Le Mans Université

Licence Informatique *2ème année*

Module 171UT01 Communication

Bataille navale : https://github.com/KillianDEUX/bataille-navale

DEUX Killian
GERARD Jules
DEROUET Corentin

19 avril 2019

Table des matières

1 Introduction

Dans le cadre de l'unité d'enseignement « Conduite de projets », nous avons du programmer un jeu en langage C à plusieurs.

L'objectif de ce projet est de mettre à contribution les enseignements appris depuis le début de cursus comme « Algorithmique ». Cela a permis de nous former au travail en groupe, une nouveauté pour nous.

Nous avons choisi comme le sujet la « bataille navale ». C'est un jeu de société qui, dans sa forme originale, se joue à deux joueurs dans lequel chacun d'entre eux doivent placer des bateaux sur une grille, inconnue de l'adversaire. La partie se termine quand l'un des joueurs trouve l'ensemble des bateaux de son adversaire avant que tous les siens ne soient découverts.

Chaque joueur dispose alors de deux grilles. La première servira à placer les bateaux et la seconde à placer des pions. Cette seconde grille sera une vue de la grille adverse.

Lors d'un tir à des coordonnées précise dans la grille adverse, l'ennemi nous indiquera si notre tir a atterri « dans l'eau », a « touché » ou a « coulé » un bateau. C'est en fonction de ces informations que l'on peut placer sur un pion de différente couleur. Nous avons cependant changé quelques règles que nous avons défini en début de projet.

2 Organisation

La planification du projet a été articulée autour de l'outil "Projects" de GitHub.
Celui-ci permet, via un système de "cartes", d'assigner des tâches à chacun d'entre nous.

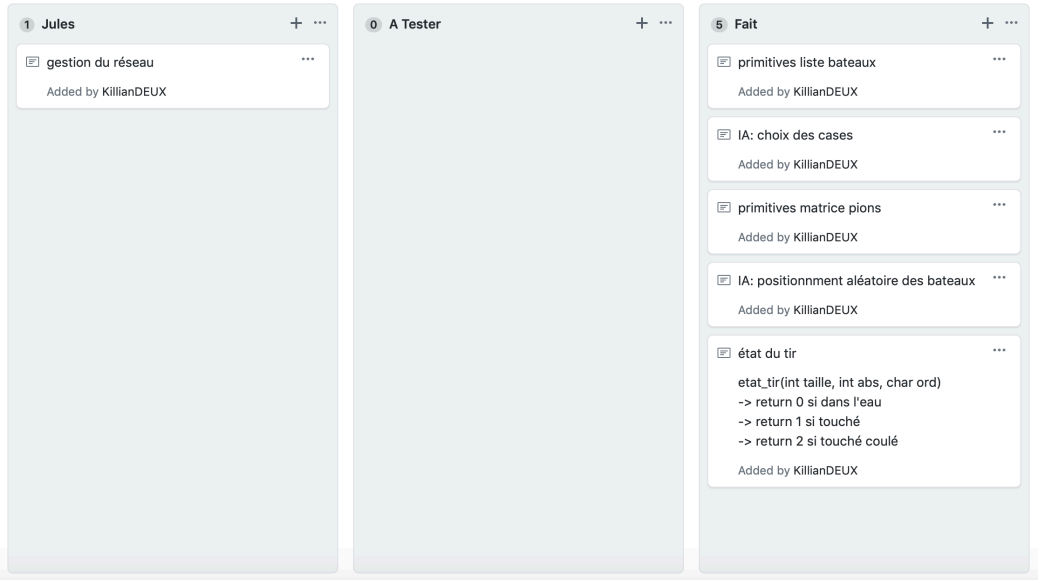


Figure 1 - Planificateur GitHub

De plus, l'outil "Code" permet de récupérer et de mettre à jour les changements effectués dans les différents fichiers.

