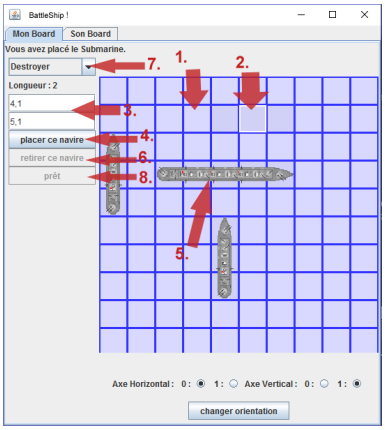

FIGURE 2 – Placement des navires

est aussi possible de retirer un navire précédemment placé en le choisissant dans cette même liste et en cliquant dur le bouton "retirer ce navire" ("6").

Enfin, une fois tous les navires placés, le bouton "prêt"("8") est activé et nous pouvons cliquer dessus pour débuter une partie.

Toutes ces étapes mènent au commencement de la partie. Nous allons donc maintenant décrire ce que l'utilisateur voit et peut faire lors d'un tour de jeu sur une partie déjà avancée.

3.2.2 Une partie typique

Pour comprendre l'interface mise en œuvre ici, il est nécessaire de pouvoir la visualiser correctement. Pour cela, une capture de la fenêtre est donnée ci-après :

Sur cette image, nous pouvons tout d'abord parler du plateau de jeu. En effet, celui ci voit ses cases changées au cours de la partie suivant si la case a été ciblé on non ("1"). Si elle a été ciblé, cette case peut marqué comme "manqué" ("A"), "touchée" ("B") ou ecnore "coulée" ("C").

Lors du tour du joueur, celui-ci doit renseigner la case ou les coordonnées sur lesquelles il souhaite tirer. Il a donc à sa disposition la possibilité de soit cliquer directement sur la case voulue("1") ou alors de rentrer les coordonnées de la case au format textuel comme défini plus haut ("2"). Le joueur fait ensuite feu grâce au bouton "Feu !" ("3") et obtient le résultat de son tir qui affiche donc une image différente pour les trois cas possibles("A",

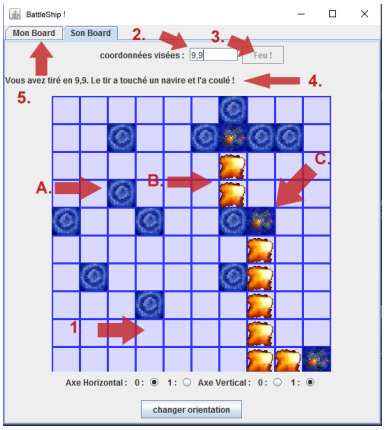

FIGURE 3 – Une partie à un stade avancée

"B" ou "C") ainsi qu'un petit texte décrivant la nature du résultat ("4"). Le joueur a également la possibilité de regarder son plateau de jeu, en cliquant sur l'onglet "Mon Board" ("5").

Voici typiquement comment l'IHM s'arrange et les différentes actions que le joueur peut effectuer au cous d'une partie.

Nous allons maintenant évoquer et expliquer le composant graphique majeur de l'application, le plateau de jeu qui se décline en plusieurs composants suivant quel plateau nous souhaitons regarder : soit le nôtre avec la position de nos navires, soit celui de l'adversaire contenant tous les résultats de nos tirs.

3.3 La représentation du plateau de jeu

La représentation d'un espace à deux, trois ou plus de dimensions se complexifie avec le nombre de dimensions. Nous avons choisi de représenter les plateaux à deux dimensions de manière classique, une surface avec un quadrillage. La représentation des plateaux à trois dimensions utilise une succession de plateaux à deux dimensions empilés entre lesquels on peut naviguer facilement.

Pour les nombres de dimensions supérieures, nous avons fait le choix de ne pas tout représenter étant donné la difficulté de visualiser un espace à quatre dimensions pour les humains et l'absence de conventions communément partagées sur la représentation de cet espace.

