

# Bataille Navale

## Séance du 19.12.2023 :

Tout d'abord, nous avons séparé notre projet en plusieurs étapes, permettant de se rendre compte des objectifs dans notre jeu, et des différentes tâches que nous avons à effectuer :

- Créer la grille
- Placer des bateaux aléatoirement
- Sélectionner une case
- Noircir la case touchée si un bateau s'y trouve (plus dur: mettre une croix)
- Avoir des bateaux de tailles différentes
- Afficher un score (nb de restant, nb d'erreurs)
- Mettre un chronomètre limitant le temps pour choisir la nouvelle case
- Afficher un bateau une fois coulé
- 
- Avoir deux fenêtre de jeux (pouvoir jouer à deux) :
  - pouvoir placer ses propres bateaux
  - afficher la répartition des bateaux de chacun des joueurs ( avec les bateaux coulés ou non ) ainsi que la grille adverse non découverte successivement;
  - Python en ligne ? Même réseau
- Faire un code clair, en anglais et respecter les conventions

Nous souhaitons, dans un premier temps, créer la grille pour la bataille navale. Pour cela, nous allons utiliser du pygame.

La première idée qui nous est venue est d'essayer d'afficher une grille dont les colonnes et lignes seront indexées par des lettres et des chiffres, mais il nous semble difficile de gérer un affichage de lettres et de chiffres dans la fenêtre.

Nous avons donc trouvé la commande `pygame.MOUSEBUTTONDOWN` qui nous permettrait de faire un jeu cliquable et nous permettrait de nous affranchir du soucis de l'indexation des cases dans un premier temps (même si à terme cela pourrait être utile).

Notre objectif final serait d'avoir un jeu qui peut être joué à deux, mais dans un premier temps nous allons commencer à essayer de jouer contre la machine.

Nous avons réussi à faire un échiquier avec des couleurs bleutées (rappelant la mer) mais nous souhaiterions plutôt faire une grille. Nous avons donc utilisé la commande `pygame.draw.line()`.

Une fois la grille tracée, nous voulons y placer des bateaux. Pour cela, nous allons d'abord essayer avec des bateaux de taille 1, avant de complexifier le tout en mettant des bateaux plus grands et avec des formes différentes. La grille est représentée dans notre code par une matrice (liste de liste). Avant que le joueur commence à jouer les cases ne présentant pas de bateau contiennent None, celle contenant un bateau de taille 1 contiennent la chaîne de caractère '0'

Pour l'instant, nous allons faire choisir à l'ordinateur la position des bateaux de taille 1 de manière aléatoire. Nous avons également écrit un code pour faire apparaître des croix lorsqu'on touche un bateau, et colorier la case en noir lorsqu'on loupe pour noter les endroits déjà essayés. Le programme fonctionne mais à chaque fois que l'écran se remet à jour, il y a apparition d'un écran noir, ce qui nous déplaît. On peut pour éviter cela utiliser la commande `pygame.display.flip()`.

problème rencontré à corriger: quand on appuie à nouveau sur une case sur laquelle se trouvait un bateau et qui a déjà été coulé la case devient noire: il faut corriger cela pour que la croix reste même si on appuie à nouveau sur la case. *(Nous avons corrigé ce problème en fin de séance en comprenant ce qui causait ce problème ( utilisation d'un else au lieu d'un elif))*

Idée d'amélioration pour la prochaine fois :

- faire apparaître un écran qui dit 'You won' à la fin quand tous les bateaux ont été coulés. et faire apparaître les messages à chaque coup dans la fenêtre et non le terminal
- Augmenter la taille des bateaux : d'abord en ligne puis formes bizarres ?
- Afficher les scores de bateaux touchés

## **Séance du 09.01.2024:**

Le but de cette séance a été d'introduire des bateaux plus longs : faire en sorte qu'ils ne sortent pas de la grille, gérer les différentes orientations possibles des bateaux et qu'ils ne se chevauchent pas. Nous avons aussi voulu faire apparaître les bateaux une fois coulés, en prenant un design trouvé sur internet par exemple. Néanmoins, après plusieurs essais, nous n'avons pas réussi à faire coïncider la taille des cases avec l'image. En effet, nous avons fait des tentatives avec la commande `screen.blit (image, position)`, mais cela était assez difficile à mettre en place : nous avons téléchargé des images de bateaux au format png mais celles-ci sont trop grandes et dépassent donc très largement de la case dans laquelle elles sont censées être. Par ailleurs, il faudrait avoir trois types d'image de bateaux différents en fonction de la taille du bateau mais aucune image n'a le bon format pour rentrer dans les cellules de notre grille. Nous avons alors préféré continuer à développer d'autres fonctionnalités du jeu. Si nous avons le temps, nous chercherons à dessiner un bateau nous même avec la fonction `pygame.draw.line`.

