Projet de programmation du premier semestre

À lire attentivement et obligatoirement

Le contrôle des connaissances du module AP1 comprend la réalisation d'un projet de programmation. Ce projet doit être réalisé par des **binômes** d'étudiants du **même groupe de TP**, sauf permission expresse accordée par un enseignant. Le travail demandé consiste à écrire un **programme** en Python 3, à rédiger un bref **rapport** et à présenter le travail lors d'une **soutenance**.

Programme. Le programme réalisé doit être écrit en Python 3 et fonctionner sur les machines de l'université. Les points suivants sont primordiaux lors de la réalisation du projet :

- **modularité :** le programme doit être structuré à l'aide de fonctions ;
- **bonne utilisation des variables :** choisir des noms pertinents et explicites pour les identificateurs et les fonctions ;
- **documentation :** utiliser des chaînes de documentation (*docstrings*) pour décrire le rôle de chaque fonction, les valeurs attendues des paramètres et la valeur renvoyée ;
- **tests :** partout où c'est utile, les fonctions doivent contenir des tests intégrés à leur chaîne de commentaire (*doctests*) ;
- **commentaires :** commenter les points importants ou difficiles du programme pour en aider la compréhension.

Rapport. Le rapport (10 pages maximum tout compris) ne doit pas consister en une simple description du code. Il doit contenir au minimum les rubriques suivantes :

- Présentation du projet et manuel utilisateur ;
- Structure générale du programme, principales fonctions et leur rôle ;
- Principales variables utilisées pour représenter les données du jeu (plateau, position des joueurs ou des pions, positions des obstacles...), leur type et leur signification ;
- Organisation et répartition du travail au sein du binôme (découpage du travail, fréquence des rencontres, difficultés rencontrées, apprentissages réalisés...).

Soutenance. À la suite du rendu du projet, une soutenance de 20mn aura lieu. Celle-ci donnera l'occasion aux étudiants du binôme de présenter leur travail pendant une dizaine de minutes, puis chaque étudiant sera soumis à des questions permettant de vérifier son implication et sa compréhension.

Choix du sujet. Les pages qui suivent décrivent plusieurs sujets. Chaque binôme doit choisir un sujet à traiter. En cas d'incompréhension d'un énoncé, postez un message sur le forum d'entraide décrivant **précisément** votre problème, ou envoyez un email à l'un de vos enseignants d'AP1.

Exceptionnellement, un sujet personnel ne figurant pas dans la liste ci-dessous peut-être proposé à l'équipe enseignante, à condition de le rédiger au préalable de manière convenable (sous une forme semblable aux sujets qui suivent). Les enseignants se réservent le droit de refuser toute proposition jugée inadaptée (sujet trop facile, trop difficile ou mal conçu).

Table des matières

1 Rasende Roboter (Ricochet Robot)

Selon Wikipedia :

> Ricochet Robots (Rasende Roboter pour la première édition en allemand) est un jeu de société créé par Alex Randolph et illustré par Franz Vohwinkel, édité en 1999 par Hans im Glück / Tilsit.
>
> Le jeu est composé d'un plateau, de tuiles représentant chacune une des cases du plateau, et de pions appelés « robots ». La partie est décomposée en tours de jeu, un tour consistant à déplacer les robots sur un plateau afin d'en amener un sur l'une des cases du plateau. Les robots se déplacent en ligne droite et avancent toujours jusqu'au premier mur qu'ils rencontrent.

FIG. 1: Le plateau de jeu de Rasende Roboter.

L'objectif de ce projet est de réaliser une version programmée la plus fidèle possible de ce jeu de société. Le jeu devrait être jouable par plusieurs personnes assises devant le même écran.

Principe et règles du jeu

Le texte suivant est adapté d'une traduction des règles officielles du jeu par l'auteur du site `http://jeuxsoc.free.fr`.

Éléments du jeu

- un plateau de jeu imprimé de 16 cases de côté (voir la photo ci-dessus)
- 4 robots de couleurs différentes
- 4 jetons carrés aux couleurs des robots (servant à mémoriser leur position de départ)
- 17 jetons de cible ronds (4 de chacune des 4 couleurs, et une multicolore, correspondant chacun à une case marquée sur le plateau)
- 1 sablier (environ 1mn)

Préparatifs

- Posez au hasard les quatre jetons colorés sur des cases libres du plateau. Placez les robots sur les carrés de leurs couleurs respectives.
- Mélangez les 17 cibles faces cachées sur la table.
- Retournez une cible au hasard et placez-la face visible. Le jeu peut commencer.