3.3.1 Explications sur les axes

Plutôt que de montrer une représentation qui ne serait pas comprise par les joueurs, nous avons opté pour un système de composantes fixes pour les dimensions non représentées. Ainsi, pour représenter un Board de dimensions (5, 5, 5, 5), on peut fixer la quatrième composante de chaque case représentée à la valeur t : les cases représentées seront celles dont les coordonnées sont (x, y, z, t) pour x, y et z entre 0 et 4, et pour t fixé. Les cases de coordonnées dont la quatrième composante n'est pas égale à t ne sont pas représentées.

Nos composants graphiques utilisent des objets Coordinates d'une façon spéciale pour déterminer les axes et les composantes fixées des coordonnées.

Dans ces coordonnées, les composantes de valeurs positives ou nulles représentent des composantes fixées et les valeurs -1, -2, -3 désignent respectivement qu'il faut représenter les composantes correspondantes sur l'axe X (horizontal, de gauche à droite), l'axe Y (vertical, de haut en bas) ou sur l'axe Z (en profondeur, vers l'écran).

Exemple : Dans une représentation en 3 dimensions d'un Board de dimensions (5, 4, 5, 3, 6), si on fixe l'axe à (-1, 2, -3, 1, -2). Les cases représentées seront celles de coordonnées (a, 2, b, 1, c) avec a, b et c des valeurs entières dans les limites des dimensions du Board. Chaque case représentée sera située à la a-ième place sur l'axe des X, la c-ième place sur l'axe des Y et la b-ième place sur l'axe des Z.

Dans la suite et dans le code source, nous appelons "axes" ces objets qui déterminent quelles composantes sont représentées sur quels axes et lesquelles sont fixées.

3.3.2 Représentation en deux dimensions : les GraphicBoards

Un objet GraphicBoard est un JComponent permettant la représentation en deux dimensions, il connaît le Board qu'il représente ainsi qu'un axe pour déterminer quelles cases du Board il doit représenter et dans quel sens.

En plus de ça, le GraphicBoard contient deux modes d'interaction : un passif et un actif. Le mode passif répond aux clics de souris en envoyant un événement simple dont la sémantique est tout simplement "on m'a cliqué dessus".

Le mode actif répond aux clics de souris par un CoordinatesEvent contenant les coordonnées de la case cliquée dans le modèle du Board. Ce ne sont pas les coordonnées "graphiques" mais bien celles correspondantes à la case du Board dont on a cliqué sur la représentation graphique.

3.3.3 Représentation en trois dimensions : Les GraphicBoardLayers

Un GraphicBoardLayer est un JPanel contenant plusieurs GraphicBoards en couches superposées. Il permet de naviguer entre ses couches de plusieurs manières : en cliquant sur la couche qu'on veut voir entièrement (elles sont toujours cliquables car la superposition est en décalé), en utilisant le slider sur le coté ou avec la molette de souris.

Le GraphicBoardLayer s'assure qu'un seul GraphicBoard est en mode d'interaction actif et qu'il soit entièrement visible (en décalant ceux du dessus).

Il contient également des options de changement de la représentation, comme les changements d'axes, avec des boutons radios, et dans le cas où le Board qu'il représente contient plus de trois dimensions, il permet de fixer les composantes non représentées sur les axes.

Le GraphicBoardLayer relaie les CoordinatesEvents du GraphicBoard en mode actif.

3.3.4 Délégation de l'affichage spécifique : les BoardDrawer

GraphicBoard et GraphicBoardLayer sont des classes génériques, aptes à représenter les plateaux sur lesquels sont placés les navires comme les plateaux de tir. L'affichage spécifique de chaque type de plateau ne peut donc pas être pris en charge.

Cet affichage est donc délégué à des objets de type BoardDrawer, qui est une interface elle aussi générique, mais que nous avons implanté en deux classes non génériques, ShipDrawer et ShootDrawer.

