Rapport de projet Bataille Navale

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

Responsable :
M. Yannick Guesnet

27 avril 2016

Table des matières

Remerciements

Nous tenons à remercier blablabla

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 adversaire physique 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.

1.2 Les intelligences artificielles

Il nous à été demandé de pouvoir jouer à la bataille navale en local. De ce fait, nous avons due développer une Intelligence Artificiel (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 expliquez le fonctionnement de chacune de ces 3 IA.

1.2.1 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'est pas déjà était tirée 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
        }
    } tant que (Board[Coord] != NON_CIBLÉ) ;
    Retourner Coord ;
Fin
```

1.2.2 IA Moyen

L'IA moyen, deuxième IA développée, fonctionne de manière similaire à l'IA facile, à ceci près qu'une fois un bateau touché l'IA passe en mode "chasse".
Le mode chasse indique à l'IA de tirer sur les quatres directions(nord, sud, est et ouest) entourant la case touchée et, si une nouvelle case est touchée, repart de cette dernière pour ainsi continuer jusqu'à avoir coulé le bateau.
Elle repart alors dans sa routine de ciblage.
L'algorithme est présenté ci-dessous :

```
Procedure MediumAdvisor :
    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.
    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) ;
            }
            Retourner Coord ;
        }
        Sinon {
            Si (Board[Coe[0]] == NON_CIBLÉ) {
                Coord = coe[0] ;
```

```
                Retirer(coe, 0) ;
                Retourner Coord ;
            }
            MediumAdvisor(Board, Dim, Coe);
        }
    Fin
```

1.2.3 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 bateau, puis passe en mode "chasse" une fois un bateau 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 bateau ennemi existant au début de partie, afin qu'elle ne tire que toute les tailles du bateau minimal. Par exemple, dans le cas ou le plus petit bateau est un batea de 2, l'IA va donc écarter ses tirs de 2 cases d'écart à chaque fois.

```
Procedure MediumAdvisor :
Entrée : Board un tableau de state contenant les resultats de tous les ti
         Dim le tableau des tailles des dimension de Board.
         Coe une liste de case.
         Minsize la taille du plus petit bateau present sur la grille
Sortie : Coord une Coordonnée d'une case non touchée.

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;
        }
    } tant que (mod != 0 || Board[Coord] != NOTAIMED) ;

    SI (Tir(enemy, coord) != MISSED) {
        AjouterCasesAdjacentes(coe, coord) ;
    }
    Retourner Coord ;
```

```
        }
        Sinon {
            Si (Board[Coe[0]] == NOTAIMED {
                Coord = coe[0] ;
                Retirer(coe, 0) ;Retourner Coord ;
            }
        }
        MediumAdvisor(Board, Dim, Coe) ;
    Fin Procedure
```

2 Le modèle réseau

2.1 Protocoles

2.1.1 Protocole Réseau

Le protocole réseau utilisé pour que deux joueurs puissent se connecter et jouer en réseau est le protocole TCP [1] 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 ([2] 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 imcomplet 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, 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 ainsi que comment se passe l'échange de données.

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,

1. Transmission Control Protocol : https ://tools.ietf.org/html/rfc793
2. User Datagram Protocol : https ://tools.ietf.org/html/rfc768

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 leur serveur respectif, 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 (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 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 effectuée.

2.1.2.2 Échanges des données

2.2 Gestion des erreurs

3 Interface Homme Machine

4 Autres

4.1 Idées d'améliorations

4.2 Problèmes rencontrés

4.3 Compétences acquises

Conclusion

Pour conclure, avec LaTeX on obtient un rendu impeccable mais il faut s'investir pour le prendre en main.

A Lexique

— 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), ...

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

— 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 datagramme qui doit être vérifier 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, ...).

— 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 a une autre machine et inversement afin que celles-ci puissent s'échanger des informations sans passer par un serveur distant.

B Webographie

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

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