But du jeu

Lors de chaque tour, le but est de remporter le jeton cible qui a été tiré au sort. Sur le plateau, il existe une case portant le même symbole et la même couleur que le jeton. Votre but est de trouver comment le robot de même couleur, dit robot actif, peut atteindre cette case cible en le moins de déplacements possible. Il est autorisé de déplacer aussi les robots d'une autre couleur pour servir de point d'appui au robot devant atteindre la cible. Le joueur qui trouve la suite de mouvements la plus courte gagne le jeton. Celui qui gagne le plus de jetons gagne la partie !

Planification des mouvements

Au début d'un tour, les robots ne bougent que dans les cerveaux des joueurs. Chaque joueur essaye de découvrir le moyen le plus économe en mouvements pour amener le robot actif sur la cible. Les robots se déplacent horizontalement ou verticalement, sans tourner, selon les directives des joueurs, mais une fois en mouvement, ils ne peuvent plus s'arrêter jusqu'à ce qu'ils rencontrent un obstacle devant lequel ils s'arrêtent. Les obstacles sont les bords du plateau, les murs dessinés entre certaines cases (voir photo), la pièce centrale et les autres robots. Lorsqu'un robot heurte un obstacle, il peut s'arrêter ou ricocher à angle droit, à droite ou à gauche, jusqu'à ce qu'il heurte un nouvel obstacle, et ainsi de suite. Chaque déplacement d'un robot jusqu'à un obstacle compte comme un mouvement.

Déroulement d'un tour

- Un tour commence lorsqu'un joueur découvre un jeton pris au hasard parmi les jetons restants.
- Chaque joueur cherche alors comment le robot actif (le robot de la même couleur que la cible) peut atteindre la cible avec le moins de mouvements possible. La plupart du temps, un ou plusieurs autres robots devront être déplacés pour libérer un chemin ou pour former obstacle ; ces mouvements comptent évidemment dans le décompte total des mouvements.
- Si le jeton à atteindre est le « Vortex cosmique » qui est de toutes les couleurs, n'importe quel robot peut l'atteindre.
- Pour atteindre la cible, le robot actif doit ricocher au moins une fois (tourner à droite ou à gauche) sur un obstacle. S'il peut atteindre la cible sans ricocher, il faut trouver une autre route.
- Dès qu'un joueur a trouvé une solution, il peut annoncer le nombre de mouvements qu'il pense être le bon. Il retourne alors le sablier. Les autres joueurs ont maintenant une minute pour trouver une solution plus économe en mouvements et annoncer un nouveau nombre. Il n'y a pas d'ordre pour annoncer, et chacun peut annoncer plusieurs fois. Les annonces successives sont généralement plus faibles, mais peuvent également être de même hauteur, voire plus élevées (par exemple quand un autre joueur a annoncé un total improbable).

- Quand un joueur a fait une annonce, il ne peut la changer pour une annonce plus forte. Un autre joueur seul peut le faire.
- Quand le sablier se termine, le joueur qui a fait la plus petite annonce joue. Il bouge les robots en comptant les mouvements à haute voix. Si le nombre de mouvements est égal ou plus petit que le nombre annoncé, il prend le jeton et le tour est terminé. S'il échoue, il remet les robots sur leur case de départ (les jetons carrés de couleur servent de témoin) et la main passe au joueur qui a l'annonce suivante la plus petite. En cas d'égalité, le joueur ayant gagné jusqu'à présent le moins de jetons prend la main. On continue ainsi jusqu'à ce qu'un joueur démontre sa solution. Si personne n'y parvient, le jeton est retourné et mélangé avec les jetons qui n'ont pas encore été tirés.
- Quand un tour est terminé, placez les jetons carrés de couleur sous les robots respectifs afin de mémoriser leur nouvelle position, tirez une nouvelle cible, etc. La partie se termine dès que tous les jetons sont gagnés.

Implémentation

On donne volontairement peu d'indications sur la méthode d'implémentation. La principale consigne est de chercher à respecter le plus fidèlement possible les règles officielles rappelées ci-dessus. La convivialité de l'interface et la jouabilité seront également appréciées.

Représentation du plateau de jeu

On cherchera à reproduire aussi fidèlement que possible la structure du plateau de jeu telle qu'elle apparaît sur la photo ci-dessus. Les informations à mémoriser sont les suivantes :

- Position des murs (entre des cases, au bord du plateau et autour du carré central) ;
- Position de chaque robot ;
- Position des jetons marquant l'emplacement précédent de chaque robot ;
- Position de chacun des 17 symboles de cible (4 formes de 4 couleurs, plus une cible "multicolore") ;
- Cases interdites (carré 2×2 central).

