Projet - Jeu de Quilles

Darina CHAN & Aubin RAHMANI

1) Introduction au projet

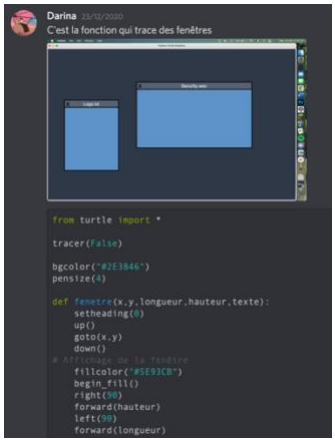
Durant ce mois de décembre, nous avions un projet de codage Python à réaliser en binôme. Il s'agit d'un jeu de quille avec plusieurs contraintes qui nous ont été imposées.

a) Règles du jeu

Les règles de ce jeu sont semblables à celles du bowling et du jeu d'allumette. Le joueur joue contre un ordinateur. Un nombre de quille aléatoire, allant de 3 à 15 s'affiche sur l'écran. Le joueur commence. Plusieurs choix se présentent à lui :

- Tirer au milieu afin de renverser les 2 quilles au centre
- Tirer sur le côté (droite ou gauche) afin de renverser une quille au bord

Ensuite vient le tour de l'ordinateur qui joue aléatoirement.
Cependant, le jeu fonctionne sous forme de ligne. C'est-à-dire que lorsque le joueur ou bien son adversaire tire au milieu, cela crée une coupure dans la rangée de quille. La ligne est alors divisée en deux lignes distinctes. À chaque tour, les joueurs doivent donc aussi choisir sur quelle ligne ils souhaitent jouer. Celui qui fait tomber la dernière quille remporte la partie. Si le nombre de quille sur une ligne est impair lorsque l'on tire au milieu, c'est la quille à gauche de la quille au centre qui tombe.



b) Organisation du projet

Nous n'avions pas de grande expérience dans le codage et c'était notre tout premier projet. Travailler en binôme était compliqué et demandait une grande organisation. La communication est la clé pour réussir tout travail de groupe. C'est pourquoi nous avons choisi de communiquer à travers le réseau social **Discord** qui est une plateforme que nous utilisons fréquemment. Cela nous permettait de faire des appels pour travailler tout en respectant les mesures sanitaires. De plus, Discord nous permet de faire des partages d'écrans, mais aussi de partager nos codes sous forme de texte grâce au formatage ou sous forme de fichier en toute simplicité.
Pour nous aiguiller, ce projet a été divisé en deux grandes étapes. Tout d'abord, nous devions rédiger une version textuelle, c'est-à-dire une version du jeu qui était fonctionnelle avec un affichage basique sur **Python**. Par la suite, nous avons dû réaliser une interface graphique à l'aide de **Turtle** et l'intégrer au jeu.

c) Mise en commun des idées

Plusieurs éléments de l'interface graphique étaient imposés. En effet, nous devions choisir un thème qui se retrouverait dans le visuel du jeu et dans l'apparence des quilles. Nous avons choisi le thème **Among us** qui est un jeu très en vogue en ce moment.
Le choix de ce thème s'est fait très rapidement car cela nous permettait de faire un clin d'œil à un jeu que nous apprécions tous les deux. Le design du jeu était très inspirant et nous avons très facilement trouvé les idées pour répondre à toutes les contraintes imposées.

