M1102: Introduction à l'algorithmique et à la programmation
Projet n°1
La Rivière de l'IUT'O

1 Présentation

L'objectif de ce projet est de programmer un jeu inspiré de *la Rivière d'enfer* `http://jeuxsoc.fr/?principal=/jeu/rivie?` de Firedman Friese. Le but de ce jeu est de descendre la rivière de l'IUT'O sur un surf aux couleurs d'un département de l'IUT'O en se jouant des obstacles et des courants. La rivière est représentée par une grille hexagonale où chaque case peut être soit un rocher, soit de l'eau. Les cases *rocher* sont inaccessibles aux surfeurs. Les cases *eau* peuvent éventuellement contenir des troncs d'arbre, qui sont des obstacles que l'on peut pousser avec son surf. Une case *eau* peut aussi avoir un courant qui pousse les objets (surfeurs ou troncs) qui s'arrêtent dessus. La descente se fait à partir d'une case départ qui se trouve sur la première ligne de la rivière et se termine sur une case arrivée qui se trouve sur la dernière ligne de la grille. Au démarrage, aucun surfeur n'est sur la grille, le joueur qui s'élance se place sur la case de départ.

À chaque tour de jeu, le joueur courant obtient un nombre (aléatoire) de déplacements compris entre 1 et 5. Le joueur n'est pas obligé d'effectuer tous ses déplacements. Pour chaque déplacement, le joueur choisit une direction. Si le joueur a moins de 2 obstacles devant lui dans cette direction, il pousse ces obstacles en se déplaçant. Si le surfeur a un rocher ou plus de deux obstacles devant lui, le déplacement n'est pas possible. **Attention !** Les bords de la rivière sont considérés comme des rochers. Une fois que le joueur a fini ses déplacements, les objets qui se trouvent sur des cases où il y a du courant se déplacent en suivant la direction de ce courant. Le vainqueur de la partie est le premier à atteindre la case d'arrivée. Le joueur arrivé sort du jeu et les autres doivent terminer la descente pour se départager. Une fois que tous les joueurs sont sortis, la partie est terminée. **Attention !** Si un tronc se retrouve sur la case de départ ou sur la case d'arrivée, celui-ci disparaît. Si un joueur se retrouve sur la case départ il ressort du jeu et au prochain tour, il reprendra le départ. À n'importe quel moment du jeu, il y a au plus un objet (surfeur, tronc ou rocher) sur une case.

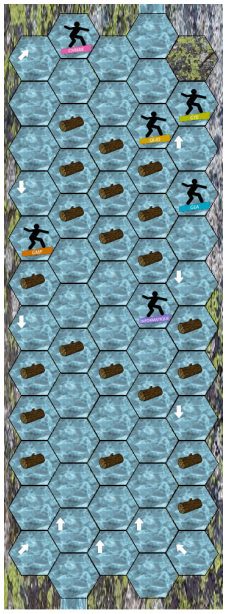


Figure 1 – La rivière de l'IUT'O

2 Quelques détails sur le jeu

2.1 La grille

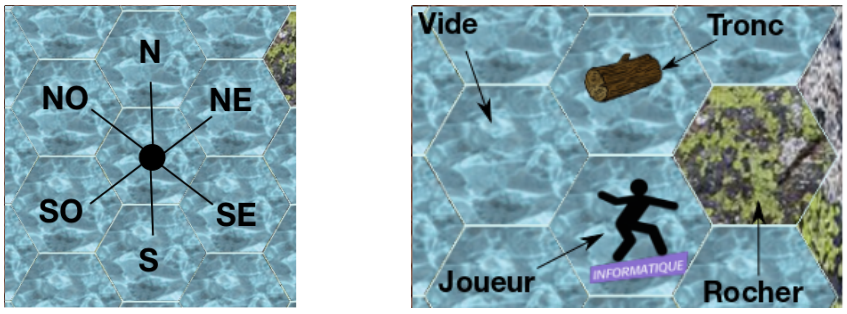
La grille est composée de cases hexagonales (voir figure 1). Dans ce type de grilles, chaque ligne ne comporte qu'une colonne sur deux. Sur la figure 1, la ligne 0 (ainsi que toutes les lignes de numéro pair) comporte colonne 1 et 3 et la ligne 1 (ainsi que toutes les lignes de numéro impair) comporte les colonnes 0, 2, 4. Nous avons deux cas de figures possibles :

- Soit les lignes paires contiennent les colonnes paires et les lignes impaires contiennent les colonnes impaires. Dans ce cas on dit que la grille est *paire* (figure 3).
- Soit les lignes impaires contiennent les colonnes paires et les lignes paires contiennent les colonnes impaires. Dans ce cas on dit que la grille est *impaire* (figure 1).