La répartition globale est la suivante :

- Killian : gestion des bateaux et des opérations sur les matrices
- Corentin : gestion des matrices de pions et du mode jeu IA
- Jules : protocole de communication et gestion du réseau

3 Conception/Analyse

3.1 Explication des règles

Les règles de la bataille navale définies précédemment dans l'introduction sont fixes. Or nous voulions pouvoir adapter celles-ci aux exigences du des utilisateurs.
Nous avons donc choisi de permettre à cet utilisateur de pouvoir changer les règles basiques à volonté pour chaque partie de jeu qu'il souhaite effectuer. Certaines règles restent inchangées, deux bateaux ne peuvent ni se toucher ni se croiser ni être placé en diagonale.
Le joueur ne peut pas tirer deux fois au même endroit car la case contient déjà un pion et donc ne peut pas bouger ses bateaux de place au cours de la partie. Nos modifications sont :

- changement de la taille du plateau de joueur ,
- choix du nombre de bateaux en fonction de la taille du plateau choisi précédemment :
- le nombre de joueur, de 1 à 5,
- choix de la taille de chaque bateau, mais toujours définie entre 1 et 5.

Chaque bateau de taille différente possède un type particulier, respectivement :

- Mine,
- Torpilleur,
- Sous-Marin,
- Croiseur,
- Porte-avion.

Ci-dessous la courbe obtenue avec Geogebra en fonction de 3 points. Les X sont déterminés par la taille de la matrice et les Y par le nombre de bateau maximum voulu. Les 3 points sont donc 2 bateaux pour une matrice de 5x5, 5 bateaux pour une matrice de 10x10 et finalement 20 bateaux pour une matrice de 20x20.

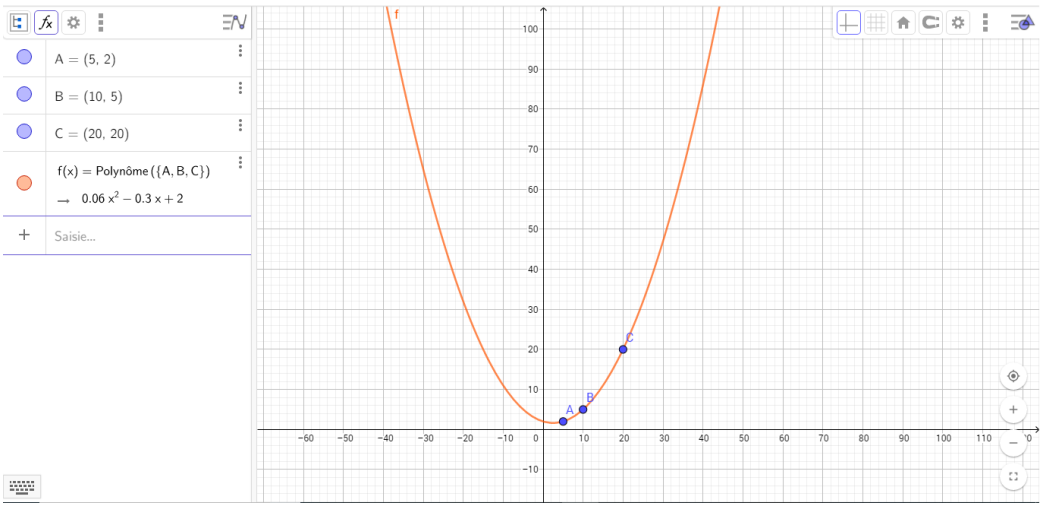


Figure 2 - Courbe Geogebra

3.2 Explication des fonctionnalités

En fonction des nouvelles règles ajoutées et des règles basiques de la bataille navale, nous avons donc dû adapter notre code.
Le choix des cases se fait par lettre et chiffre. Pour une question de practicité

les cases sont numérotées, en ligne et en colonne, de 1 à n et de 1 à m (n et m étant demandés au joueur).
Le joueur peut également choisir un plateau de jeu carré ou non.

Comme l'énonce les règles de l'introduction, chaque joueur possède 2 plateaux. L'un comprenant ses bateaux et ses cases attaquées.

- le ou les autres joueurs l'ont attaqués. et un plateau ou des plateaux adversaires qui montrent là où il a déjà attaqués et si la partie est à plus de 2 joueurs là où les autres joueurs ont également attaqués. Chaque case du plateau bateaux possède une cactéristique soit :

— une case est vide,
— une case touchée,
— un bateau normal,
— un bateau touché,
— un bateau coulé.

Et chaque case du plateau adversaire possède d'autres caractéristiques, soit :

— pas de pion,
— un pion blanc,
— un pion rouge.