Chacune de ces informations peut par exemple être mémorisée dans une liste contenant des données du type approprié.

Gestion de la partie

On suggère le fonctionnement suivant pour le contrôle de la partie. Dès qu'un joueur pense avoir trouvé une solution, il clique sur un bouton (ou appuie sur une touche du clavier) indiquant son numéro de joueur puis saisit un nombre de mouvements, ce qui déclenche un chronomètre d'une minute. Pendant cette minute, les autres joueurs peuvent à leur tour saisir un nombre de mouvements (inférieur, mais aussi éventuellement égal ou même supérieur comme indiqué dans les règles). Un même joueur ne peut cependant immédiatement changer son annonce pour une annonce supérieure.

À l'issue de la minute de réflexion, les annonces sont vérifiées selon le mécanisme décrit au paragraphe suivant, par ordre croissant. La conception des mécanismes exacts de gestion des différents joueurs et de tenue du score est laissée à la réflexion des programmeurs.

Vérification des annonces

À l'issue de la minute de réflexion, comme cela est détaillé dans les règles, le joueur ayant proposé le plus petit nombre de mouvements doit démontrer sa solution grâce à la souris ou au clavier. Deux mécanismes de saisie de la suite de mouvements sont proposés, mais d'autres solutions peuvent être adoptées :

- Le joueur appuie sur une touche correspondant à la couleur d'un robot (par exemple R, V, B, J), puis indique une suite de mouvements (nord, sud, est ou ouest) à l'aide des flèches du clavier. Pour sélectionner un autre robot, il peut entrer à nouveau une lettre de couleur, et ainsi de suite.
- Le joueur sélectionne un robot en cliquant dessus, puis clique sur une case adjacente pour indiquer la direction du prochain déplacement. Le robot reste sélectionné tant que la cible n'est pas atteinte ou qu'un autre robot n'est pas sélectionné.

Idéalement, les deux mécanismes de saisie peuvent être implémentés. Quelle que soit la solution choisie, chaque mouvement doit être représenté sur le plateau au fur et à mesure, un compteur doit indiquer le nombre de mouvements déjà effectués ainsi que le nombre total autorisé par l'annonce du joueur. On rappelle que les robots ne peuvent changer de direction que lorsqu'ils rencontrent un mur ou un autre autre robot, et ne peuvent dans ce cas que ricocher à gauche ou à droite (et pas faire demi-tour ni bien sûr continuer tout droit).

Si la solution proposée est correcte (atteint la cible en au plus le nombre de coups annoncé), le joueur marque un point. Sinon, tous les robots sont replacés à leurs positions précédentes et le joueur ayant proposé la deuxième plus petite annonce est invité à démontrer sa solution, etc.

2 Réversi à deux joueurs

Description du jeu

Le jeu de Réversi se joue à l'aide du matériel suivant :

- un plateau de 64 cases (8 x 8) ;
- 64 pions ayant chacun une face blanche et une face noire.

Au début de la partie, 4 pions (2 sur la face noire et 2 sur la blanche) sont disposés en croix au centre du plateau. Les adversaires jouent à tour de rôle, sauf si un joueur se trouve dans l'impossibilité de placer un pion. Dans ce cas, il passe son tour. La partie prend fin lorsque tous les pions ont été joués ou lorsqu'aucun des deux joueurs ne peut plus jouer.

Chaque joueur joue avec une couleur. Les pions dont la face blanche (resp. noire) est visible seront appelés indifféremment pions blancs (noirs) ou pions du joueur blanc (noir) par la suite.

Pour jouer, le joueur doit placer un nouveau pion de sa couleur de telle sorte que :

- il soit sur la même ligne ou la même colonne qu'au moins un autre pion de sa couleur ;
- entre ces deux pions, il n'y ait que des pions de l'autre couleur.

Il retourne alors les pions adverses encadrés par le pion qu'il vient de jouer et ses autres pions.

Le joueur qui a le plus de pions de sa couleur à la fin de la partie a gagné.

Exemple de partie

Voici le début d'une partie. La figure 1 représente la position initiale et les figures 2, 3 et 4 le résultat de trois coups successibles possibles. Les deux couleurs sont symbolisées par o et x. Le joueur o commence.

