Université de Montpellier

Projet Bataille Navale

HLIN302

Rapport de projet

Auteurs:
Rémi Mattei
Mohamed Bouchengour
Rémi Lagardère
Guillaume Kostrzewski

Encadrants:
Pascal Giorgi
Alban Mancheron

Sommaire

1 Introduction

Ce rapport contient principalement des explications et commentaires sur les méthodes utilisées pour réaliser diverses étapes du projet, ainsi que les quelques problèmes rencontrés lors de sa réalisation.

Nous avons commencé la programmation du jeu en octobre avec comme outil de travail WSL sur Windows 10 et Github pour une collaboration à distance.

2 Comportement de l'IA

Une manière simple mais efficace pour un bot de sélectionner une case complètement aléatoire est d'utiliser la fonction rand() sur deux variables x et y, représentants la position d'une case sur la grille du jeu.

Dans le cas où la case aurait déjà été "découverte" (c-a-d visée par un tir de missile), l'opération de recherche serait répétée jusqu'à obtenir un résultat satisfaisant. Cette tâche n'a posée aucun problème à être implémentée.

Il va de soi qu'il est indispensable pour un joueur virtuel d'utiliser une méthode alternative dans le cas où un navire aurait déjà été partiellement découvert sur la grille ennemie ciblée, à moins qu'une IA abrutie soit souhaitée.

Nous avons donc pensé à un système assez similaire où la différence majeure se situe au niveau de l'intervalle sur lequel x et y sont générés. Nous avons restreint la zone dans laquelle chercher des cases aléatoires autour de la dernière position dite "gagnante", c'est à dire là où un navire a été touché mais pas coulé.

Pour le mettre en place, il nous a notamment fallu créer et utiliser une fonction (aDesVoisins) qui vérifie si une case, passée en paramètre, possède autour d'elle (dans un rayon de 1) au moins 1 voisin non touché par un tir de missile parmi ses 8 (ou moins si elle est collée à un bord).

Dans le cas où cette dernière (avec comme paramètre une case possédant un navire touché) renvoie TRUE, alors tous les prochains missiles envoyés par des joueurs virtuels sur cette même grille seront concentrés autour de cette case, qui est gardée en mémoire dans la classe de la grille du joueur ciblé (via les entiers focusx et focusy), afin de ne pas re-lancer de recherches inutiles à chaque nouveau tour. Un navire touché après ça devient à son tour la nouvelle zone à abattre.

Un navire coulé ré-active la méthode de recherche entièrement aléatoire, juste après l'exécution d'un algorithme (findFocus) vérifiant que la grille ne possède pas de navires détectés. Dans le cas contraire, la case en question est sauvegardée (toujours via les entiers focusx et focusy) et les bots continueront à attaquer autour d'elle. Sinon, alors focusx et focusy prennent la valeur -1, qui est la condition nécessaire à l'exécution de la fonction TirDeMissileAleatoire.

Dans tous les cas, les intervalles sur lesquels générer x et y (la position de la case qui sera ciblée par le bot) prennent évidemment en compte les bords de la grille ainsi que la difficulté.

3 Difficulté

Nos trois niveaux de difficultés implémentés influent sur deux choses :

la sélection des joueurs adverses comme nouvelle cible

le rayon, autour d'un navire touché, dans lequel le bot doit sélectionner une case.

Il existe trois critères pris en compte lors de la sélection d'une nouvelle cible par l'IA :

- aléatoire
- vengeance : prend pour cible le dernier joueur ayant touché un de ses navires
- vulnérabilité : prend pour cible un joueur possédant sur sa grille des navires localisés mais pas encore coulés. (Dans le cas où plusieurs joueurs sont concernés, l'un d'entre eux est choisi aléatoirement)

Tous les critères ne sont pas pris en compte selon le mode de difficulté :

- Facile: 1 chance sur 3 aléatoire, vengeance, vulnérabilité.
- Moyen : 1 chance sur 2 vulnérabilité, vengeance.
- Difficile : vulnérabilité

Le rayon depuis la case d'un navire touché est le suivant :

- 1 si Moyen ou Difficile (soit un total de 8 cases concernées maximum)
- 2 si Facile (soit 25 cases maximum)

4 Scores

Nous avons d'abord réfléchi au système de score proposé dans le sujet du projet, avant d'abandonner rapidement cette idée : compter simplement le nombre de missiles tirés par le gagnant serait plus ou moins inéquitable selon les options du jeu : nombre de joueurs, nombre d'humains, taille de la grille, taille des navires...