4 Développement

4.1 Présentation de la conception d'un bateau

4.1.1 Présentation d'un bateau

Un bateau est constitué : -sa taille -son type -ses premières coordonnées (là où il débute) -sa direction (verticale ou horizontale) -son état (touché ou coulé) -nombre de fois qu'il a été touché par un joueur adverse

Toutes ses données sont nécessaires pour les traitements et sont donc utilisées tout au long du code et d'une partie.

4.1.2 Présentation des liste de bateaux

Le premier joueur définit une série de bateaux qui sera reprise par les autres participants. Nous avons donc besoin de stocker ces bateaux dans des listes (vues au premier semestre) pour traiter éfficacement ces éléments. Une liste possède un élément courant permettant de la parcourir ainsi qu'un "drapeau" signifiant la fin de celle-ci. Evidemment, nous avons dû adapter le système de liste pour qu'il soit applicable à nos bateaux.

4.1.3 Fonctions de modifications

Toutes les fonctions de modifications de listes ou de bateaux sont placées dans des fichiers séparés pour une réutilisation facile.

Lors de la mise en place des bateaux d'un joueur, une vérification est nécessaire. En effet, comme énoncé dans l'introduction deux bateaux ne peuvent pas se toucher même en diagonale. La fonction parcours_matrice est utilisée lors de ce processus. Elle parcourt la liste de bateaux déjà placés et récupère toutes les cases où le joueur ne pourra pas poser le bateau. Donc pour chaque bateau, la fonction vérifie s'il est placé à la verticale ou à l'horizontale et parcourt toutes les cases autour de lui pour les rentrer dans un tableau de cases non disponibles.
Une deuxième fonction appelée verif_placement_bateau, permet de vérifier que les coordonnées transmises, et donc souhaitées, n'interfèrent pas avec les coordonnées d'un autre bateau déjà placé. Donc pour chaque coordonnée voulue du bateau, la fonction parcourt entièrement le tableau de cases non disponibles.

4.2 Présentation des matrices

Nous avons eu besoin de créer deux types de matrices. La première matrice servira à placer nos propres bateaux, c'est la matrice "joueur". La seconde matrice servira à placer des pions en fonction de l'état du tir qui comprend trois choix possibles : touché, coulé ou dans l'eau. C'est la matrice "adverse".

4.2.1 La matrice "joueur"

La matrice "joueur" se compose d'un nombre de lignes et de colonnes choisi par l'utilisateur en début de partie. Le joueur peut choisir de donner une taille de grille minimale (5 lignes par 5 colonnes pour avoir un plateau jouable) et une taille maximale de 26x26. Les lignes étant modélisées par les lettres de l'alphabet.

Grâce à ces informations, le programme crée alors un tableau en conséquence. Chaque case sera initialisée avec des cases vides. Le premier joueur décide du nombre de bateaux et de leur taille pour tous les autres joueurs. Il place alors, en toute discrétion, ses propres bateaux en choisissant la direction et les coordonnées de chacun d'eux. Les autres joueurs ou l'intelligence artificielle placent ensuite leurs bateaux. Pour une meilleur compréhension par l'utilisateur, une légende est affichée. Celle-ci comprend la définition de chaque symbole en fonction de l'état de la case. Lors d'un tir d'un adversaire sur la flotte du joueur, les informations sur la case se mettent à jour. Les changements se font en même temps sur la grille adverse.

4.2.2 La matrice "adverse"

De plus, une autre matrice est créée. Il s'agit de la matrice adverse, elle possède la même taille que la matrice joueur. Cette matrice représente la vision partielle de la matrice adverse que possède ce joueur.

Elle contient donc antant de lignes que la matrice "joueur" et autant de colonnes que la matrice "joueur". Pour bien différencier les deux types de matrices, l'affichage est complètement différent. Tout comme précedemment, la légende aide l'utilisateur à comprendre les tirs qu'il a fait sur la ou les grille "adverses". Une fois un tir effectué sur la grille de l'adversaire choisi (si un choix est possible), la case en question se transforme pour indiquer au joueur l'état de son tir. De plus avant chaque affichage de la matrice, le programe affiche par une phrase si le tir est efficace ou nul.

4.3 Présentation des pions

La matrice "adverse" possède plusieurs caracteristiques. Les principales fonctionalitées de cette matrice sont d'indiquer l'état du tir et placer un pion, symbolisé par un caractère défini dans la légende en fonction de la case choisis.