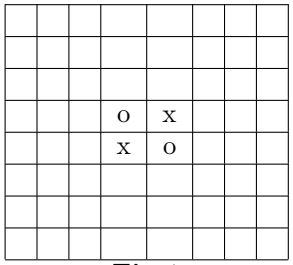


Fig 1

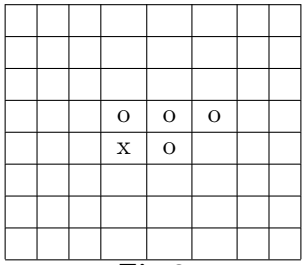


Fig 2

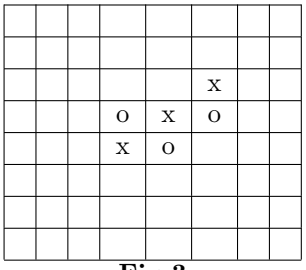


Fig 3

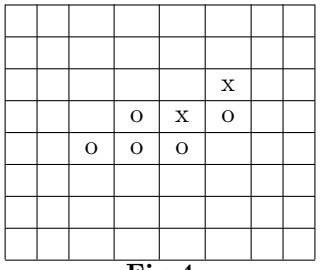


Fig 4

Dans les figure suivantes, les coups possibles pour o sont montrés à l'aide du symbole s, ceux du joueur x à l'aide du symbole p.

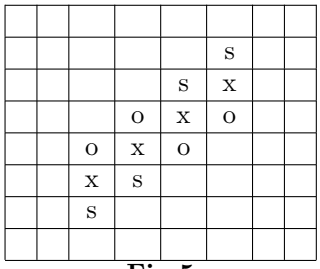


Fig 5

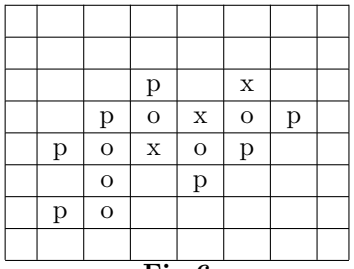


Fig 6

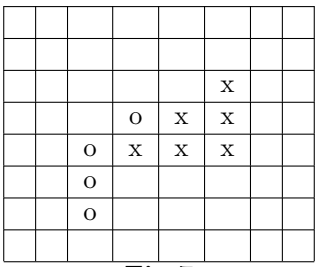


Fig 7

Fonctionnalités obligatoires

Le programme réalisé doit :

- dessiner le plateau de jeu ;
- dessiner les pions déjà placés ;
- au tour de chaque joueur, indiquer les cases où il est permis de jouer ;
- retourner les pions après que chaque joueur a placé un pion ;
- détecter la fin de la partie quand elle survient, ainsi qu'une situation où un joueur est obligé de passer.

Le dessin du plateau et des pions sera réalisé à l'aide de la bibliothèque graphique `upemtk`. Le choix du coup sera fait par un clic de souris sur la case souhaitée.

Le plateau pourra être représenté dans votre programme par une liste de listes d'entiers. Une case contenant un pion noir sera par exemple représentée par l'entier 1, une case contenant un pion blanc par l'entier -1, une case vide par 0.

Améliorations suggérées

Sauvegarde et restauration de partie

On pourra offrir aux joueurs la possibilité de sauvegarder une partie et de charger une partie précédemment sauvegardée. La description de la partie en cours sera stockée dans un fichier contenant éventuellement la date de début de la partie. On pourra également stocker le nom des joueurs concernés.

Jeu contre l'ordinateur

Une autre amélioration envisageable est de donner la possibilité de faire jouer l'ordinateur. On pourra s'inspirer de la méthode suivante en remarquant que les cases de coin et les cases y menant sont importantes.

Pour choisir le meilleur déplacement du pion, on peut attribuer un score à chaque coup possibles dans une situation donnée. Le choix du programme ira alors vers la case ayant le plus grand score.

Par exemple pour le joueur noir :

- une case occupée ou noire aura un score négatif (-1) ;
- les autres auront comme score le nombre de cases blanches vides adjacentes à la case à noter.

Une autre possibilité est de favoriser le coup que l'on juge le plus gênant pour l'adversaire (et si possible le moins gênant pour soi-même).

Ces suggestions ne sont pas focrément les meilleures mais sont une piste de départ. Les étudiants réalisant ce projet sont libres de rechercher d'autres méthodes pour évaluer la qualité des coups et rendre leur programme le plus fort possible.