2) Codage de la version textuelle

Notre première approche a été de travailler sur la version textuelle. Ce fut nos premiers pas dans ce projet et nous nous sommes très vite rendu compte de la complexité de la tâche.

a) Répartition des tâches

Ce programme a été décomposé en plusieurs parties. Pour commencer et prendre nos repères, nous avons choisi de travailler à deux sur la procédure **afficheQuilles** à travers la plateforme Discord. Après avoir défini cette première procédure, nous nous sommes réparti les tâches.
Aubin s'est donc chargé de programmer une fonction pour simuler les choix aléatoires de l'ordinateur ainsi que de programmer la fonction qui prend en compte les choix du joueur car elle est semblable à la première fonction. Cependant, la tâche était plus difficile car le joueur effectue ses propres choix.
En parallèle Darina, devait coder deux fonctions qui mettent à jour l'état des quilles lorsque le joueur ou l'ordinateur joue en fonction de leur choix (côté ou milieu). Ces deux dernières ont ensuite été utilisées dans une fonction appelée **jouer**.
Enfin, la dernière étape pour finaliser la version textuelle était d'écrire le moteur du jeu en suivant un algorithme donné.

b) Démarche pour le codage

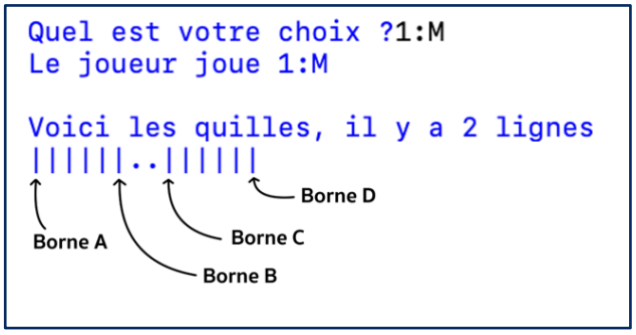
La fonction **ordiJoue** est assez courte puisque l'algorithme récupère les lignes restantes et choisit aléatoirement l'une d'entre elle. Puis vient le choix de la direction. Encore une fois, l'ordinateur choisit aléatoirement entre la gauche, le milieu ou la droite. Puis la fonction retourne le choix de l'ordinateur sous forme de string qui respecte la syntaxe ligne:direction (où ligne est un chiffre et direction une lettre « G », « M » ou « D »).
La fonction **joueurJoue** est similaire, sauf que les choix se font sous forme d'input. Cependant, il faut veiller à ce que la syntaxe soit bien respectée. Si le choix est validé, ce dernier sera retourné. Dans le cas contraire, le programme doit redemander à l'utilisateur de faire un choix. Ainsi, chaque type d'erreur doit être étudié :

- Lorsque le **1er caractère** n'est pas un chiffre ou qu'il désigne une ligne inexistante
- Lorsque le **2ème caractère** est différent de « : »
- Lorsque le **3ème caractère** est différent de « G », « M » ou « D »
- Lorsque l'utilisateur utilise plus ou moins de **3 caractères**.

Il doit d'abord vérifier que le 1er caractère est un chiffre puis ensuite vérifier qu'il désigne une ligne du jeu. Dans cas le contraire, si ce caractère est une lettre par exemple, le programme ne pourra pas utiliser cette lettre pour vérifier s'il existe bien une telle ligne dans le jeu car ce n'est pas un chiffre. Il ne peut donc pas désigner un numéro de ligne.

La fonction **jouerMilieu** calcule les 4 nouvelles bornes qui représentent les lignes restantes après la coupure.
Suivant le jeu, les lignes peuvent disparaitre. On distingue donc trois cas :

- Cas où la ligne est simplement séparée en deux comme sur l'image ci-dessus
- Cas où il ne reste plus qu'une quille sur la ligne
- Cas où la ligne disparait



Lorsque l'on a identifié le cas dans lequel on se trouve, on efface les bornes qui ont été modifiées afin d'insérer les nouvelles.
La fonction **jouerCote** est semblable à la fonction précédente. Cette fois, on utilise seulement les bornes A et D car lorsqu'on joue sur le côté, ce sont les seules quilles dont l'état peut changer.

