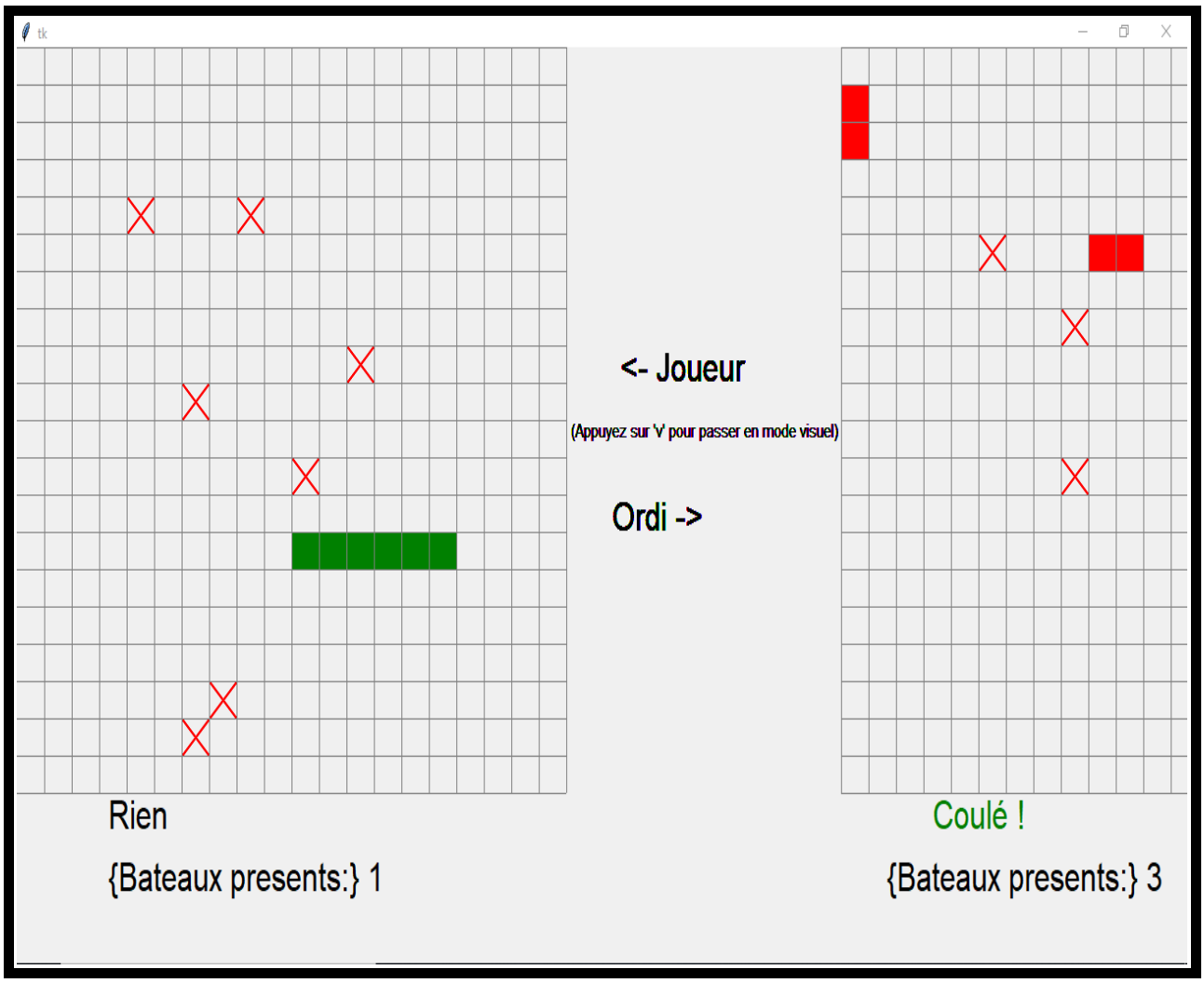
Bataille Navale



Paul Lesager : ***Licence Maths/Info – Première année – UPEM - TD A – TP 2***

Diagne Ben : ***Licence Maths/Info – Première année – UPEM - TD B – TP 2***

Sommaire

Présentation

La bataille navale est un jeu de société qui prend place dans un plateau de jeu qui est un quadrillage. Le quadrillage est constitué de rectangles qui représentent les bateaux. Les joueurs doivent d'abord placer leurs bateaux, puis ensuite tenter de faire couler les bateaux adverses grâce à des tirs. Le jeu se termine quand un des deux joueurs n'a plus de bateaux en jeu.