Bilan sur la manière dont nous avons rempli notre matrice `grid_state` pour rendre compte de l'état de la grille de jeu:

avant le début de la partie:

- None : pas de bateau dans cette case
- '0' : un bateau de taille 1 a été placé sur la case
- [(,)] (une liste de un ou deux tuples) : la case contient un bateau de taille 2 ou 3, le ou les tuples contenus dans la liste correspondent aux coordonnées des autres cases occupées par ce même bateaux

après le début de la partie:

- 'X' : le bateau qui était avant sur cette case a été coulé (quelque soit sa taille)
- 'T' : cette case a déjà été touchée, un bateau était présent sur cette case mais le reste du bateau n'a pas été touché, il n'est donc pas encore coulé
- '.' : cette case a été touchée mais elle ne contenait pas de bateau

Au niveau de la grille, il a fallu introduire de nouvelles bandes pour afficher les lettres et les chiffres, pour la rendre plus lisible et proche du vrai jeu "la bataille navale". C'était assez compliqué jusqu'à trouver la commande `screen.blit`. Il a également fallu changer tous les indices dans l'affichage (à cause du décalage par l'ajout des bandes). Ensuite, on cherche à faire apparaître le score de l'utilisateur. Esthétiquement, nous avons préféré mettre celui-ci en bas : il faut donc rajouter une bande blanche en bas, et parvenir à faire varier le score en temps réel. Une première difficulté rencontrée a été de réussir à écrire la phrase en y insérant la variable `score` et `amount boat` : en effet, la commande ne pouvait pas être "Score :", `score` "/", `amount boat` comme on aurait pu s'y attendre. Il faut que le tout soit en une seule phrase, c'est-à-dire tout entre guillemets. Nous avons alors pensé à utiliser une commande de la forme : `phrase = f"Score : {score} / {amount_boat}"`, qui affiche bien la valeur des variables `score` et `amount boat`.

Pour être plus clair, nous avons aussi souhaité afficher les phrases de réussites et d'échecs directement sur l'interface du jeu.

### Séance du 16.01.2024 :

Pour faire apparaître le temps passé sur le jeu, nous voulons utiliser la commande `time.time()` notamment. Le problème est que l'affichage ne se fait pas toutes les secondes de manière exacte : il y a toujours un décalage lié au chargement etc. Nous avons réussi à faire apparaître le timer sur une fenêtre. Néanmoins, on a eu plusieurs problèmes en essayant de l'insérer dans le code initial, cela repose sans doute sur la problématique de où est ce qu'il faut insérer la fonction permettant au temps de défiler.

Comme prévu à la séance précédente, nous avons codé l'affichage des commentaires dans la fenêtre de jeu suite à la tentative. L'enjeu est de les placer correctement pour que chacun de commentaires rentre dans la fenêtre.

Une des difficultés que nous avons trouvé aussi est la fusion, qui a été plus compliquée que prévue: en rassemblant nos différentes versions il y a beaucoup de débogage à faire, que ce soit en 'mergeant' nos versions à la main ou à travers git. Effectivement, nous avons pas mal modifié certaines fonctions communes aux différentes versions.

Durant la suite de la séance, nous avons cherché à pouvoir jouer à deux sur cette bataille navale. La première difficulté a été de modifier l'affichage pour faire deux grilles (il fallait être très rigoureux dans les indices). L'optique est d'avoir une grille qui nous permet de placer nos bateaux puis de voir où l'adversaire à jouer, et une autre grille où nous jouons nous-mêmes.

Plus encore, la première étape de ce jeu en commun est de pouvoir faire placer les bateaux au joueur : nous avons donc fait un programme qui permet au joueur de placer les bateaux de différentes tailles à la souris, s'assurer que les cases choisies sont contiguës pour les bateaux de taille 2 et 3.