Une règle nous interdit de placer des bateaux les uns à côtés des autres. Pour faciliter l'expérience utilisateur, lorsqu'un bateau est touché, une fonction transforme toutes les cases autour comme 'dans l'eau'.

Une fois le bateau coulé, la fonction repère ce bateau, puis va sélectionner une par une chaque case présente autour et va remplacer le contenu de cette case par un pion "dans l'eau". Elle vérifiera cependant que la case est bien dans la grille car si un bateau est le long d'un bord, certaines cases autour se trouvent hors de la zone de jeu.

4.4 Présentation de l'IA

En temps normal, on ne peut pas jouer seul à la bataille navale. Or une des fonctionnalités de notre code permet de jouer à la bataille navale contre une IA. Cette IA se met en place lorsque le joueur indique qu'il veut jouer tout seul.

4.4.1 Placement des bateaux

L'IA place ses bateaux aléatoirement dans le plateau tout en vérifiant qu'aucun ne sortent de la grille ou ne se touchent (en utilisant la même fonction que pour les joueurs).

4.4.2 Choix des cases de tir

Puis vient les attaques que doit effectués l'IA. Lors de son premier tour, l'IA choisit une case du plateau aléatoirement. Puis, lors de son deuxième tour et pour tous les tirs suivants, elle commence par vérifier si un bateau a déjà été touché et non coulé.
Si c'est le cas elle effectue un tir sur toutes les cases où elle n'a pas encore tiré et qui entoure les cases du bateau déjà touchées. Elle commence par les cases situées au nord de ce bateau puis par l'est ensuite par le sud et pour finir l'ouest.

Si aucun bateau n'a été touché, l'IA choisit une case la plus éloignée possible de toutes les cases qui ont déjà été sélectionnée.
Si aucune case n'a été trouvée, elle essaye de trouver une case pseudo aléatoirement, c'est-à-dire qu'elle trouve une case aléatoirement mais recommence tant qu'elle reste collée à une autre case déjà touchée.
Si elle n'en trouve toujours pas, cette fois c'est une case totalement aléatoire qui est trouvée.

4.5 Présentation du réseau

Dans ce mode de jeu, on dissocie deux exécutables : le serveur et les différents clients. Le serveur, en plus de s'occuper des communications, présente une vue d'ensemble de la partie en cours. Le client propose l'interface de jeu.

4.5.1 Serveur

Pour permettre léchange de données entre les différents utilisateurs, nous avons choisi dimplanter le réseau dans ce projet grâce au protocole TCP. Celui-ci présente un avantage important : le contrôle de linformation reçue.

Comme le jeu doit supporter un nombre variable de joueur, le serveur est en mesure douvrir plusieurs sockets clients.
Pour cela, il a été nécessaire de faire un tableau pour établir les canaux de communication de chaque client. Ainsi, le canal dindice 0, correspond au 1er client qui sest connecté et ainsi de suite.

No Joueur	1	2	3	4	5
Socket	client_fd[0]	client_fd[1]	client_fd[2]	client_fd[3]	client_fd[4]

Figure 3 - Equivalence joueur et indice client

Ces canaux vont permettre denvoyer des variables par le réseau via les fonctions send et recv de la bibliothèque « sys/socket.h ».
Ce mode de fonctionnement est adapté au jeu de la bataille navale tour par tour car la fonction recv est bloquante.
La main est rendu à lutilisateur dès que linformation est reçue, dans le cas contraire, celui-ci reste en attente.

Du coté client, un unique socket est établi vers le serveur. Les clients ne peuvent donc pas communiquer directement entre eux. Cette contrainte implique de nombreux envois...

4.5.2 Clients

De plus, il est nécessaire de fournir l'information au client pour trouver le serveur. Notre choix s'est porté sur l'adresse IP pour plus de simplicité au lieu du nom d'hôte. Cette dernière est demandée à l'utilisateur avant d'initier la connexion. Le port est quant à lui défini en constante.

Pour plus de cohérence et d'équité, seul le premier joueur (le premier client connecté au serveur) choisi la taille de la grille et la liste de bateaux. Ces choix sont mémorisés et redistribués aux autres clients. Il leur reste la liberté de placer leurs navires.