3 Jeu d'échecs

Le but du projet est de représenter un échiquier et les déplacements de pièces en cours de partie.

Fonctionnalités obligatoires

L'objectif de ce TP n'est **pas** de réaliser un programme sachant jouer aux échecs, mais simplement de proposer un échiquier graphique grâce auquel deux joueurs "humains" peuvent s'affronter. L'accent devra être mis sur la convivialité et la commodité de l'interface.

L'échiquier sera dessiné à l'aide de la bibliothèque graphique `upemtk`. Dans un premier temps, les pièces seront représentées par une simple lettre (R pour Roi, D pour Dame, T pour Tour, F pour Fou, C pour Cavalier et P pour Pion). La sélection de la pièce à déplacer se fera par un clic de souris sur la pièce souhaitée, suivi d'un clic de souris sur la case d'arrivée. Le programme doit évidemment vérifier que le coup demandé est conforme aux règles des échecs (voir par exemple cette explication des règles), ou encore mieux, mettre en évidence les déplacements autorisés pour la pièce sélectionnée.

Améliorations suggérées

Les étudiants choisissant ce projet devront obligatoirement traiter au moins une des améliorations suivantes (même partiellement) :

1. **Améliorations de l'interface.** Le programme permet la saisie du nom de chaque joueur, représente les pièces du jeu d'échecs à l'aide d'images (par exemple celles- ci), annonce les tours de jeu...

2. **Respect des règles.** Le programme permet d'utiliser les règles "avancées" du jeu d'échecs (roque, prise en passant, parties nulles, etc.).

3. **Situation du roi.** Après chaque déplacement de pièce, on signale si l'un des deux rois est en situation d'échec, ou en situation de mat. Dans ce cas, on annonce le vainqueur de la partie.

4. **Position initiale du plateau.** Le programme accepte une description de position initiale différente de la position par défaut, par exemple en utilisant la notation décrite sur cette page. Cette amélioration peut permettre de charger des problèmes d'échecs, en prenant soin d'encoder également l'objectif du problème (mat blanc en 2 coups...)

5. **Feuille de partie.** Chaque coup joué est noté, en suivant les conventions d'usages (voir par exemple cette explication), dans un fichier texte. Le nom du fichier doit mentionner la date et l'heure de début de la partie. On peut envisager un mode "spectateur" permettant de visualiser les coups successifs d'une partie précédemment jouée. Cette fonctionnalité devrait également permettre de sauvegarder une partie non terminée et de la reprendre plus tard.

Toute autre amélioration (de difficulté raisonnable) est envisageable à condition d'en avoir parlé avec le chargé de TP.

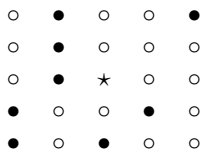
4 Le jeu de la vie

Introduction

Le jeu de la vie se déroule normalement sur un quadrillage infini à mailles carrées (On travaillera sur un quadrillage fini).

Chaque maille du quadrillage contient soit une cellule vivante (notée par •, dans les schémas suivants), soit une cellule morte (notée par ∘dans les schémas suivants).

On appelle voisines d'une cellule les 8 cellules qui sont en contact avec celle-ci. Ainsi on constate que la cellule ⋆ de l'exemple ci-dessous possède pour voisines 3 cellules vivantes (•) et 5 cellules mortes (∘).

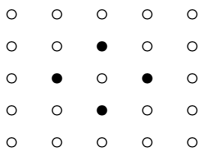


On détermine l'état du plateau de jeu au tour suivant en appliquant les régles suivantes :

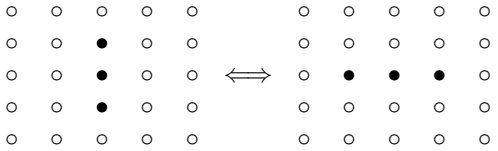
- Une cellule vivante meurt d'asphyxie si elle possède moins de 2 voisines.
- Une cellule vivante meurt d'étouffement si elle possède plus de 3 voisines.
- Une cellule morte ressucite si elle possède exactement 3 voisines.
- Dans tous les autres cas il n'y a pas de changement.