Et bien qu'il n'était pas précisé dans le cahier des charges d'implémenter une méthode de calcul de scores plus poussée, nous avons décidé de nous pencher dessus malgré tout.

Nous avons donc longuement réfléchis à une formule, pas forcément trop compliquée mais efficace dans tous les cas, qui prendrait en compte toutes les variables citées plus haut.

Nous nous sommes d'abord penchés sur un système de points récompensés en fonctions des tirs gagnants, avec éventuellement des points bonus accordés aux joueurs qui toucheraient consécutivement (lors du même tour) un navire, ou qui éliminerait un adversaire (une mort qui surviendrait à la suite de la destruction des cinq navires du joueur). Mais là également, il serait facile de profiter de ce système en paramétrant de grands navires sur une grille de petite taille.

Nous avons ensuite pensé à récolter les scores en fonction de la précision des joueurs : en effet, celle-ci pourrait se calculer simplement par le rapport entre le nombre de tirs gagants du joueur sur le nombre de tirs totaux.

Mais même dans ce cas, le problème de la taille des navires et de la grille était toujours présent. Et parce que le score d'un jeu vidéo est rarement (voire jamais) une précision, nous sommes restés dubitatifs quant au fait de l'afficher sous cette forme.

Mais nous avons vite remarqué que la précision serait dans tous les cas un élément primordial pour leurs calculs.

Diviser la précision du joueur par le rapport entre le cumul de la taille des navires et la taille de la grille (WxH) ne pourrait-il pas fonctionner ?

Ainsi, une partie paramétrée avec des navires minuscules de taille 1x1 sur une grille de 20x20 (la taille maximum) permettrait au joueur le plus chanceux d'obtenir un score maximum de 80 en éliminant son adversaire en cinq coups gagnants, soit donc une précision de 100

Afin d'obtenir un résultat maximum de 100, nous pensions le multiplier par 1.25.

Un joueur qui commencera la même partie sur une grille de 10x10 ne pourra obtenir un score supérieur à 25, malgré le même niveau de précision : une grille quatre fois plus petite étant en cause.

Nous décidons finalement de garder cette formule avec une touche finale permettant des scores allant jusqu'à 1 000 000 : changer le multiplicateur 1.25 en 12 500.

Le seul moyen d'obtenir ce score de un million est d'utiliser la configuration donnée plus haut (5 navires de taille 1x1 sur une grille de 20x20) et d'avoir une chance sur... 9 986 232 009 600.

Il est ainsi 71 412 fois plus probable de gagner à l'Euro Million (1 chance sur 139 838 160 pour 5 numéros + 2 étoiles)

5 Modélisation de la flotte

Il nous semblait impossible de modéliser les navires sans faire usage des tableaux. Notre toute première version ne prenait en compte que les 5 navires de base présents dans le sujet du projet, et nous avons vite réalisé que l'achèvement de l'étape deux nécessiterait manifestement une refonte complète de notre modélisation actuelle, en quelque chose de bien plus élaboré et capable de lire des navires "formés" par l'utilisateur dans un fichier texte.

Cette étape fut l'une des toutes premières réalisées, en même temps que la création des grilles.

Ainsi fut créé le fichier de configuration principal (config), dans lequel nous retrouvons en première ligne la taille de la grille (LxH) suivi des navires :

La taille de la grille est simplement obtenue en lisant la première ligne du fichier config, après avoir ouvert ce dernier via ifconfig : Pour la largeur, depuis le premier caractère, censé être un chiffre (vérifié via l'utilisation de isdigit(ligne[i]), jusqu'à la rencontre d'un caractère non numérique. Pour la hauteur, depuis le première caractère non numérique (on suppose un x ou * entre les deux dimensions) jusqu'à la fin de la chaine de caractère numérique.