3.3.5 La représentation des tirs : GraphicBoardShooter

Le GraphicBoardShooter contient un GraphicBoardLayer dont il récupère les CoordinatesEvents pour remplir un champ de coordonnée de tir. Il

est doté d'un bouton "feu" qui s'active si les deux conditions suivantes sont remplies :

— La case correspondante aux coordonnées actuelles n'a pas encore été visée.

— C'est le tour du joueur (attribut booléen myTurn qui passe systématiquement à Faux après un tir, et que le contrôleur de jeu peut passer à VRAI ou FAUX) : C'est ce contrôleur (LocalController ou SuperController) qui donne donc l'autorisation de tirer avant chaque tir.

Comme tout n'est pas forcément visible, un label indique la dernière coordonnée de tir et son effet sur l'adversaire (s'il a touché ou coulé un navire).

3.3.6 La représentation des navires : GraphicShipBoard

Le GraphicShipBoard contient lui aussi un GraphicBoardLayer dont il récupère les CoordinatesEvents pour remplir alternativement deux champs de coordonnées (c'est-à-dire qu'il remplit le premier au premier événement, le second au second événement, puis de nouveau le premier, etc).

Avec ces deux champs de coordonnées qui représentent les coordonnées potentielles des extrémités d'un navire, le joueur peut essayer de placer le navire actuellement sélectionné dans la liste déroulante, s'il n'est pas déjà placé. Si les conditions d'un bon placement de navires sont réunies, le navire sera alors placé. Sinon, un message s'affichera, expliquant pour quelle raison le navire n'a pas pu être placé.
Lorsque tous les navires sont placés et que le joueur clique sur le bouton "prêt", le panneau de placement des navires disparaît. Le GraphicShipBoard représente les tirs subis en plus des navires.

Nous avons donc vu ensemble la partie graphique de l'application, de son démarrage jusqu'au déroulement d'une partie classique. Il nous reste une dernière partie à présenter, qui est la partie "humaine" du projet, c'est à dire les choses que nous avons acquis au cours du projet, les divers problèmes rencontrées, les idées d'améliorations potentielles ainsi que les technologies que nous avons utilisées tout au long du projet.

4 Autres

Dans cette partie, nous allons présenter les aspects "humains" du projet, c'est à dire notre ressenti lors du projet, les technologies qui nous ont servis tout au long du projet, nos idées d'améliorations que nous proposons pour les futurs développeurs souhaitant reprendre notre projet par exemple, les problèmes que nous avons pu rencontrer au cours du développement de notre application, ou bien encore ce que nous avons appris au cours du projet et ce que celui-ci nous a apporté.

4.1 Technologies utilisés

Les technologies que nous avons utilisé durant le projet sont les suivantes.
La première technologie utilisée fut le langage Java. En effet, nous avons choisi de prendre le Java, car notre client nous l'a conseillé, en particulier pour ce qui est la gestion de l'IHM. De plus, étant un langage que nous maîtrisons tous plus ou moins, nous avions ainsi davantage de facilité à le manier et donc à développer le projet voulu.
La seconde technologie fut l'utilisation de GIT, un logiciel de gestion de version de code source. En effet, via GIT, nous pouvions aisément travailler sur des partie du code différentes sans interférer avec les autres. De plus, cela nous permettait, en cas de soucis, de récupérer le code antérieur pour repartir dessus si des améliorations, ou des modifications apportées au code courant modifiaient trop le fonctionnement de l'application, ou pire, créaient des bugs non réparable.
La troisième technologie que certains d'entre nous apprirent fut le LaTeX, car nous tenions a rendre un projet propre, agréable a lire et surtout "professionnel", hors LaTeX est la technologie la plus adapté a nos besoins, c'est pourquoi deux personnes furent chargées d'apprendre le LaTeX afin de rendre le compte-rendu via ce langage.