Pour une rivière, la grille aura une colonne de départ (sur la première ligne) par où les joueurs commencent la descente, et une colonne d'arrivée (sur la dernière ligne) par où les joueurs sortent de la rivière. Ces cases sont matérialisées sur les figures 1 et 3 par des filets d'eau.

2.2 Les cases et les directions

La figure 2(a) montre une case (celle avec un point noir) et la manière dont sont codées les directions pour accéder à ses cases voisines. On peut remarquer que pour aller vers le nord (N) ou le sud (S), on reste sur la même colonne mais on passe deux lignes. Pour les quatre autres directions on passe une colonne **ET** une ligne.
Une case peut contenir un rocher, un tronc d'arbre, un joueur ou rien (dans ce cas là c'est de l'eau) comme illustré figure 2(b). Les rochers sont inamovibles par contre les troncs d'arbre et les joueurs peuvent bouger.



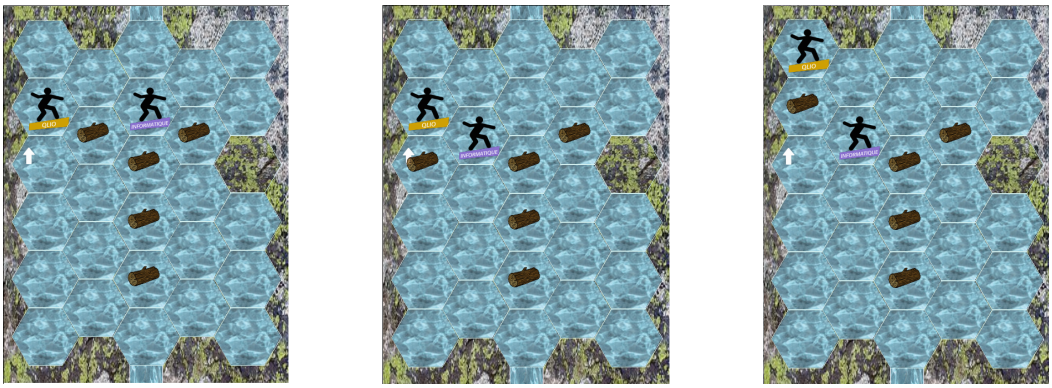
(a) Une case et ses voisines (b) Les objets sur les cases

FIGURE 2 – Les cases

Enfin une case qui n'est pas un rocher peut contenir un courant. Ce courant a une direction qui indique vers quelle cellule voisine un objet qui se trouve dans cette case sera dirigé. Sauf s'ils sont bloqués par des obstacles, les objets ne peuvent pas rester sur une case qui contient du courant car ils sont emportés par le courant vers la case voisine indiquée.

2.3 Les déplacements

Au départ, aucun joueur n'est sur le plateau. Lorsqu'il s'élance, un joueur est placé sur la case de départ. À chaque tour de jeu un joueur a entre 1 et 5 déplacements possibles. Ce nombre est tiré au sort. Chaque déplacement consiste à passer d'une case à une case voisine en indiquant la direction. Par exemple, en partant de l'état illustré figure 3(a), si le joueur Informatique décide d'aller vers le sud-ouest (SO), il va pousser le tronc d'arbre qui s'y trouve. Ce qui nous amène à la situation illustrée 3(b).