En itérant ce processus plusieurs fois, on constate l'apparition des phénomènes suivants :

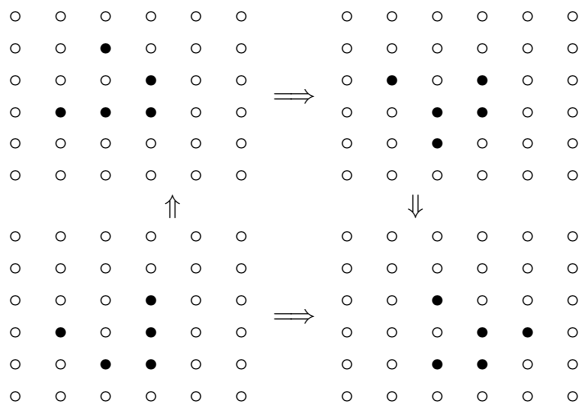
- Il y a des formes stables qui ne bougent pas : dans l'exemple suivant chacune des 4 cellules vivantes possède exactement 2 voisines, donc elles vivent. Et aucune des cellules mortes possède exactement 3 voisines pour ressuciter. Donc la configuration ne se transforme pas.



- Il y a des formes oscillantes immobiles :



- Il y a des formes oscillantes mobiles :



- Et bien d'autres formes plus complexes…

Implémentation

Un quadrillage infini étant non implémentable sur une machine, on se contentera d'un plateau fini de taille $N \times N$ repésenté par une liste de liste contenant des booléens (**True**=cellule vivante, **False**=cellule morte).
De plus on supposera que le pourtour du plateau est constitué de cellules qui seront toujours mortes.

Une execution du programme devra se dérouler comme suit :

1. Au départ le plateau est initialisé par uniquement des cellule mortes.
2. Saisir la liste des cellules vivantes en précisant leurs abscisses et leurs ordonnées.
3. Saisir le nombre d'itérations que l'on veut effectuer.
4. Indiquer si l'on veut afficher le resultat des itérations intermédiaires. Et si *oui* le programme attendra, par la suite, l'appui sur **RETURN** entre chaque affichage.
5. la simulation est effectuée en fonction des paramètres précédents, et le résultat est affiché à l'écran.

Votre programme proposera un affichage :

1. en mode texte dans le terminal en utilisant par exemple le caractère `X` pour représenter les cellules mortes et le caractère `o` pour représenter les cellules vivantes ;
2. en mode graphique en utilisant par exemple la bibliothèque upemtk.

Toute fonctionnalité supplémentaire sera la bienvenue (par exemple la gestion d'un plateau circulaire (quand on atteind une extrémité du plateau, on se retrouve à l'autre extrémité)). On pourra également saisir l'emplacement des cellules vivantes initiales grâce à la souris

5 Bataille navale

Le but du projet est de réaliser un programme permettant de jouer à la bataille navale. La partie minimum du projet consiste uniquement à gérer les réponses de l'adversaire.

Les différentes choix de l'utilisateur seront gérés à l'aide d'un menu. Par défaut, le terrain de jeu est un carré de $20 * 20$ cases et l'ensemble des bateaux est formé de :

Premier choix	1 navire de 6 cases,
Deuxième choix	2 navires de 5 cases,
Troisième choix	3 navires de 4 cases,
Quatrième choix	4 navires de 3 cases,
Cinquième choix	5 navires de 2 cases,
Sixième choix	6 navires de 1 case.

Une option du menu devra permettre à l'utilisateur de changer ces valeurs.

Déroulement du programme dans sa version minimale :

- première phase : définir la place des bateaux.
  Pour chaque nouveau bateau, on affichera un plateau de jeu avec les bateaux positionnés. Les bateaux ne peuvent être qu'en position horizontale ou verticale. Toutes les cases d'un bateau sont alignés et doivent être sur le plateau de jeu. Deux bateaux ne peuvent être situés sur des cases contigues. Tous les bateaux doivent être placés.
- deuxième phase : simulation de l'adversaire.
  Après avoir caché le plateau contenant les bateaux défini dans la première phase, on affiche un nouveau plateau indiquant la position et le résultat des tirs effectués, ainsi que les vaisseaux coulés.

Un tir est entré au clavier comme un couple de coordonnées dont on vérifiera la validité.

Le résultat indiqué sera :

- si on atteint un bateau :
  - *touché* si le bateau possède encore des cases non atteintes
  - *coulé* si toutes les cases du navire sont atteintes
- si on n'atteint pas de bateau :
  - *rien* s'il n'y a pas de bateau dans l'une des 8 cases adjacentes
  - *en vue* si au moins un navire est dans une des 8 cases adjacentes

Déroulement du jeu à deux joueurs

1. L'utilisateur place lui même ses bateaux.
2. Il dispose d'un plateau tir affichant les résultats de ses tirs et d'un plateau bateaux affichant ses bateaux et les tirs de l'adversaire.
3. Chaque joueur effectue un tir alternativement.