3) Codage de l'interface graphique

Coder l'interface graphique était la partie la plus ludique car elle faisait appel à notre créativité. Pouvoir faire des références à notre jeu favoris était très motivant. Cela nous a poussé à ne laisser aucun détail au hasard !

a) Répartition des tâches

Travailler avec des formes pour faire en sorte que l'utilisateur comprenne ce que l'on a voulu représenter était quelque chose de nouveau pour nous. Les dessins que nous voulions représenter étaient assez simple à réaliser. De plus, grâce au brouillon réalisé Darina, nous avons pu prévoir chaque détail de notre interface graphique, ce qui a facilité la répartition du travail. Les dessins que nous voulions représenter étaient assez simple à réaliser à partir de rectangles. Nous avons donc codé deux fonctions qui nous ont été d'une grande aide :

- La fonction **arrondis** qui permettait de faire des coins arrondis en fonction du sens (droite ou gauche)
- La fonction **rectangle** qui permettait de faire des rectangles (avec des coins arrondis) en fonction de la longueur et largeur que nous désirions.

Ces deux fonctions ont permis à Aubin de faire le dessin des fenêtres, des post-it ainsi que des fichiers qui donnent réellement l'impression d'être sur le bureau d'un ordinateur. À partir de ces deux fonctions, Darina a aussi pu réaliser les fonctions qui dessinent l'interface des fenêtres mais aussi certaines parties des personnages.

b) Explication des choix

Nous avons choisi de faire une interface qui représente l'écran d'une machine, dans le but de rester dans le principe du jeu original où le personnage manipule des appareils. L'interface du jeu original a un design particulier que nous avons étudié. Nous avons utilisé un stylo de taille 5 afin de reproduire le style de dessin d'Among us qui utilise un trait épais. Nous avons choisi des teintes grises et bleues de sorte afin de faire un rappel au jeu. L'état debout de la quille est représenté par un personnage debout. C'est la fonction **quilleDebout**. Quant à la quille tombée, nous l'avons illustrée avec un personnage découpé à l'aide de la fonction **quilleTombee** qui dérive de la quille debout.
Nous avons fait apparaître deux fenêtres sur l'écran :

- L'une permet d'afficher les quilles sur une caméra de sécurité, tout comme dans le jeu où le personnage peut observer les autres joueurs depuis les vidéos surveillance.
- La seconde permet de visionner les actions précédentes. Elle apparait sous forme de chat avec des boutons pour communiquer.
- L'affichage des logs est semblable au chat présent dans le jeu. Nous avons utilisé la fonction **quilleTournee** qui est une dérivée de la fonction **quilleDebout** afin de faire le personnage debout mais tourné dans le sens contraire.

Images issues du jeu original — Image issue du projet

Nous avons aussi fait un clin d'œil au jeu original en reprenant la fameuse image du personnage éjecté pour marquer la fin de la partie. Pour cela, nous avons aussi utilisé une dérivée de la fonction **quilleDebout** et nous l'avons penchée vers la droite de 45°, ce qui a donné la fonction **quilleFlotante**.

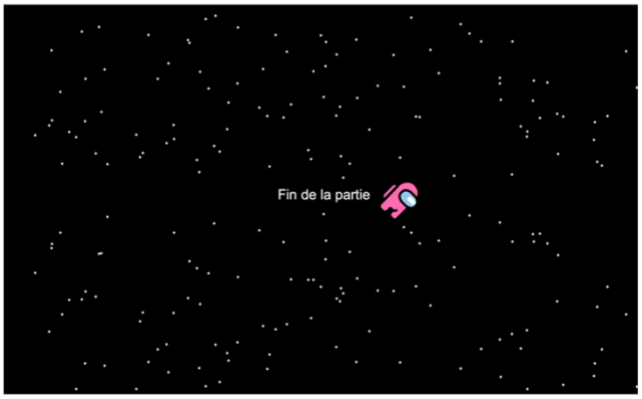


Image issue du jeu original — Image issue du projet

4) Assemblage des deux programmes

L'assemblage des deux programmes était de loin la partie la plus complexe, mais c'était très motivant de savoir que c'était la dernière étape de notre projet. Nous étions très déterminés face à l'idée de voir le rendu final de ce dernier.