(a) C'est au joueur Informatique de joueur

(b) Il a choisi SO et pousse le tronc

(c) En fin de tour le tronc est poussé par le courant

FIGURE 3 – Exemple de déplacement

La direction X représente le fait que le joueur décide de rester sur place (et donc renonce au reste de ses déplacements). Une fois qu'un joueur a effectué tous ses déplacements, les objets qui se trouvent sur une case où il y a du courant sont déplacés dans la direction de ce courant et en suivant les mêmes règles qu'un déplacement de joueur. Ainsi, dans la situation illustrée figure 3(b), le tronc d'arbre qui avait été poussé par le joueur Informatique, se retrouve à nouveau poussé par le courant vers le nord. Lui-même va pousser le joueur QLIO vers le nord. On arrive ainsi dans le situation illustrée figure 3(c).
Il existe trois restrictions aux déplacements :

1. Un joueur ou un tronc ne peut pas aller en dehors de la rivière.
2. Un joueur ou un tronc ne peut pas se déplacer sur une case qui contient un rocher. Par exemple sur la figure 3(a), le joueur Informatique ne peut pas aller vers le sud-est (SE) car pour cela il devrait pousser le tronc vers le sud-est mais celui-ci est bloqué par un rocher.
3. Un joueur ou un tronc ne peut pas se déplacer dans une direction où il y a plus de deux objets amovibles devant lui. Par exemple sur la figure 3(a), le joueur Informatique ne peut pas aller vers le sud car il y a trois troncs d'arbre dans cette direction.

À n'importe quel moment du jeu, si un joueur ou un tronc se trouve sur la case arrivée, celui-ci sort de la rivière. Si c'est un joueur, il est alors ajouté au classement.
Enfin, lorsqu'un tronc d'arbre ou un joueur se retrouve sur la case départ à la fin d'un tour de jeu, celui-ci sort de la rivière. Si c'est un tronc d'arbre, cette sortie est définitive, si c'est un joueur celui retournera sur la case départ lors de son prochain tour.

3 Travail à faire

Vous devrez développer, par groupe de trois personnes, un programme qui permet à des surfeurs (entre 2 et 6) de descendre la rivière de l'IUT'O. Le structuration du projet vous est imposée. Elle s'organise en plusieurs grandes parties qui vous permettront de proche en proche d'arriver à l'implémentation finale du jeu. Pour chaque structure de données, vous êtes en grande partie libre de la représentation mémoire de cette structure par contre l'interface de programmation vous est imposée. Si vous respectez bien les spécifications, vous pourrez utiliser les scripts permettant de jouer en mode texte et en mode graphique à la rivière de l'IUT'O.

Nous allons décrire les différentes structures de données que vous aurez à implémenter dans les fichiers Python qui vous sont fournis. Les spécifications plus précises de chaque fonction sont données en commentaire dans les fichiers Python.

3.1 case.py

Comme décrit dans la section 2.2 les cases peuvent contenir un rocher, un tronc, un joueur ou rien. Par ailleurs, elle peuvent avoir un courant. L'API doit vous permettre de gérer une case de la rivière. Le dictionnaire `directions` permet d'associer à chaque direction un caractère en forme de flèche qui sera utilisé notamment pour le jeu en mode texte. La direction 'X' indiquant aucun déplacement, aucune flèche n'y est associée. Trois constantes permettent de coder les rochers, les troncs et le vide. Les joueurs seront représentés par un caractère différent des trois constantes.

3.2 grille.py

C'est la structure qui permet de gérer une grille hexagonale sans préjuger du type de données qu'elle va contenir. Une telle grille a un nombre de lignes, un nombre de colonnes et un type (paire ou impaire). Voir section 2.1 pour les détails de la numérotation des cases de la grille. La fonction `afficheGH` vous permet d'afficher le contenu d'une grille. La fonction `initAlphaGH` permet d'initialiser une grille avec des caractères ce qui vous permet de tester vos fonctions sur cette structure de données.

3.3 joueursPossibles.py

Cette structure permet de gérer la liste des joueurs qui pourront participer à la course c'est-à-dire l'association entre le nom des joueurs (utilisé par le mode graphique pour connaitre l'image associée à chaque joueur) et sa représentation dans la grille (sous la forme d'un caractère). La liste des joueurs possibles pourra être chargée à partir d'un fichier dont un exemple vous est donné dans le fichier `data/joueurs.txt`.
Attention ! Pour le mode graphique, il faut pour chaque joueur possible qu'un fichier de nom `nomjoueur.png` existe dans le répertoire `images` pour que le jeu fonctionne. Pour la rivière de l'IUT'O les joueurs connus sont ceux du fichier `data/joueurs.txt` et la représentation graphique de ces joueurs sont dans le répertoire `images`.