Une symbolisation claire sera choisie pour chacun de ces résultats.

Une option du menu devra permettre d'afficher simultanément les deux plateaux, afin de permettre une vérification lors de la soutenance.

La représentation des vaisseaux pourra utiliser les caractères alphanumérique ou, beaucoup mieux, les fonctions de la bibliothèque graphique upemtk.

Evolution du programme
Le jeu à deux joueurs :

La première évolution : consiste à placer aléatoirement les bateaux.

Pour cela on utilisera la fonction de hasard rand()qui renvoie un entier pseudo-aléatoire compris entre 0 et 32767. Afin de ne pas obtenir toujours les mêmes suites pseudo-aléatoires, la fonction doit être initialisée au début de votre programme, par l'instruction `srand(getpid())`. *Voir man rand.*

Deuxième évolution ; Le jeu à un joueur, le deuxième étant géré par le programme. Le menu propose plusieurs adversaires possibles, adoptant chacun des stratégies de jeu différentes.

Troisième évolution : le programme gère les deux joueurs afin de déterminer la meilleure des stratégie implémentées.

Quatrième évolution : suivant votre inspiration.

6 Takuzu

Le Takuzu est un puzzle logique de type Sudoku.

Le jeu

L'objectif du puzzle est de remplir une grille carrée de taille paire avec des 0 et des 1 en respectant les conditions suivantes :

- chaque ligne et colonne doit contenir autant de 0 que de 1 (cette condition implique que le carré est de taille paire),
- les lignes ou colonnes identiques sont interdites,
- il ne doit pas y avoir plus de deux 0 ou 1 placés l'un à coté ou en dessous de l'autre.

Les figures 2 et 3 présentent des exemples de Takuzu.

	1		0
		0	
	0		
1	1		

$\longrightarrow$

0	1	1	0
1	0	0	1
0	0	1	1
1	1	0	0

Fig. 2: Grille de Takuzu de taille 4.

		0		1			1		
1		0		1				1	
	0					0			
		0	0				1	1	
								0	
						0			
0	0		0			0			0
		1		1					
	0							0	
1					1				

$\longrightarrow$

0	1	0	0	1	0	1	1	0	1
1	1	0	0	1	0	1	0	1	0
0	0	1	1	0	1	0	1	0	1
0	1	0	0	1	1	0	1	1	0
1	0	1	1	0	0	1	0	0	1
1	1	0	1	0	1	0	0	1	0
0	0	1	0	1	1	0	1	1	0
0	1	1	0	1	0	1	0	0	1
1	0	0	1	0	0	1	1	0	1
1	0	1	1	0	1	0	0	1	0

Fig. 3: Grille de Takuzu de taille 10.

Description du projet

Le projet consiste à :

1. charger un puzzle depuis un fichier,
2. permettre à l'utilisateur d'essayer de résoudre le puzzle en mode terminal : le puzzle est affiché à chaque tour et l'utilisateur doit entrer au clavier les coordonnées d'une case. Si cette case est déjà occupée, son contenu est supprimé, si elle était libre, le programme demande à l'utilisateur la valeur qu'il souhaite y stocker. Le programme doit empêcher les coups illégaux et afficher un message lorsque l'utilisateur a gagné.

3. permettre à l'utilisateur d'essayer de résoudre le puzzle en mode graphique. La grille est affichée dans une fenêtre et un clic sur un case change sa valeur de façon cyclique entre 1, 0, et vide.

4. permettre à l'uitlisateur d'annuler, un par un, les coups précédents

La structure de donnée utilisée pour manipuler le puzzle sera une liste de listes.
Le confort d'utilisation est un élément important. Il est possible par exemple :

1. de remplacer en mode graphique les chiffres 0 et 1 par des jetons blancs et noirs ou des ronds et des croix, etc. ;

2. de donner lorsqu'il le demande un indice à l'utilisateur, en essayant d'appliquer une des trois règles du jeu.

7 ISOLA

Il s'agit d'un jeu à deux joueurs.

Description du jeu

Le matériel

- Un plateau carré de 36 cases initialement blanches (6 x 6).
- Deux pions différents : un par joueur.
- Des carrés noirs pour interdire certaines cases.