4.2 Idées d'améliorations

Voici une liste d'améliorations possibles qui pourraient être implantées sur le projet afin de le rendre encore plus complet.
La première amélioration serait d'ajouter une IA très difficile, qui fonctionnerait de la même façon que l'IA difficile, avec cependant une variante qui consisterait à adapter les coups qu'elle tire en fonction des bateaux touchées et des cases restantes sur la grille. Nous avions développé en ce sens une classe pouvant calculer toutes les positions possibles des navires, mais dont l'exécution demandait trop de ressources de la machine.
La seconde amélioration possible serait le fait qu'un thread utilisateur soit lancé lorsque nos sockets de service sont en attente. En effet, lorsque

celles-ci sont en attente, EDT [10] le thread gérant toute l'IHM se retrouve aussi bloquer due a l'attente de réception de données.
Une troisième amélioration potentielle serait d'améliorer l'IHM afin que celle-ci affiche un beau menu lors de l'ouverture du logiciel, afin que les utilisateurs aient vraiment la sensation de jouer à un jeu.
Dans le même ordre d'idées, la possibilité de changer les images représentant les navires serait un plus indéniable. Ce n'est pas possible dans l'application actuelle, mais une option permettant de sélectionner un répertoire d'images est déjà implémentée.
Nous pourrions aussi ajouter le Drag'n'Drop, appelé aussi "Glissé-Déposé", pour placer plus facilement nos bateaux sur la grille, lors du début d'une partie. En effet, nous plaçons actuellement nos bateaux par le biais d'un clique plaçant l'avant du bateau, puis l'arrière du bateau par un second clique, et le faire par Drag'n'Drop serait plus intuitif pour certaines personnes.
Il serait aussi possible d'offrir la liberté d'ajouter plus de bateaux, ou de modifier ceux existant, par exemple en modifiant leur taille, ou encore en jouant avec 7 bateaux au lieu de 5 initialement dans les règles officielles de la bataille navale. Là encore, cette possibilité est partiellement implémentée.
Une autre amélioration développable serait de pouvoir changer les règles ou d'ajouter des règles durant une partie. Par exemple, nous pourrions utiliser ajouter un chronomètre lors d'un tour de jeu, afin que chaque joueur doive jouer rapidement, sous peine de voir son tour être passé sans avoir eu le temps de jouer. Nous pourrions aussi ajouter un chat pour que les joueurs puissent communiquer entre eux, afin qu'ils puissent par exemple discuter entre eux, ou encore se charrier gentiment lors d'un coulé par exemple.
Comme nous l'avons vu, ce projet ne possède guère de limite si ce n'est notre imagination.

4.3 Problèmes rencontrés

De nombreux problèmes ont été rencontrés au cours du projet, les plus important d'entre eux sont le fait que nous avons manquée de temps pour le projet, ainsi que le manque de salle informatique pour travailler efficacement, notamment le vendredi après midi, ou aucune salle informatique n'était libre pour nous permettre de travailler tous ensemble.
Un autre problème que nous avons rencontré est le fait que nous n'avions pas de redirection de port, ne permettant pas de tester facilement la partie réseau du projet, car il faut absolument que chacun possède son point d'accès, pour pouvoir le modifier, ce qui n'était pas forcément le cas.
Nous avons rencontré aussi des problèmes organisationnels et logistiques vis-