Manuel

Naviguer dans le menu, déplacer un bateau = Touches directionnelles/les boutons ZQSD
Sélectionner un mode, placer un bateau = Entrée
Tourner un bateau vers la gauche = A
Tourner un bateau vers la droite = E
Tirer = Clic gauche
Mode Visuel (voir les bateaux cachés) = V

Sur quelles idées allons nous nous baser ?

Pour écrire notre programme, on s'est principalement basé sur upemtk, un module qui nous a été disposé pour la base du jeu. On s'est aussi aidé du site de documentation d'upemtk pour mieux comprendre le module et l'utiliser à son plein gré. Enfin, notre programme s'est aussi basé sur plusieurs TP et mini-projets, comme le TP 6 pour établir la fonction créeant le quadrillage et pour gérer certaines autres fonctions du programme.

Découpage du travail

Le plus souvent, soit on se répartissait les tâches, communiquait nos résultats et finalisait chacune de nos tâches, soit on travaillait en parallèle sur une partie, comme la gestion des clics. Sinon, Ben s'était focalisé sur la gestion de la bataille et la présentation du jeu (annonces, résultat, instructions) tandis que Paul s'est focalisé sur la gestion du menu, du placement des bateaux et de la bataille en général.

Moyens de communication

Nous avons principalement travaillé lors des heures de TP, mais aussi en particulier en dehors des cours . Pour cela, nous avons communiqué à l'aide de Discord et aussi par mail pour communiquer nos résultats, envoyer nos tâches et organiser le projet en général. La plupart du temps, on se répartissait les tâches après la fin des cours de TP.

Le jeu

Le jeu est découpé en trois parties : Le menu, le placement des bateaux et la bataille. Le programme est donc découpé en trois boucles principales qui gèrent respectivement les trois parties du jeu. Ces boucles sont précédés de fonctions qui sont essentiels pour faire fonctionner le programme car elles sont aussi utlilisés dans les boucles.

I / Le menu

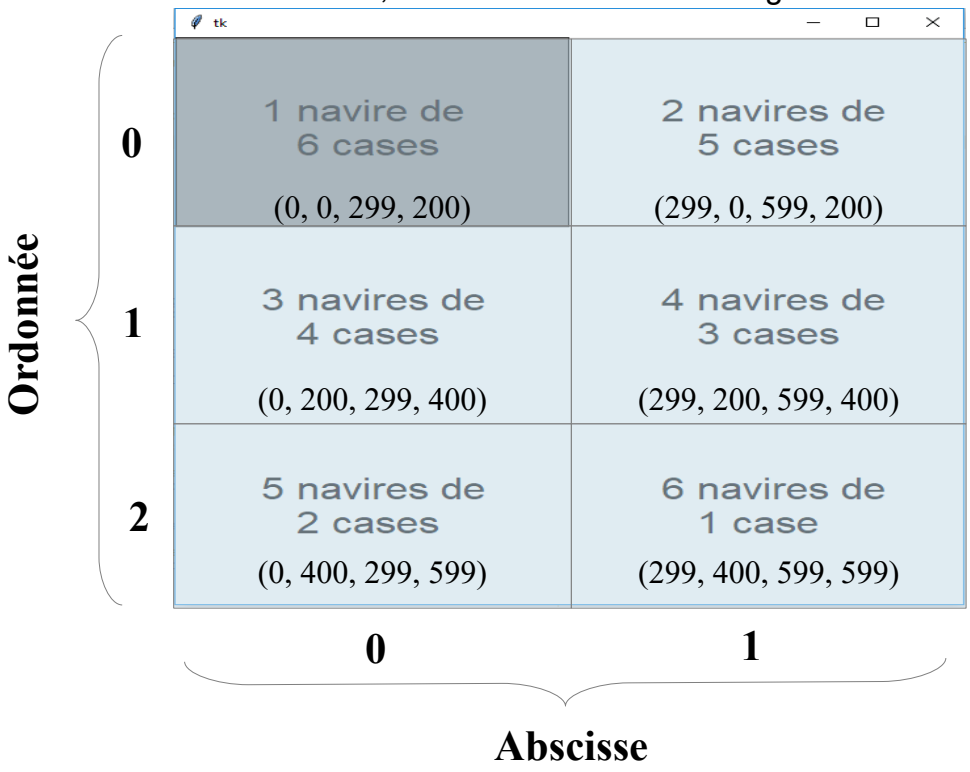
Le menu est une fenêtre qui regroupe les choix qui s'offrent aux joueurs. Ces choix permettent au joueur de choisir son type de composition de bateaux: 1 bateau à 6 cases, 2 bateaux à 5 cases, etc. Pour cela, on a deux fonctions :

