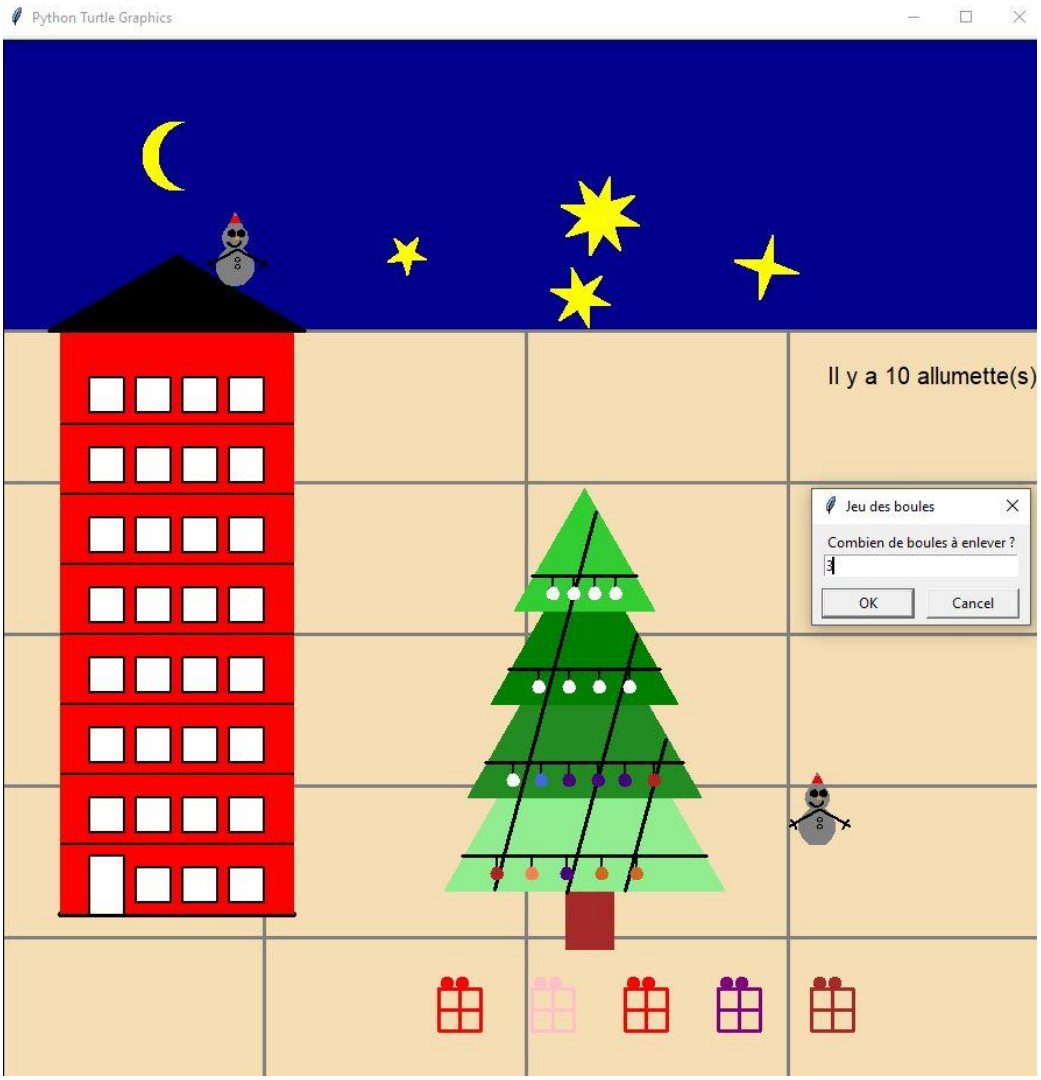

Micro-rapport du projet python

ARBRE DE NOEL : NIVEAU DE CONTRAT 1. NOTE OBTENUE : 17/20

MEURILLON Alex et PRESSENDA Ugo | Groupe 4 | Semaine 21-22

Introduction

Ce projet est une reprise de celui réalisé lors du premier semestre. Le but de ce premier était de créer un jeu piloté par une interface graphique (turtle()) dans notre cas. Nous avions émis la critique dans ce projet, de jouer sans adversité. A travers ce second projet, nous allons implémenter dans notre code du premier semestre, les fonctions nécessaires pour permettre à l'ordinateur de gagner à tous les coups.

Reprise du projet python du semestre 1

Le sapin est constitué de plusieurs guirlandes où des boules de Noël sont placées dessus. Entre 10 et 20 sont placées à chaque début de jeu. Le nombre de boules est décidé aléatoirement. La règle du jeu est fixe : 1,2 ou 3 retraits chaque tour est possible. Cette règle est fournie dans une liste, que l'on peut modifier dans le code du programme. Une fois les guirlandes placées et la règle posée, le jeu peut commencer.

Seul une pile d'allumette, de 10 à 20 allumettes, est présente dans le jeu.

En réalisant le jeu, nous n'avions pas réussi à coder des fonctions essentielles dans le jeu. En effet, la règle était fixe, nous ne pouvions pas la modifier, mise à part de la modifier directement dans le code.

Description des difficultés rencontrées

Lors de la reprise de notre code, nous n'avons pas forcément ressentis de difficultés. Dès le TD de la semaine 21, nous avons corriger tous les bugs présentés ci-dessus. De plus, nous avons aussi implémenter l'algorithme de stratégie gagnante, que nous détaillerons plus tard.

Le nom des fonctions et des variables, la pertinence des commentaires ainsi que la structure du code nous ont permis de comprendre quelle partie du code modifiée pour fixer les bugs du premier semestre et de commencer le travail à réaliser.