Une de nos volontés a ensuite été de mettre ce jeu en ligne : nous nous sommes donc intéressées à la documentation sur la bibliothèque pyro5. Néanmoins, il y avait très peu d'informations et les sites ne sont pas tous très clairs. Au vu du temps restant, nous préférons perfectionner notre jeu, plutôt que de se lancer dans cette nouvelle fonctionnalité qui prendra trop de temps.

Objectifs pour la prochaine séance :

- Commenter le code
- Relecture du code

### Séance du 23.01.2024:






Ce que nous avons souhaité faire durant cette séance a été notamment de retravailler notre code afin de le rendre le plus lisible et clair possible. Nous sommes donc repassées sur les noms de fonctions, les commentaires, les espaces etc.








En faisant cela, nous avons remarqué que l'affichage "What a pity, you've missed" apparaît même quand on touche un bateau de plusieurs cases. En fait, les commentaires de victoire ne fonctionnent que lorsqu'on coule un bateau de taille 1. Nous avons donc travaillé à nouveau sur cette partie du code pour corriger cela.




De plus, l'une de nos volontés a été de faire des bateaux de différentes formes : il peut, dans notre code, avoir des bateaux en forme d'angles droits, complexifiant le jeu.

et traduire en anglais certaines choses

Retour sur nos objectifs initiaux :

- **Créer la grille**  : utilisation des fonctionnalités de pygame pour tracer des rectangles, lignes, mais aussi apprentissage de nouvelles commandes pour faire apparaître des lettres et des chiffres
- **Placer des bateaux aléatoirement**  : utilisation de la commande randint et création de matrices développées
- **Sélectionner une case**  : Découverte de la fonctionnalité pygame.mouseBUTTONDOWN
- **Noircir la case touchée si un bateau s'y trouve**  (plus dur: mettre une croix)  : Réfléchir à la matrice créée et parvenir à l'actualiser, nécessité de bien relier notre fonction battleship\_game avec l'affichage. Pour la croix : appropriation des indices et rigueur nécessaire liée à la taille de la case

- **Avoir des bateaux de tailles différentes**  : Utilisation de tuples à parcourir pour pouvoir adapter l'affichage lorsqu'une partie d'un bateau est touchée, réussir à prendre en compte la taille totale d'un bateau avant de le placer dans le programme
- **Afficher un score** (nb de restant, nb d'erreurs)  : Réussir à écrire une phrase avec une variable, relier à nouveau les différentes fonctions, apprentissage de la commande screen.blit + affichage des phrases pour plus de lisibilité
- **Mettre un chronomètre limitant le temps pour choisir la nouvelle case** :  Réussite de la création d'un programme chronomètre mais échec dans l'insertion de celui-ci dans le programme total.
- **Afficher un bateau une fois coulé**  : Échec à cause de la taille des images mais compréhension de comment faire afficher une image sur une fenêtre pygame
- Avoir deux fenêtre de jeux (pouvoir jouer à deux) :
  - **Pouvoir placer ses propres bateaux**  : Programme réalisé, contact plus fort avec la fonction pygame.mousebuttondown (mais le programme n'a pas pu être testé par manque de temps et parce que cette partie du projet n'a pas pu réellement aboutir)
  - **Afficher la répartition des bateaux de chacun des joueurs** ( avec les bateaux coulés ou non ) ainsi que la **grille adverse** non découverte successivement  ; Double grille créée (rigueur au niveau des indices, long), pas eu le temps pour le reste mais même programme à réadapter légèrement pour avoir les actions des deux en parallèle
  - **Python en ligne?**  : Découverte de pyro5 mais pas le temps de l'utiliser : peu de documentation et difficile à comprendre + très ambitieux (demandait beaucoup de recherches en autonomie) → nous avons préféré perfectionné notre code déjà existant plutôt que de se lancer dans cette nouvelle partie que nous n'aurions très certainement pas pu finir.
- **Faire un code clair, en anglais et respecter les conventions** : nous avons essayé de faire au mieux et on trouve ça plutôt réussi !

 : objectif réalisé  
 : objectif presque atteint  
 : objectif abandonné

En conclusion, ce projet nous a beaucoup apporté : il nous a permis de mieux utiliser git, nous sommes maintenant plus à l'aise pour coder à plusieurs, de réussir à mieux aborder un projet (le découper en petites tâches notamment), de nous confronter à travailler sur un même code plusieurs semaines d'affilées (ce qui nous a notamment obligées à produire un code clair pour ne pas avoir trop de mal à se remettre dedans d'une semaine à l'autre).