Dans les deux cas, la taille est un entier obtenu après la conversion (itoa) du caractère numérique en cours de lecture.

Les navires sont eux représentés dès la ligne deux et chacun est espacé d'un retour à la ligne. Un x représente une portion du navire, tandis qu'un espace sera "vide".

Le fichier doit absoluement posséder 5 navires, chacun espacé par un seul retour à la ligne, afin d'être lu correctement par l'algorithme.

Celui-ci va d'abord calculer la largeur et la hauteur de chaque navire, avec une limite imposée de 5x5, de manière similaire à celle employée pour la lecture des dimensions de la grille.

La fonction créée retournera un tableau d'entiers à trois dimensions qui contiendra tous les navires. navire[n][x][y] où n est le numéro du navire (0-4) et x et y correspondent aux axes d'un tableau bidimensionnel où chacune des cases possède un 1 si une partie du bateau est présente dessus, et un 0 sinon. Le tableau de chacun des navires, quelle que soit leurs formes, est carrée (la raison sera expliquée plus loin) de taille max(hauteur,largeur).

Nous avons délibérément ajouté une ligne et une colonne supplémentaire pour chacun des tableaux, dont toutes les cases comprises à l'intérieur contiennent -1, afin de délimiter leurs tailles et ainsi éviter des erreurs de segmentation dans le futur. Dans le cas où une fonction lirait le tableau d'un navire donné sans en connaître les dimensions, l'accès à une case de valeur -1 pourrait être une condition d'arrêt marquant la dimension de ce navire.

C'est à la suite de la réalisation de cette étape importante que nous avons souhaité permettre la modification des navires et des dimensions de la grille directement dans le menu du jeu.

Bien que le cahier des charges ne le demandait pas, nous avons malgré tout décidé de nous lancer dans la mise en place d'un menu Options permettant ces fonctionnalités, à un stade où seul un quart du travail demandé était complété.

La modification de la taille de la grille n'a pas posé de soucis. L'éditeur de navires, une fonctionnalité que nous souhaitions vraiment implémenter au jeu, a été plus longue (surtout en terme débuggage).

6 Sauvegarde / Chargement

Lors de la sauvegarde, ce sont trois fichiers qui sont créés dans le dossier "save" pour un futur chargement :

1. config.txt, qui contient les navires et la taille de la grille
2. le fichier log dont le nom peut être édité avant le lancement d'une partie (par défaut, "fichier.log")
3. sauvegarde

Ce dernier contient toutes les informations nécessaires à la reprises d'une partie entamée : joueurs, humains, tours, noms, historique, difficulté, états des cases de toutes les grilles, positions des navires, nombre de navires restants etc..

Le chargement de la dernière partie sauvegardée depuis le menu propose à l'utilisateur si il souhaite supprimer tous les fichiers de sauvegarde après leurs chargements.

La partie reprend ensuite à l'endroit exact où celle-ci a été enregistrée.

7 ncurses

Pour coder ce jeu, il nous a évidemment été indispensable d'apprendre le fonctionnement de la librairie et de nombreuses fonctions qu'elle propose, au-delà de la simple utilisation des fonctions "print" et création de fenêtres.

Mais bien que ncurses soit certainement un bon moyen d'apprendre à gérer une interface graphique (qui était notre première), nous avons régulièrement été freiné par les limitations imposées par cette librairie.

Par exemple, la disparition d'une fenêtre laisse des traces bien trop visibles, ce qui nécessite un rafraichissement de la ou des fenêtres dérrière celle-ci, ainsi que de tous leurs contenu (texte).

Nous nous sommes également rendu compte qu'il fallait être très prudent lors de la création d'un programme executable dans un terminal : certains terminaux ne supportent pas toutes les fonctionnalités proposées par ncurses.