- Affiche_cases_choix, qui affiche le figuré du curseur du menu (qui est un rectangle).
- Position_choix, qui gère les déplacements de ce curseur à l'aide d'évènements.

Affiche_cases_choix (position) :

→ On initialise d'abord les coordonnées de position du curseur

→ On établit une condition avec les coordonnées de position pour modifier les coordonées du rectangle dont ses valeurs dépendront de la coordonnée de position : Par exemple: si abscisse = 0 et ordonnee = 2, le curseur serait en bas à gauche du menu :



Position_choix (position, touche) :

→ On initialise d'abord la variable « position », qui est la coordonnée du curseur.

→ On établit aussi une condition : Si l'évènement saisi est une touche directionnelle, alors

la coordonnée de position du curseur va être modifié pour respecter la direction de la touche.

→ La fonction va alors retourner les coordonnées modifiés pour ensuite être ré-utilisé par la fonction affiche_cases_choix dans la boucle principale.

II / Le placement

Une fois le choix fait, une nouvelle fenêtre s'affiche et le programme passe à la boucle principale du placement des bateaux.

A/ Le quadrillage

Quadrillage (ordre)

La fonction qui va permettre de dessiner le quadrillage. Pour cela, on a les variables x et y. On établit deux boucles : Une boucle pour tracer les lignes verticales et une autre pour tracer les lignes horizontales. Les variables x et y vont être les compteurs des boucles, mais aussi les coordonnées de position des lignes : Par exemple, quand les compteurs x et y vont être incrémentés de 50, la ligne va se décaler de 50 pixels pour tracer une nouvelle ligne/colonne.

L'ordre est une variable de condition qui verifie dans quel partie du jeu nous sommes pour creer un quadrillage adéquat (1 quadrillage pour le placement et 2 pour la bataille)

B/ La structure de données des bateaux

Bateau_liste(taille_bateau, liste_bateau, sens)

Cette fonction va être le coeur la structure de données des bateaux. C'est la fonction qui stocke les coordonnées des cases des bateaux dans une liste. Pour cela, on initialise d'abord la liste avec une coordonnée (x, y) (comme [(6, 5)], puis on établit une boucle while de 0 à taille_bateau (= nombre de cases d'un bateau). Le corps de la boucle incrémente ou décrémente (selon le sens du bateau) la valeur de la coordonnée qui précède pour ensuite l'ajouter dans la liste.

Par exemple, pour un bateau à 3 cases tourné verticalement :

1er itération 2eme itération
[(6, 5)] → [(6, 5) , (6, 4)] → [(6, 5) , (6, 4), (6, 3)]

L'ordonnée de la dernière coordonnée est soustrait par 1 à chaque itération pour créer le corps du bateau 1 par 1

On retourne alors cette liste nouvellement crée.