a) Problèmes rencontrés

L'un des plus gros problèmes était l'affichage des quilles. Nous devions calculer la place de chaque quille en fonction du nombre de quilles initial. Mais l'affichage des quilles en deux rangées rendait le codage plus complexe. Pour commencer, nous avons tout d'abord rédigé la procédure **placementQuille** ainsi que la procédure **nombreQuille** qui permettaient de placer les quilles en fonction du nombre initial de quilles. En effet, lorsqu'il y a entre 3 et 8 quilles, l'affichage ne se fait que sur une seule rangée. En revanche, lorsque qu'il y a plus de 8 quilles, l'affichage se fait sur 2 rangées. Placer les quilles de façon à ce que ces dernières soient bien centrées sur la fenêtre quel que soit le nombre de rangées n'était pas simple. La difficulté a été de déterminer la/les quille(s) au centre pour chaque cas. Il y avait toujours une confusion pour savoir où était le milieu en fonction du nombre de quille pair ou impair. D'autant plus que les listes partent de 0, ce qui nous a causé beaucoup de soucis de calcul dans notre programme.
De plus, lors de notre entretient avec le professeur, ce dernier nous a recommandé de numéroter les quilles afin que l'utilisateur ne soit pas perturbé par l'affichage des quilles qui se faisait sur deux rangées. Cela compliquait davantage la procédure car nous devions faire en sorte que la numérotation soit ordonnée avec l'ordre d'apparition des quilles. L'un des problèmes rencontrés était le cas où il y avait une seconde rangée car il ne fallait pas recommencer à numéroter les quilles depuis le début.

b) Solutions apportées

Pour pouvoir faire un affichage centré des quilles, nous avons choisi de dessiner chaque quille une par une, rangée par rangée en partant du centre et en allant vers les bords. Il y a le même espacement entre chaque quille. Grâce à des calculs, nous avons adapté la numérotation des quilles à ce système d'affichage de quille. Nous avons donc divisé la fonction en deux grands cas :

- Nombre de quille paire sur la rangée
- Nombre de quille impaire sur la rangée

Pour savoir si la quille est debout ou tombée, nous nous aidons de la procédure **afficheQuille** de la version textuelle. Pour étudier l'état des quilles, nous partons du centre pour ensuite étudier les quilles à gauche et à droite du centre.

Pour éviter la confusion entre le nombre de lignes en jeu et l'affichage des quilles qui se fait sur 2 rangées, nous avons ajouté en bas un post-it qui indique le nombre de lignes restantes. Nous avons aussi apporté une petite amélioration en ajoutant le nombre de tours passés pour les plus compétiteurs ou alors pour les plus curieux.

c) Création des modules

Le programme que nous avions rédigé n'était regroupé que sur un seul fichier et comprenait plus de 600 lignes. De ce fait, le code était peu lisible, d'une part à cause des nombreux commentaires et d'autre part à cause des fonctions de dessin et de calcul qu'on avait du mal à distinguer. Nous avons donc décidé de diviser le programme en 4 fichiers :

- **gui1.py** qui contient les dessins des quilles
- **gui2.py** qui contient les dessins de l'interface du jeu
- **engine.py** qui contient tous les calculs
- **quilles.py** qui contient le moteur du jeu

La création de modules nous a permis d'obtenir un code plus clair, mais surtout, cela nous a permis de corriger toutes nos erreurs, comme par exemple des problèmes de variables définies hors des fonctions et qui ne sont pas utilisées comme des paramètres car en effet, cela nous empêchait de séparer les fonctions dans des fichiers distincts.

Conclusion

Ce projet était une expérience enrichissante, d'une part parce que c'était la première fois que nous codions en binôme mais aussi parce que c'était le premier projet que nous entreprenions. Travailler sur un projet sur plusieurs jours pourrait être décourageant pour certains mais à notre grande surprise, le codage de ce jeu était très stimulant, ce qui apportait satisfaction lorsque nous étions productifs.