Le mode de jeu réseau présente une spécificité : la grille de pions des joueurs ne varie pas en fonction du joueur attaquant mais en fonction du joueur attaqué.
Ce choix, qui diffère de la règle originale, permet plus de stratégie lors des parties à plus de 2 joueurs.
D'autre part, toujours dans le cas d'une manche à plus de 2 joueurs, le jeu s'achève lorsqu'un joueur a perdu l'intégralité de sa flotte.

4.6 Programmation modulaire

4.6.1 Fichiers utilisés

Pour pouvoir utiliser toutes nos fonctions de traitement comme par exemple le traitement des listes ou des bateaux, les fichiers générant l'IA ou encore les fichiers gestionnant les matrices de pions, un makefile est nécessaire. Le makefile demande la création de fichier .h contenant la déclaration des structures, par exemple de quoi est composé un bateau, ainsi que les primitives de chaque fonction. Il permet de compiler, c'est-à-dire de transformer les fichiers, codés en language C, en fichiers contenant le language machine approprié. Ces fichiers transformer finissants par .o sont ensuite utilisés par le serveur et le client.

4.6.2 Fichiers généraux

Le serveur et le client sont les deux fichiers utilisés par l'utilisateur lors de l'exécution. Pour jouer en solitaire et donc contre l'IA l'exécution du serveur suffit. Sinon il faut exécuter le serveur ainsi que, pour chacun des joueurs, autant de client que de nombre de joueur, chaque joueur ouvrant le sien. Le serveur et le client sont donc compilés en utilisant tous les fichiers .o nécessaires et générés précédemment.

5 Résultat

A l'heure actuelle, le programme est fonctionnel et propose de manière indépendante les deux modes de jeu que nous souhaitions : solo contre l'IA et multijoueur en réseau. Les règles personnalisées sont toutes présentes ce qui apporte un peu de nouveautés au jeu classique. Néanmoins, certaines fonctionnalités n'ont pas été implémantées.

Le plus gros point noir est l'absence de l'interface graphique intialement prévue. Cette dernière aurait permis une bien meilleure expérience utilisateur. Par exemple, il aurait été plus naturel et ergonomique de devoir cliquer sur une case au lieu d'entrer des coordonnées au sein d'une console.

Nous aurions également voulu ajouter un "timer" afin d'imposer une limite de temps sur un tour. Ainsi, l'attaquant qui dépasserait le temps imparti pour choisir sa cible serait obliger de passer son tour. Cette limite éviterait de trop longues attentes pour les autres participants.

Une autre option qui avait été evisagée était de pouvoir déplacer un bateau au lieu d'attaquer un joueur. Cette option aurait rajouter un aspect stratégique intéressants. L'idée était de pouvoir déplacer un bateau au cours de la partie eu lieu d'attaquer. Chaque joueur pourrait déplacer ses bateaux à condition qu'il ne soit pas déjà touché.

De plus, il serait intéressant que le mode de jeu IA puisse prendre la relève en cas d'un joueur réseau inactif ou encore en cas de perte de connexion.

Malgré quelques difficultés à la réalisation finale de notre code, nous avions l'idée, qui n'a malheureusement pas pu aboutir, de faire une bataille navale mais en 3D. C'est à dire avec plusieurs types de sous-marins ainsi que des avions. Donc le but était de faire 3 plateaux de jeu pour chaque joueur.