Par exemple, WSL (l'outil que nous avons principalement utilisé pour compiler / executer) ne supporte pas certains effets (tels que A BLINK, A INVIS) et caractères spéciaux (notemment les flèches ACS ARROW) tandis que les terminaux d'Ubuntu 16 les affiche correctement.

Nous avons fait en sorte de ne pas utiliser ces effets pour assurer une compatibilité sur un maximul de terminaux.

Nous avons d'ailleurs programmé et rajouté (comme suggéré dans le sujet) plusieurs fonctions dans la classe Window permettant une meilleur personnalisation des fenêtres et des bordures. Parmi elles, une numérote la bordure de toutes les grilles des joueurs de 1 à H (hauteur) verticalement et de A à W (largeur) horizontalement comme le veut le vrai jeu de Bataille Navale.

Le déplacement à l'intérieur d'une grille met même en surbrillance les numéros de la case visée directement dans la bordure de la fenêtre.

La bordure inférieure de chaque grille contient également le nombre de navires restants sur celle-ci, le nom du joueur ainsi que qu'un caractère spécial (BULLET, proposé par la librairie ncurses) autour du pseudo correspondant au joueur en train de jouer.

Nous avons également rajouté des couples de couleurs et modifié la façon dont le constructeur des fenêtres gère leurs positions. Désormais, chaque fenêtre créée a pour base le milieu du terminal (ncurses permet d'obtenir très facilement sa taille avec LINES et COLS).

Nous avons fait ça afin de centrer au maximum tous les éléments du jeu. Qu'importe le nombre de joueurs : leurs grilles, flottes, le menu et la bordure seront toujours au centre du terminal.

8 Structure

Nous n'avons pas de classe joueur mais une classe grille qui contient à la fois toutes les données realtives aux cases et au joueur, sa flotte (classe) et son score (classe), ce qui n'a posé aucun problème dans le developpement du jeu, bien que d'autres structures auraient pu être utilisées.

L'ajout progressif de fonctions au fil des semaines/mois nous a fait prendre conscience à la fin du semestre que certaines parties du jeu (bien que parfaitement fonctionnelles) auraient pu être codées différemment, peut-être plus simplement, mais nos connaissances à cette époque du C++ et de la librairie ncurses n'étaient pas les mêmes.

Une modification des parties concernées a été réléchie, mais les 6 000 lignes de code déjà présentes auraient nécessitées beaucoup de travail supplémentaire et nous avons estimés avoir déjà consacré énormément de temps sur ce projet, au détriment des autres modules.