10. Voir lexique

à-vis du travail de chacun au début, en particulier sur le début du projet.
Au niveau du développement, nous avons rencontré un problème au niveau de l'uniformisation du code. En effet, nous avions convenu initialement de coder en anglais, langue universelle de l'informatique. Or, plusieurs fois nous avons créé des noms de variables ou encore de méthodes écrites en français, pour des raisons de compréhension sur certains termes, notamment du vocabulaire maritime, faisant ainsi une grosse différence entre chaque développeur présent sur le projet.
Toujours au niveau du code, nous avons eu de grosse difficultés à manipuler les threads, en particulier le lien entre EDT et les thread utilisateurs, ce qui explique que cette gestion n'ait pas été réalisée et implantée dans le code, malgré de nombreuses heures passées dessus.
Nous nous sommes également heurtés à la difficulté d'implantation du Drag'n'Drop, l'API java ne convenant pas à nos besoins sur ce sujet.
Pour terminer, une dernière difficulté s'est aussi présenté à nous, en effet, l'appréhension et la visualisation du modèle permettant des grilles à nombre de dimensions variable et inconnu a été assez difficile à comprendre, car au delà de la troisième dimension, nous étions régulièrement perdu (et perdant contre l'IA).

4.4 Compétences acquises

Au cours du projet, nous avons acquis nombre de connaissances tels que par exemple l'initiation a GIT, un logiciel de versionning de code, permettant ainsi un travail efficace en groupe, car chacun peut facilement travailler sur un module du projet sans risquer de "casser" le code fonctionnel.
Nous avons aussi acquis de nombreuses connaissances sur le réseau, en particulier sur le protocole TCP/IP qui est un standard d'internet, ainsi que le fonctionnement d'un protocole question/réponse et pour finir l'utilisation des sockets par le biais du langage Java.
La réflexion pour la modélisation d'un tableau à nombre de dimensions variable nous a également été profitable, nous poussant à sortir des sentiers battus de l'algorithmique (du moins, des sentiers que nous avions parcouru jusque-là).
Pour finir, nous avons acquis la capacité de gérer et de se répartir le travail efficacement lorsqu'on travail en groupe.

Conclusion

Ce projet s'est révélé très enrichissant dans la mesure où nous avons pu utiliser toutes nos compétences afin de produire une application la plus fonctionnelle, la plus propre et la plus adaptée au besoin de notre client M.GUESNET.
Ce projet nous a aussi permis de mettre en corrélation les parties théorique et pratique vues en cours afin de produire un projet complet, reliant divers aspect de l'informatique, nous pensons notamment à la partie réseau, couplée à l'IHM et au modèle MVC que nous avons étudié jusqu'ici. Nous avons acquis nombre de nouvelles compétences qui nous seront très utiles dans nos projets futurs, nous pensons notamment à GIT, qui est un outil extrêmement puissant et efficace quand il s'agit de travailler en groupe, ou encore à LaTeX, qui nous permettra de rendre des rapports très beaux et fonctionnels.
Tout en restant positif, nous n'oublions pas les nombreuses difficultés que nous avons rencontrées tout au long du projet, à savoir principalement la difficulté de travailler dans un groupe, être capable d'uniformiser le code, ou l'apprentissage de nouvelles technologies.
Pour conclure, nous souhaitons que notre production soit appréciée par tous et qu'un jour, des développeurs aient envie de reprendre le projet afin de l'améliorer.

Annexe A Lexique

— EDT : EDT est un acronyme signifiant Event Dispatch Thread. Concrètement, EDT est le thread principal s'occupant de la partie graphique d'une application Java, c'est-à-dire le dessin de l'IHM ou la gestion des interactions avec l'utilisateur.

— Itérateur : En programmation, un itérateur est un objet permettant de parcourir un tableau ou une collection, non pas nécessairement depuis le début du tableau, mais depuis n'importe quel index du tableau. Dans le cadre de notre projet, notre itérateur nous permet de parcourir notre Board.

— Pair à pair : Le pair à pair (ou peer-to-peer en anglais), est un modèle de réseau permettant à deux machines de discuter d'égale à égale. Dans les faits, cela s'explique par le fait qu'une machine se connecte à une autre machine et inversement afin que celles-ci puissent s'échanger des informations sans passer par un serveur distant.