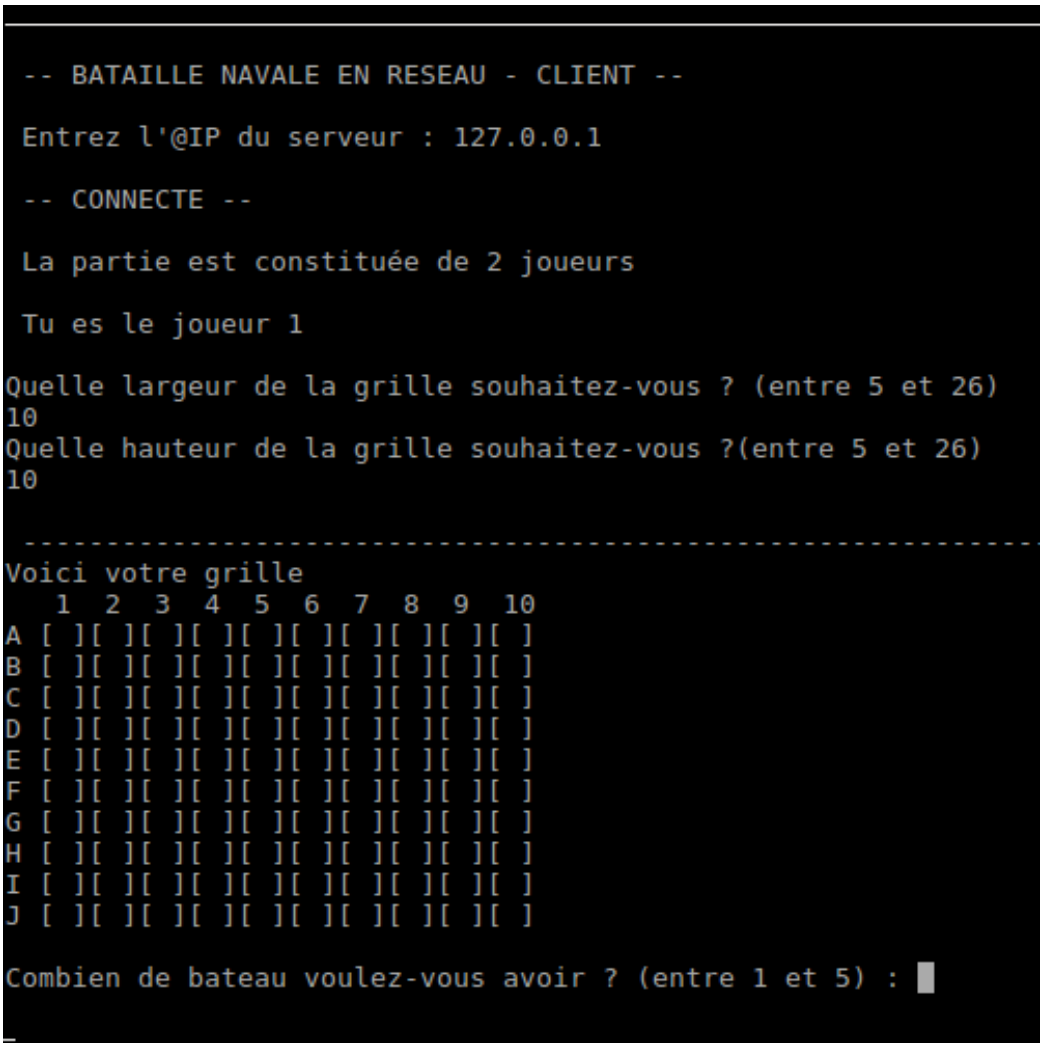
Enfin, il aurait été souhaitable, de mettre en place des pseudos et des highscores.

6 Conclusion

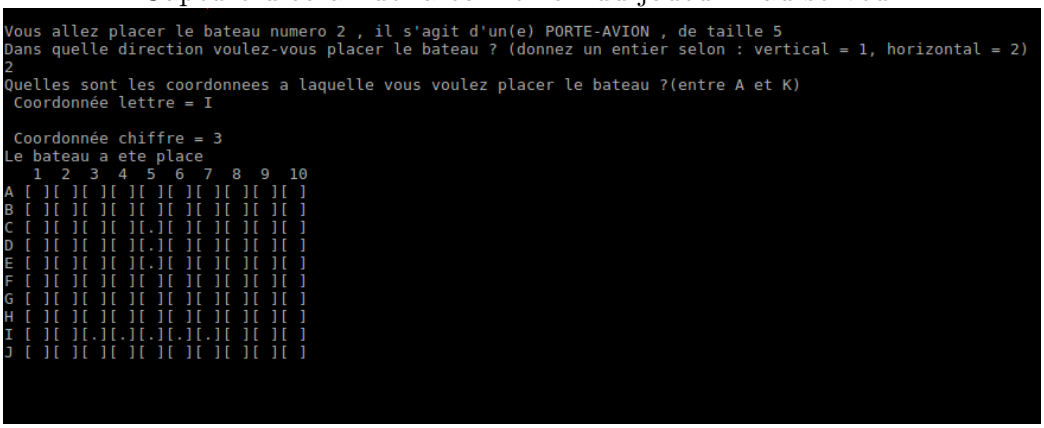
Ce projet assez conséquent nous a permis de découvrir comment travailler à plusieurs. En effet, nous avons dû nous organiser et prévoir une répartition des tâches. Les contraintes de temps ont été les plus difficiles. À cause de cette contrainte, nous n'avons pas réussis à ajouter toutes les fonctionnalitées voulues, en particulier la SDL. Nous avons pourtant réussi à avoir un programme jouable sur terminal, que ce soit contre une IA ou en réseau, avec les règles prévues dans le cahier des charges. Même si, au niveau du réseau, l'optimisation n'est pas complète, il est quand même possible de jouer à la bataille navale jusqu'à 5 joueurs. Nous avons perçu les difficultés de travailler en groupe.