C/ L'affichage

Affichage_bateau(liste_bateau, taille_bateau, liste_bateau_placé, placé, couleur)

Cette fonction prend les valeurs de la liste crée et affiche le bateau à partir de ces

coordonnées. On crée une boucle de 0 à taille_bateau, on parcourt la liste et on convertit les coordonnées en pixels en les multipliant par la taille des cases du quadrillage. On reprend ces valeurs nouvellement convertis pour créer le rectangle. La fonction a deux parties : Si le joueur place le bateau (liste_bateau), et si le bateau est déjà placé (liste_bateau_place). Pour cela, on prend comme paramètre une variable 'placé ' qui vérifie si la fonction est appelée pour afficher un bateau qu'on tente de placer ou pour afficher un bateau déjà placé.

Placer_bateaux (liste_bateau, touche, taille_bateau):

La fonction prend les coordonnées de la liste crée avec bateau_liste() qu'on stocke dans des variables. Ensuite, elle étudie deux conditions : Si une des touches directionnelles est pressée et si le bateau n'est pas aux bords de la fenêtre. Si les deux conditions sont valides, alors elle additionne (ou soustrait, dépendant de la direction) les coordonnées des cases de 1. Elle retourne alors la nouvelle coordonnée.

III / La bataille

Après avoir placé les bateaux, le programme passe à la partie de la bataille.

D'abord, à l'aide d'une boucle, on ajoute aux coordonnées des bateaux une nouvelle valeur qui est une valeur booléenne. En effet, elle va agir comme l'indice de l'état d'un bateau : Si False, la case est active et si True, la case est inactive.

Ensuite, on initialise les coordonnées du bateau adverse. L'ordi, grâce à une variable taille_bateau_ordi qui stocke un nombre aléatoire entre 1 et 6, va choisir aléatoirement sa composition de bateaux.

A / Gestion des clics

Vient alors la boucle principale du jeu. La partie principale de cette boucle intervient quand le joueur effectue un clic gauche. Une fois cette condition remplie, on prend les coordonnées du clic gauche (qu'on nommera les variables x_joueur, y_joueur) et on la stocke dans une liste lst_croix qui regroupe les coordonnées de tout les clics. On introduit deux nouvelles variables x_ordi et y_ordi qui vont stocker les coordonnées de tir de l'ordi, qui seront des coordonnées aléatoires. C'est ici que la fonction affiche_croix intervient.

Affiche_croix(x, y, tour) :

Cette fonction prend les coordonnées du clic, la convertit en pixels, et affiche la croix en

traçant deux lignes à l'aide d'upemtk. Avant ça, on vérifie si les coordonnées du clic se situent bien dans le quadrillage. Sinon, on ne fait rien et on renvoie False. Si c'est le cas, on renvoie True (qui va nous permettre de vérifier si le joueur a bien fait son tir)

Ensuite, on vérifie deux conditions : Si le joueur a bien tiré (si affiche_croix == True), et si il a bien tiré dans une case vide (si [x_joueur, y_joueur] not in lst_croix). Si c'est le cas, alors on appelle la fonction affiche_croix, mais cette fois avec les coordonnées du tir de l'ordinateur en paramètre: Un tir de l'ordinateur vers le plateau du joueur va alors se faire en même temps que le tir du joueur chez le plateau adverse.

B/ Gestion des contacts

Pour gérer le contact entre les tirs et les bateaux, on appelle deux fonctions :

Contact(clic, liste_bateau, tour)

Le rôle de cette fonction est de vérifier si les coordonnées de la variable clic appartiennent à celle d'une des cases d'un bateau (liste_bateau). Si c'est le cas, elle modifie l'état de cette case et renvoie la liste du bateau nouvellement modifié. Pour cela, on crée une boucle imbriqué (une première boucle qui parcourt le bateau à l'aide d'un compteur « bateau » et une deuxième qui parcourt les cases du bateau parcourut à l'aide d'un compteur « case ») et on vérifie l'appartenance pour chaque case de chaque bateau. Le paramètre « tour » détermine le tour.