— Protocole réseau : Un protocole est une méthode standard qui permet la communication entre des processus (s'exécutant éventuellement sur différentes machines), c'est-à-dire un ensemble de règles et de procédures à respecter pour émettre et recevoir des données sur un réseau. Il en existe plusieurs selon ce que l'on attend de la communication. Certains protocoles seront par exemple spécialisés dans l'échange de fichiers (le FTP), d'autres pourront servir à gérer simplement l'état de la transmission et des erreurs (c'est le cas du protocole ICMP), ...

— Protocole TCP : Acronyme de Transmission Control Protocol, le protocole TCP/IP est le protocole standard utilisé sur internet, pour la liaison entre deux ordinateurs. Le protocole TCP vérifie la validité des paquets après leur réception afin d'être sûr de la validité de celle-ci. Le protocole TCP est située sur la couche 4 (couche de transport) du modèle OSI.

— Protocole UDP : Acronyme User Datagram Protocol, le protocole UDP est un des protocoles standards utilisé sur internet. La différence avec TCP est que les paquets sont reçus sous forme de datagrammes qui doivent être vérifiés pour valider la qualité du paquet reçu. Le protocole UDP est très utilisé, notamment, dans le cadre du jeu en ligne, ou encore le streaming, car la perte de paquet influe peu sur la

quantité reçus. Le protocole UDP est située sur la couche 4 (couche de transport) du modèle OSI, au même titre que le protocole TCP.

— Socket : Une socket est une interface de connexion bidirectionnelle permettant l'échange de données entre deux processus (distants ou non).

— Socket de Berkeley : Les sockets de Berkeley, sont un ensemble normalisés de fonctions de communications lancé par l'université de Berkeley au début des années 1980. De nos jours, elle est la norme utilisé par quasiment l'ensemble des langages de développement (C, Java, Python, ...).

— Socket d'écoute : Une socket d'écoute est une socket présente uniquement dans le protocole TCP. En effet, son rôle consiste, comme son nom l'indique, à écouter les demandes de connexion de socket externe sur un port prédéfini, afin de créer une socket qui permettra ensuite l'échange de données avant de reprendre son rôle d'écouteur.

— Socket de service : La socket de service est la socket crée par la socket d'écoute lorsque celle-ci reçoit une demande de connexion. La socket de service permet la communication entre le serveur et le client ayant fait une demande de connexion. C'est par cette socket que transitent toutes les données émises par le client et le serveur.

— Thread : Un thread est une sorte de processus, dit "léger". Le rôle d'un thread est d'exécuter une suite d'instruction précise que l'on peut nomme routine. Le fait de lancer une application informatique lance automatiquement un thread, celui-ci peut alors créer d'autres threads afin de délégué par exemple une tâche longue a un autre thread, afin que le thread principal (main thread), puisse continuer son fil d'exécution à lui.

Annexe B Webographie

— Fonctionnement des intelligences artificielles : `http://www.datagenetics.com/blog/december32011/index.html`

— API Java : `https://docs.oracle.com/javase/7/docs/api/`

— Règles de la bataille navale : `http://www.regles-de-jeux.com/regle-de-la-bataille-navale/`