Chaque personne travaillait sur une partie du code et lors de la mise en commun, même si nous nous étions mis d'accord sur des fonctionnalités, certaines fonctions n'était pas compatibles entre elles. A force de debogage, nous avons tout de même pu finir par avoir un jeu auquel il est possible de jouer. S'il fallait continuer ce projet, on pourrait alors améliorer certains points important comme l'interface et l'optimisation du réseau. Une fonctionnalitée intéressante aurait été de ne pas arrêter la boucle de jeu lorsqu'un joueur perd.

7 Annexes

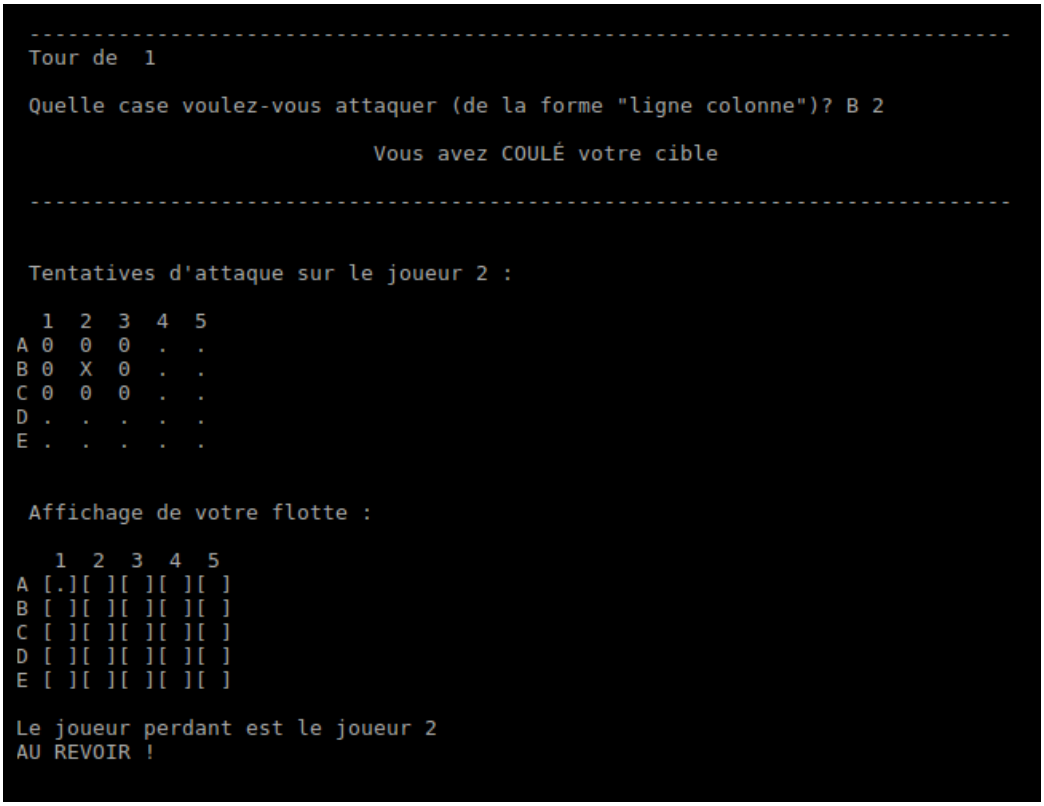


Capture d'écran de la connexion du joueur 1 au serveur



Capture d'écran du placement de 3 bateaux

Capture d'écran de l'attaque d'un bateau



Capture d'écran de la fin de partie

Références

[1] http ://benhur.teluq.uquebec.ca/ mcouture/apa/index.htm

[2] http ://benhur.teluq.uquebec.ca/ mcouture/apa/docsweb.htm