Dans ce projet nous allons jouer à un jeu tiré du concours castor (2021)
Voici les règles :

- A chaque mouvement, le castor se déplace sur une case adjacente à celle qu'il occupe actuellement selon les 4 directions habituelles : haut, bas, droite gauche.
- Une case ne peut être empruntée qu'une seule fois tout au long du parcours.
- Le castor ne peut pas aller sur une case du même type que celle qu'il occupe actuellement. Mais il a toutefois le droit à un joker qui lui permet de pas respecter cette règle une seule fois sur tout son parcours.
- Le but du castor est de pouvoir rejoindre la case qui comprend le drapeau sur la grille.

Pour l'interface graphique nous utiliserons la bibliothèque Tkinter de Python.
Dans le fichier de travail interfacecastor.py, une interface graphique comprenant une première grille a été créée. Cette grille comprend des boutons actifs. Cela devrait vous permettre de découvrir (un peu) Tkinter. Nous ne représenterons pas le castor ni le drapeau sur la case d'arrivée.

I. Première partie : Une grille exemple

Dans cette partie on considère une seule grille de taille fixe 5×6 , celle qui est donnée ci dessus. C'est elle qui sert aussi de modèle dans votre fichier de travail. On considère ici que les points de départ et d'arrivée sont toujours les mêmes. Plus généralement même si les grilles changent de taille, le départ sera la case en bas à droite et l'arrivée en haut à gauche.

a) Points clés des fichiers de travail

- Vous avez deux fichiers à votre disposition. interfacecastortravail.py qui s'occupe de l'interaction avec le joueur. Le fichier jeucastortravail.py comprend quant à lui les autres fonctions (de validation, recherche etc...)
- Une grille est une liste de liste qui comprend des "P" pour pierre, des "B" pour bois et "F" pour feuille. Les coordonnées sont données sous la forme Ligne colonne : ainsi le drapeau est en sur la case [0,0] et le départ du castor en [4,5].
- Le début du programme permet de créer l'interface pour une grille donnée. Chaque case est cliquable. Tous les boutons sont stockés dans la variable tableauBoutons (dans lequel un bouton donné est repéré par ses coordonnées) Elle crée aussi un bouton "j'ai fini" qui permet de dire lorsque votre parcours est terminé. Au fur et à mesure des clics les cases sont désactivées et, à la fin, le chemin parcouru est mémorisé sous forme d'une liste de liste.

b) Rappel des règles

Un parcours est valide si les 3 conditions suivantes sont remplies :

- Le parcours débute et termine sur les bonnes cases ici [4,5] et [0,0]
- Toutes les cases lors du parcours sont adjacentes

— Toutes les cases du parcours respectent la règle d'alternance : pas deux cases identiques à la suite sauf une seule fois (utilisation du joker)

c) A faire

Vous devez donc écrire 3 fonctions qui se chargent de vérifier si ces règles sont remplies. Une quatrième indiquera au joueur s'il a gagné ou perdu via un message sur l'interface.

— departArrivee(parcours) vérifie la validité des points de départ et d'arrivée.

— cheminContinu(parcours) vérifie l'adjacence des cases lors du parcours. Cette fonction fera appel à la fonction sontVoisines(case1,case2) qui vérifie si deux cases sont adjacentes

— ordreDesCases(parcours) se charge de vérifier la règle d'alternance des cases (en tenant compte du joker)

— enfin la fonction verification(parcours) se charge de d'annoncer si le chemin est correct ou non via l'interface et affiche "Bravo " ou "Perdu".

II. Un peu d'aléatoire

Dans cette partie on va s'intéresser à des chemins aléatoires empruntés par le castor.

a) Le castor progresse toujours vers son objectif

Dans un premier temps, à chaque étape le castor, pressé d'arriver, avance toujours soit vers le haut, soit vers la gauche afin de pas perdre de temps en route
Vous devez écrire une fonction cheminAleatoire1(grille) qui crée et renvoie un chemin aléatoire entre les points de départ . Du côté interface un bouton déclenchera l'appel de cette fonction ainsi que la vérification de la validité du chemin créé.

b) Le castor se déplace sur une case disponible autour de lui

Maintenant le castor choisit une case au hasard adjacente disponible (donc non déjà visitée). Il risque toutefois d'être de rester coincé et de ne jamais atteindre son but!
Vous devez ici écrire une fonction cheminAleatoire2(grille) qui crée et renvoie un chemin aléatoire entre les points de départ et d'arrivée ou tout autre point s'il reste coincé. Désorienté, mais pas complètement fou, s'il est à une case de l'arrivée, il privilégie ce mouvement. Du côté interface un bouton déclenchera l'appel de cette fonction ainsi que la vérification de la validité du chemin créé.

c) Le castor s'ennuie...

Un peu fatigué d'explorer toujours la même grille, notre castor veut des nouvelles grilles. Il vous faut donc écrire une fonction randomGrille qui génère une nouvelle grille à taille variable (par défaut de 5×6). On créera aussi une fonction initialisenew qui se chargera d'effacer l'affichage précédent de changer la grille courante. Cette fonction sera appelée par un bouton "nouvelle grille"

III. Recherche d'un chemin : plus difficile!

Notre pauvre castor semble un peu perdu.
Ecrire une fonction qui recherche et permet de proposer un chemin valide s'il existe ou à défaut de dire s'il n'en n'existe pas. Attention on ne veut pas ici de fonction récursive (c'est à dire qui se rappelle elle même)
Aide : le principe est de créer un chemin valide pas à pas jusqu'a éventuellement être bloqué et rebrousser chemin. Pour cette question vous fournirez un schéma qui indique la méthode retenue pour explorer votre grille. L'idéal serait de présenter une grille et sa résolution à la main.

IV. Améliorations possibles

Voici quelques améliorations possibles pour votre projet :

— Ajouter des boutons (type scale) pour gérer la taille des grilles que l'on veut générer.

— Ajouter un bouton qui permet d'annuler le dernier déplacement (lorsque l'on cherche à trouver soi même un chemin)

— Montrer le castor qui se déplace sur la grille.
— Annuler le parcours proposé et en proposé un nouveau.
Cette liste n'est bien sûr pas limitative.

V. Conseils

Pour mener à bien ce projet, vous veillerez à bien utiliser les deux fichiers proposés (que vous pourrez renommer en supprimant le suffixe travail) L'intérêt du fichier jeucastortravail.py est de pouvoir tester vos fonctions sans l'interface. Vous rendrez un rapport en pdf du même type que celui du projet 1
Un organigramme des fonctions est demandé.
Bien évidemment votre code sera correctement commenté et les fonctions documentées.