Affiche_toucher(liste_bateau)

Cette fonction prend en paramètre la liste de bateaux de l'un des des joueurs et vérifie l'état de chaque case de chaque bateau. Si une case a été touché, on dessine un rectangle rouge qui montre qu'un bateau a été touché. On utilise alors la même boucle imbriqué pour vérifier l'état des bateaux.

C/ Gestion des annonces et résultats

Touché / Rien

Variables principales : touche_j1, touche_ordi (booléens)

Démarche : On modifie la valeur de ces variables dès que la fonction Contact modifie l'état d'un bateau. Pour modifier dans une fonction une variable qui est en dehors de la fonction, on utilise la commande **global.**

Ensuite, dans la boucle principale, on vérifie une condition. Si touche_j1 ou touche_ordi est égal à True, on affiche « Touché! ». Sinon ,on affiche « Rien. »

En vue

Fonction/variable principales : La fonction « en_vue »

En_vue (x, y, liste_bateau)

Cette fonction vérifie et renvoie True si un navire se situe dans une des 8 cases adjacentes du tir. Pour cela, on parcourt liste_bateau avec la même boucle imbriqué et on vérifie si au moins l'une des 8 cases adjacentes appartiennent à la liste.

Dans la boucle, on établit alors une condition : Si la fonction == True, alors on affiche 'En vue !' en dessous du plateau de jeu.

Coulé

Fonction/variable principales : La fonction « coulé », la variable « coule »

Coulé (liste_bateau, taille_bateau, tour) :

Fonction qui vérifie et renvoie True si un bateau a coulé. La fonction parcourt « liste_bateau » à l'aide d'une boucle imbriqué, puis incrémente « coule » de 1 si toutes les cases du bateau parcourut par la boucle est à l'état « True ». Si « coule » atteint la valeur du nombre de bateaux qui a coulé, on sort alors de la boucle et on renvoie True. Le paramètre « tour » détermine pour quel joueur cette fonction doit être appelée.

Dans la boucle, on établit alors une condition: Si la fonction == True, alors on affiche 'Coulé !' en dessous du plateau de jeu.

Fin de jeu

Le jeu se finit quand l'un des deux joueurs n'a plus de bateaux. La fonction fin_de_jeu en est responsable.

Fin_de_jeu(liste_bateau)

La fonction parcourt la liste pris en paramètre et renvoie False si l'état de toutes les cases de la liste est égal à True. On procède à la même manière que les autres fonctions pour parcourir la liste.

A la fin de la boucle principale, on étudie cette condition : Si la fonction == False, alors on efface tout grâce à une fonction d'upemtk et on affiche un écran de victoire ou de défaite. On ferme alors la fenêtre.

Pensées du projet

Difficultés rencontrés

La difficulté générale de notre projet était de s'organiser et de gérer notre temps : En effet, avec les nombreux travaux à faire en dehors de la programmation, nous avions eu du mal à trouver notre temps pour non seulement travailler individuellement, mais aussi en groupe étant donné que nous ne sommes pas dans le même groupe de TD.

En dehors de l'organisation, nous avions eu pas mal de difficultés à gérer les boucles en général car nous avions souvent des listes trop grandes (comme la liste des bateaux du joueur) ou trop de listes imbriqués. Gérer ces listes nous a permis de mieux comprendre comment marche les listes de listes.

Ce que nous a appris ce projet

Ce projet nous a permis de comprendre plus en détails les fonctionnalités d'une fonction et nous a ouvert les yeux sur sa polyvalence et son importance pour l'organisation du programme. Aussi, ce projet nous a aidé à nous familiariser avec upemtk. Enfin, le projet nous a appris à savoir comment gérer les projets à deux plus efficacement et plus rapidement.