3.4 joueur.py

Cette structure permet de stocker un joueur qui contient trois propriétés : son nom, sa représentation et un booléen indiquant si le joueur est humain ou automatique.

3.5 listeJoueurs.py

Cette API doit gérer une liste de joueurs. La structure `Joueurs` permet donc de gérer une liste de joueurs engagés dans la course mais aussi le classement. Cette liste est agrémentée d'un joueur courant qui permet de savoir lequel des joueurs doit se déplacer. L'API va donc permettre notamment d'ajouter et d'enlever des joueurs d'une liste de joueurs et de passer au joueur suivant.

3.6 riviere.py

Cette structure de données va gérer la rivière : c'est-à-dire une grille hexagonale contenant des cases telles que définies plus haut. Une rivière possède en plus une colonne de départ sur la première ligne et une colonne d'arrivée sur la dernière ligne. Les fonctions qui sont à implémenter permettent de gérer les déplacements sur la rivière.
La fonction `afficheRiviere` permet d'afficher la rivière en mode texte. **Attention !** Cette fonction ne marche pas dans le terminal Wing, il faut l'exécuter dans un terminal.
La fonction `lireRiviere` permet d'initialiser une rivière à partir d'un fichier texte dont un exemple vous est donné dans le fichier `data/riviere.txt`.

3.7 jeu.py

Cette structure de données va gérer l'ensemble du jeu, c'est-à-dire la rivière, les joueurs et le classement. Les deux principales fonctions sont `jouerDirection` et `finirDeplacement` qui vont gérer les déplacements sur la rivière en fonction des choix des joueurs.
La première fonction permet de gérer le déplacement du joueur courant en fonction de son choix. Cette fonction va vérifier que la direction demandée est bien valide et si c'est le cas, va effectuer le déplacement demandé (ainsi que toutes ses conséquences). Le résultat de la fonction est une chaîne de caractères qui rend compte de ce qui s'est passé. Cette chaîne de caractères sera utilisée par les scripts `riviereGraphique` et `riviereTexte`. Les différentes chaînes possibles sont :

- `"le joueur xxx renonce à ses derniers déplacements"` si la direction choisie est `'X'`
- `"Le tour se termine car le joueur xxx est arrivé"` si le déplacement choisi engendre l'arrivée du joueur courant sur la case arrivée.
- `"Le joueur xxx est arrivé"` si le déplacement choisi engendre l'arrivée d'un joueur autre que le joueur courant. Cela arrive quand le joueur est poussé sur la case arrivée lors du déplacement.
- `"Attention direction incorrecte"` si la direction choisie n'existe pas ou si le déplacement est impossible à cause des contraintes posées dans la section 2.3
- `""` dans tous les autres cas ; c'est-à-dire que le déplacement n'engendre pas d'événement particulier.

La fonction `finirDeplacement` quant à elle permet de gérer la fin du tour du joueur courant c'est-à-dire rechercher si un objet autre qu'un rocher se trouve sur une case où il y a du courant et si c'est le cas, effectue le déplacement associé. Si plusieurs déplacements sont possibles, la fonction n'effectue qu'un seul de ces déplacements pour que le programme d'affichage puisse montrer tous les états intermédiaires de la rivière. L'ordre dans lequel s'effectue les déplacements est le suivant :
On commence par les objets qui ont le moins d'obstacles devant eux. Si plusieurs objets ont le nombre minimum d'obstacles devant eux, on commence par celui que l'on rencontre le premier en parcourant la grille du nord vers le sud et d'ouest en est.
Le résultat de la fonction est un couple comprenant un booléen permettant de savoir si un déplacement a effectivement eu lieu ou non et une chaîne de caractères destinée aux programmes principaux. Si le booléen retourné est `True` cela veut dire que le tour est terminé (aucun déplacement n'a pu avoir lieu), sinon il est à `False`. Les messages de retour possibles sont les suivants :

- `"La partie est terminée"` si aucun déplacement n'a pu se faire et que tous les joueurs sont arrivés.