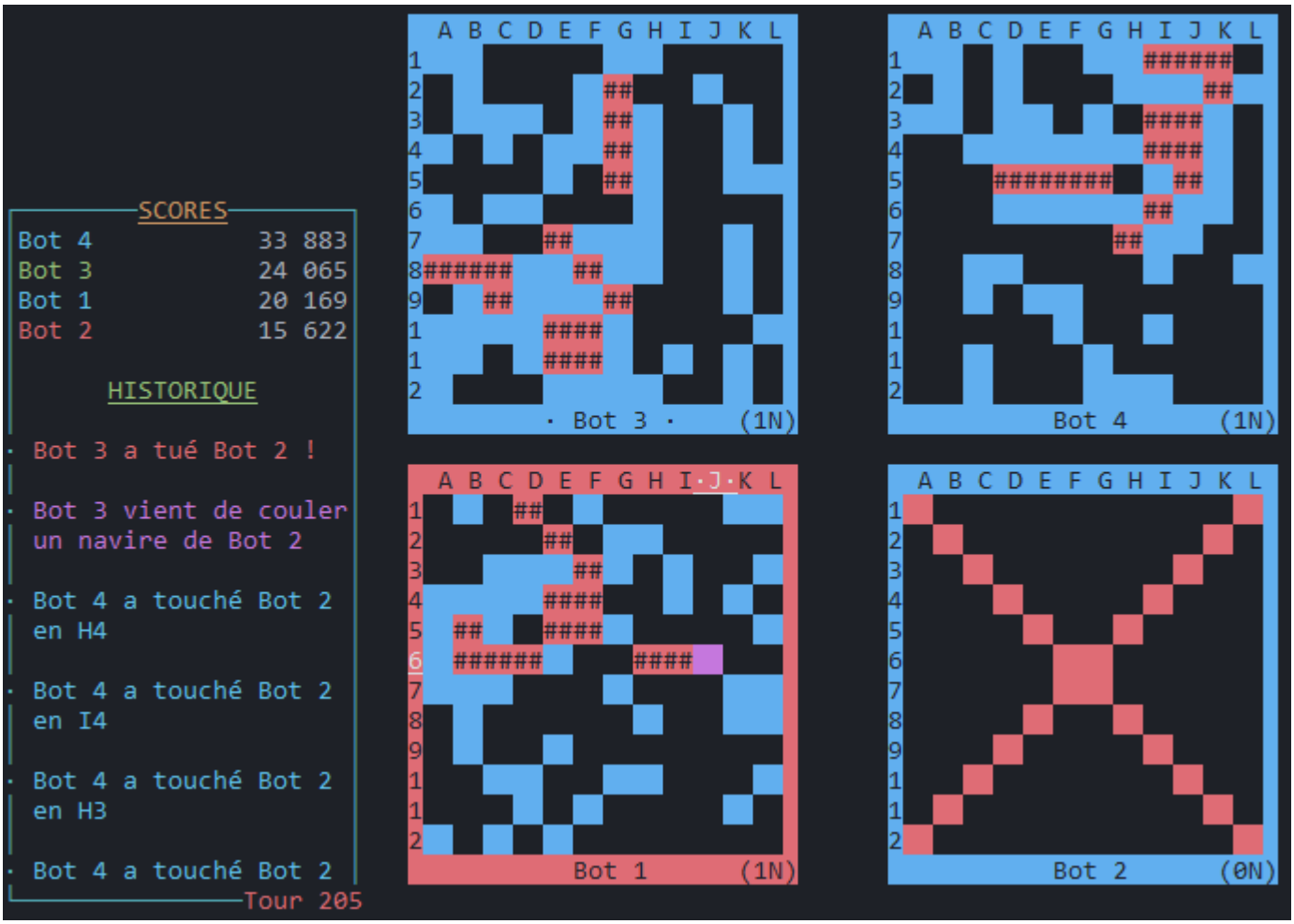

Figure 1: Capture d'écran d'une partie de quatre joueurs virtuels, dont un mort

9 Conclusion

Le cahier des charges a été respecté à l'exception de la fonctionnalité permettant le retour en arrière. Nous comptions utiliser un système de pile, ce qui nous paraissait le plus efficace.

Nous avons rajouté de nombreuses fonctionnalités pour une meilleur expérience de jeu, parmi lesquelles l'éditeur de navire, les thèmes, les flottes pré-définies, un historique des actions en jeu, un affichage des scores dont l'ordre est constamment actualisé et plusieurs options disponibles avant le lancement d'une nouvelle partie (difficulté, nom du fichier log...).

Nous avons permis une visualisation en temps réel des mouvements de l'IA lors de la sélection des cibles et des cases, ainsi que la possibilité de modifier leurs vitesses à tout moment, même lors d'une partie composée exclusivement de joueurs virtuels.

Nous nous sommes assurés que chaque étape du jeu affiche de manière claire les instructions à l'écran (étapes, à qui le tour, scores, avertissements...).

De même pour les fonctionnalités proposées dans le menu des options : des fenêtres sont présentes pour aider l'utilisateur, toujours dans le but d'afficher un maximum d'informations pertinentes.

Le débugage du jeu a été très long et nous avons dû régulièrement jouer afin de repérer les quelques bugs après l'ajout de nouvelles fonctions. Souvent, nous paramétrions une partie composée de joueurs virtuels et les laissoins jouer seul, tout en les surveillant, pour qu'ils fassent ainsi une partie du travail à notre place.

Ce jeu a été notre "vrai" premier projet de programmation depuis notre première année et nous a beaucoup appris, en paralèlle des cours et TPs de ce semestre.