Ce premier projet en groupe de la PeiP1 nous avait demandé de l'organisation. Nous nous étions basés sur une décomposition simple à comprendre afin de réaliser nos parties de code de part et d'autre et de les joindre pour le rendu.

De plus, nous prenons de l'expérience en python grâce aux cours de programmation impérative et de POO ce qui facilite la reprise et l'écriture du programme.

Description des modifications apportées à notre programme

Nous avons tout d'abord récupéré les codes permettant de déterminer qui avait la stratégie gagnante fait dans un des TD précèdent.

- Algorithme de stratégie gagnante

La fonction valPileAllumettes(n, r) calcule la valeur de la pile d'allumettes n en utilisant les règles du jeu. Elle prend en paramètres n, qui représente le nombre d'allumettes dans la pile, et r, qui est une liste des nombres d'allumettes que le joueur peut enlever à chaque tour. Elle utilise la récursion pour calculer les valeurs possibles de la pile d'allumettes en enlevant différents nombres d'allumettes de la pile et en appelant récursivement la fonction pour la pile restante.

La fonction mex(l) est utilisée pour calculer le minimum exclusif (minimum excluant) d'une liste l, c'est-à-dire le plus petit nombre qui n'est pas présent dans la liste.

La fonction sumNimXOR(n, m) calcule la somme XOR (ou exclusif) entre deux nombres n et m.

La fonction valJeuAllumettes(a, r) calcule la valeur du jeu d'allumettes en utilisant la stratégie gagnante. Elle prend en paramètres a, qui est une liste des nombres d'allumettes dans chaque pile, et r, qui est une liste des nombres d'allumettes que le joueur peut enlever à chaque tour. Elle utilise la fonction valPileAllumettes pour calculer la valeur de chaque pile d'allumettes et utilise la fonction sumNimXOR pour calculer la somme XOR des valeurs de toutes les piles.

- Fonctions impliquées dans cet algorithme

La fonction afficheJoueurStratGagnanteAllumettes(a, r) affiche le choix du joueur selon la stratégie gagnante dans le jeu d'allumettes. Elle prend en paramètres a, qui est une liste des nombres d'allumettes, et r, qui est une liste des nombres d'allumettes que le joueur peut enlever à chaque tour. Elle itère sur chaque pile d'allumettes et chaque nombre d'allumettes pouvant être enlevé, et vérifie si le jeu d'allumettes devient égal à zéro après avoir enlevé un certain nombre d'allumettes. Si c'est le cas, elle retourne ce nombre d'allumettes. Sinon, si la valeur du jeu d'allumettes est différente de zéro, elle choisit un nombre aléatoire parmi les possibilités r.

La fonction ordi(nbAllumettes, r) est utilisée pour simuler le tour de l'ordinateur dans le jeu d'allumettes. Elle prend en paramètres nbAllumettes, qui représente le nombre d'allumettes restantes, et r, qui est une liste des nombres d'allumettes que le joueur peut enlever à chaque tour. Elle appelle la fonction strategie_gagnante_ordi pour obtenir le nombre d'allumettes à enlever, puis met à jour le nombre d'allumettes restantes en soustrayant le nombre enlevé. Elle retourne ensuite le nouveau nombre d'allumettes et le nombre enlevé.

La fonction strategie_gagnante_ordi(nbAllumettes, liste_jeu) est utilisée pour déterminer la stratégie gagnante de l'ordinateur dans le jeu d'allumettes.

Ce code représentait le déroulement d'un tour de jeu dans une partie d'allumettes, où le joueur et l'ordinateur s'affrontent.

L'instruction if valJeuAllumettes(liste_allumettes,r)==0 vérifie si la valeur du jeu d'allumettes est égale à zéro en appelant la fonction valJeuAllumettes avec les piles d'allumettes liste_allumettes pour savoir si le deuxième joueur a la stratégie gagnante et les possibilités de retrait r. Si c'est le cas, cela signifie que c'est au tour du joueur de jouer en premier.

Dans ce cas, la boucle while nbAllumettes>0 : est utilisée pour continuer le jeu tant qu'il reste des allumettes. À chaque itération de la boucle, le joueur et l'ordinateur effectuent leurs tours de jeu.

- Dans la première branche de la condition (if nbAllumettes-min(r)>=0:), le joueur effectue son tour. Peu importe le choix du joueur, l'ordinateur ferra le bon choix au deuxième tour de jeu afin de gagner.

- Dans la deuxième branche de la condition (else:), c'est au tour de l'ordinateur de jouer en premier. Le déroulement est similaire à la première branche, mais les rôles du joueur et de l'ordinateur sont inversés.

Dans les deux cas, l'ordinateur gagne tout le temps.

- Fonctions modifiées pour améliorer la lisibilité du code

Nous avons restructuré notre code en trois fichiers, au lieu d'un, afin d'améliorer la visibilité, et pour séparer le code du turtle avec celui du programme même. Le fichier allumettes comportent le code récursif et celui du joueur et de l'ordinateur. Le fichier Map comporte le code du turtle qui permet d'afficher dans la fenêtre la partie graphique du code. Le fichier main.py permet de lancer le programme.

Conclusion

La bonne organisation de notre code au premier semestre nous a permis de corriger, rapidement, les erreurs que nous n'avions pas corrigées lors de ce premier semestre. La reprise faite, nous avons pu nous concentrer sur l'objectif attendu lors de la reprise de note projet, qui est, l'implémentation d'une stratégie gagnante pour l'ordinateur peu importe la règle et le nombre de départ d'allumettes. Nous espérons pouvoir régler les quelques détails restés du premier semestre comme le nom de la fenêtre du numinput(). Nous pouvons aussi essayer de ne pas avoir un if et else pour valJeuAllumettes .