— `"Le joueur xxx est arrivé"` si il y a eu un déplacement qui a provoqué l'arrivée d'un joueur
— `""` dans tous les autres cas.

Il faut noter que si aucun déplacement n'a pu se faire, cela veut dire que le tour du joueur courant est terminé et donc la fonction devra faire passer au joueur suivant.

3.8 ia.py

Ce fichier permet la gestion de l'intelligence artificielle des joueurs automatiques. Il vous est donner une fonction de choix de direction qui retourne une direction aléatoire (ce qui n'est pas très efficace). Vous aurez à implémenter une fonction de marquage permettant de connaitre la distance minimum pour atteindre une certaine case de la rivière. Cette fonction devrez vous permettre d'améliorer l'IA proposée. Vous pourrez évidemment proposer votre propre IA.

3.9 riviereModeTexte.py

Vous n'avez rien à implémenter dans ce fichier. Cependant voici comment se structure le programme principal qui permet de jouer à la rivière de l'IUT'O
La fonction `jouer` permet simplement d'effectuer plusieurs parties sans avoir à relancer le programme. À la fin de chaque partie, on demande aux utilisateurs s'ils veulent refaire une partie ou non.
La fonction `jouerUnePartie` permet de gérer une partie. Après une phase d'initialisation et un premier affichage du jeu, la boucle principale de la fonction est lancée. Cette boucle a deux phases :

1. Faire une action sur le jeu
2. Afficher le jeu

Trois actions sont possibles en fonction de l'état du jeu :

1. Demander au joueur courant la direction de son déplacement
2. Effectuer le déplacement demandé par le joueur courant
3. Finir le tour du joueur courant

Pour savoir quelle action effectuer, la fonction utilise un booléen `attenteDirection`. Lorsque ce booléen est à `True`, on sait que l'action à effectuer est l'action 1. S'il est à `False`, alors
— si le nombre de coups restants du joueur courant est à 0, on effectue l'action 3. Le code retour de la fonction `finirDeplacement` permettra de remettre le booléen `attenteDirection` à `True` le moment venu.
— sinon il faut jouer le déplacement choisi par l'utilisateur lors du tour de boucle précédent. Si le joueur n'a plus de tour à jouer on laisse le booléen `attenteDirection` à `False` pour permettre d'appeler la fonction `finirDeplacement` au tour de boucle suivant.

3.10 riviereModeGraphique.py

Le principe est pratiquement le même que le mode texte sauf que l'environnement graphique a sa propre boucle d'interaction ce qui fait que les appels aux différentes fonctions sont gérés un peu différemment.

4 Rendu

Les groupes de projets doivent être constitués de **deux ou trois personnes du même groupe de TD**. Les groupes de projets seront formés par l'équipe enseignante.
Le rendu final se fera sur l'ENT dans une archive `zip` portant le nom des développeurs du programme.
Vous aurez à rendre

- DEUX versions de vos programmes correspondant aux deux répertoires `versionClassique` et `versionObjet` fournis avec le sujet.
  1. La première version implémente et utilise les API fournies dans le répertoire `versionClassique`.
  2. La seconde version implémente les structures de données sous la forme de classes. Les méthodes des classes devront porter le nom des fonctions correspondantes dans la version non objet. Des précisions vous seront données après les vacances sur cette version.
- Un petit dossier de programmation au format `pdf` indiquant votre répartition du travail, les méthodes de tests adoptées, le choix des structures de données utilisées, les principaux algorithmes que vous avez implémentés, les bugs connus de votre programme et les extensions que vous avez apportées. En annexe vous devrez insérer pour chaque participant une fiche individuelle indiquant le travail effectué au jour le jour.

Ce rendu se fera sur Celene le **19 janvier 2017** avant 23h55. Le **20 janvier** sera organisée une soutenance de 15 minutes au cours de laquelle vous présenterez votre travail et ferez une petite démonstration.