Annexe C Schéma du protocole de l'application

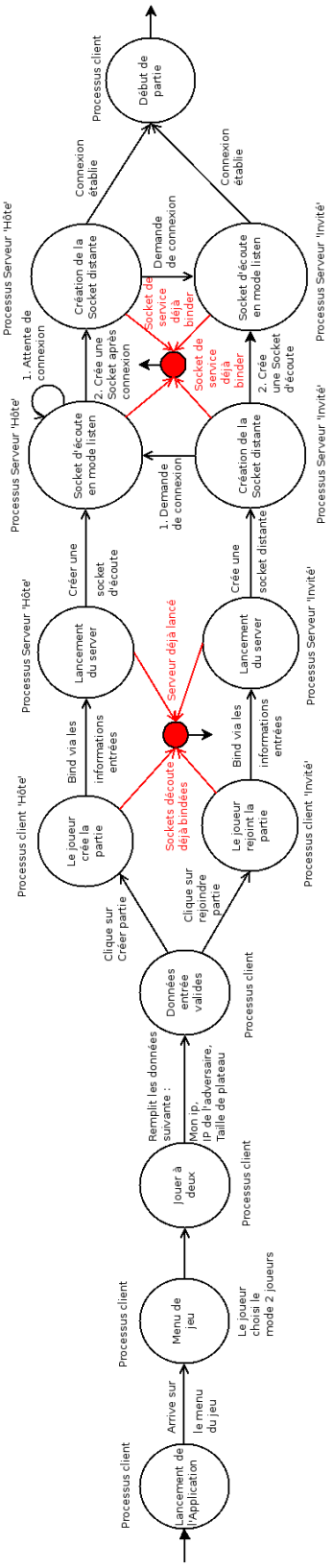

FIGURE 4 – Schéma représentant le protocole de connexion entre deux joueurs

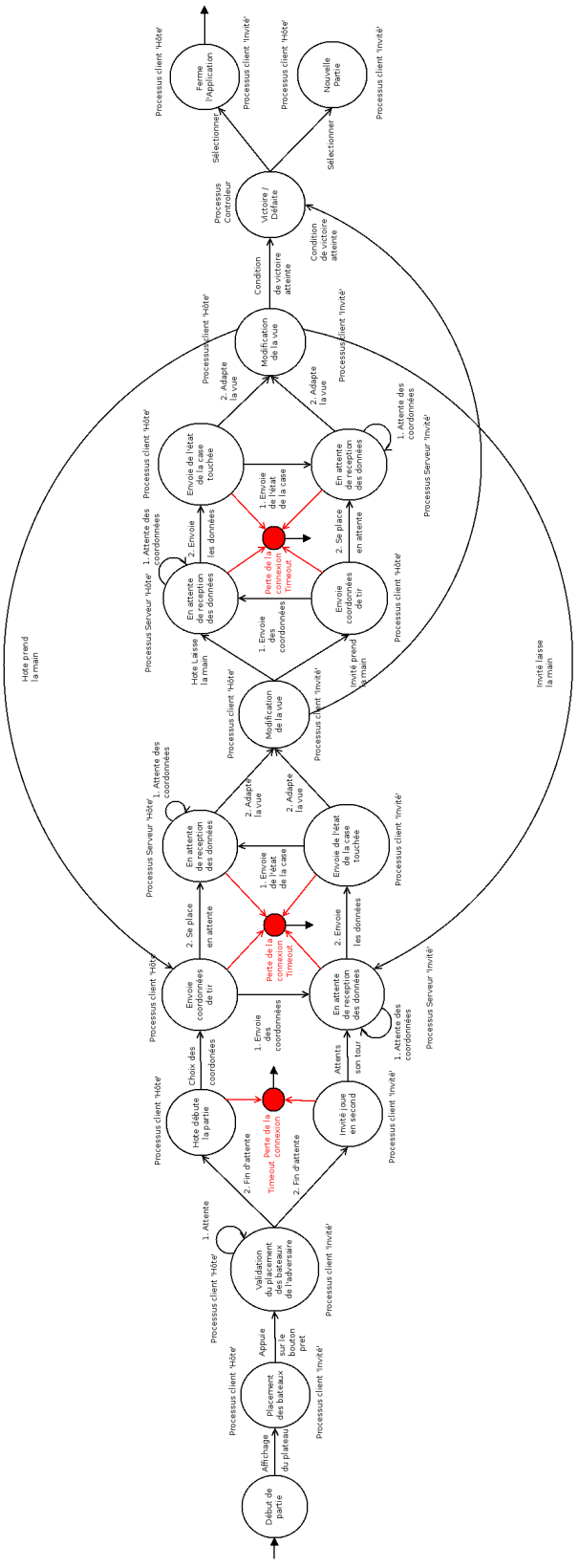

FIGURE 5 – Schéma représentant le protocole d'échange de données entre deux joueurs