Les règles du jeu

La position initiale des pions est laissée au choix des joueurs. Les adversaires jouent à tour de rôle. Jouer consiste à effectuer les deux étapes suivantes :

- Déplacer son pion sur l'une des cases blanches voisines qui n'est pas occupée par le pion adverse. Il n'est donc pas possible de se déplacer sur une case noire, ni sur la case occupée par le pion adverse. Un pion a donc au maximum 8 possibilités de déplacement.
- Interdire l'accès d'une case en y plaçant un carré noir. Evidemment, il n'est possible de placer un carré noir ni sur une case occupée par un pion, ni sur une case noire.

Voici un début de partie. La figure 1 représente la position initiale choisie par les deux joueurs. Les deux joueurs sont O et S. C'est O qui commence à jouer dans la figure 2. X symbolise la case interdite par O.

	1	2	3	4	5	6
A						
B		O				
C						
D					S	
E						
F						

Fig 1

	1	2	3	4	5	6
A						
B						
C			O		X	
D					S	
E						
F						

Fig 2

Après ce remarquable coup, S ne peut plus se déplacer en C-5 …

But du jeu

Un joueur a perdu, si, quand vient son tour, il ne peut pas se déplacer. C'est à dire quand toutes les cases voisines de son pion sont soit noires, soit occupées par le pion adverse. Voici deux exemples de fin de partie :

	1	2	3	4	5	6
A						
B				X		
C			X		X	X
D		X		X	X	O
E		X	S		X	X
F			X			

Fig 3

	1	2	3	4	5	6
A						
B		X	X	X		X
C		X	S	X		
D		X	X	O		X
E			X		X	
F				X		

Fig 4

Dans la figure 3, O ne peut plus jouer, donc S a gagné. Dans la figure 4, O vient de jouer et gagne car le pion O bouche la dernière possibilité de déplacement de S.

Description du projet

Le projet consiste :

1. à fournir les outils pour pouvoir jouer :
   - dessin du plateau de jeu,
   - dessin des pions que l'on peut déplacer à tour de rôle selon les règles énoncées ci-dessus,
   - possibilité pour chaque joueur, après qu'il eût déplacé son pion, de noircir une case de son choix (tout en respectant les règles),
   - le programme doit détecter la fin de la partie quand elle survient.
2. à donner la possibilité à l'utilisateur, de jouer contre l'ordinateur. Il pourra choisir de jouer avec un ami humain, ou bien avec votre programme. Il va donc falloir que vous écriviez un programme qui puisse jouer à Isola le mieux possible. Et si vous êtes plusieurs à choisir ce projet, on pourra comparer vos différentes stratégies en faisant jouer vos programmes les uns contre les autres !

Améliorations possibles

Toute amélioration est la bienvenue. En voici trois possibles :

- Vous pouvez laisser le choix de la taille du plateau de jeu à l'utilisateur. Il peut en effet être intéressant d'essayer de jouer sur des plateaux de taille différente (et même des plateaux rectangulaires : 8×6, 24×2, *etc.*).
- Vous pouvez utiliser une fonction aléatoire pour éviter que le programme ne choisisse toujours la même position initiale. Cela peut aussi servir à choisir un déplacement quand plusieurs possibilités sont considérées comme équivalentes par le programme.
- Il existe une autre version de ce jeu légèrement différente où seuls les déplacements des pions sont modifiés. Ils ne se déplacent plus sur une case adjacente, mais à la manière d'un cavalier du jeu d'échec. Vous pouvez donc laisser le choix du mode de déplacement à l'utilisateur : type classique ou type cavalier. On peut même imaginer que les deux joueurs puissent choisir des modes différents !

Si vous avez d'autres idées d'amélioration, vous pouvez bien sûr les réaliser sans faire celles proposées.

Implantation

Pour saisir les déplacements des pions et les cases que l'on noircit, vous pourrez numéroter les colonnes et mettre des lettres sur les lignes. Ainsi, chaque case aura un nom : (C-5), (E-3), …. Il faudra alors faire une transcription pour transformer le nom de la case en un couple de coordonnées dans une clé pour un dictionnaire python.

Le plateau pourra être représenté dans votre programmme par un dictionnaire de données. Une case sera *'Prise'* si elle est occupée par un pion, *'Noire'* si c'est une case noire, ou *'Vide'* dans le cas d'une case blanche normale. Voici donc un moyen possible de déclarer et d'utiliser un plateau :

```
plateau = {};
plateau[...] = ...;
```

Il reste à déterminer ce qu'il faut mettre en clé et en valeur.

Vous aurez besoin de la position des pions et peut-être du dessin de chaque pion. Voici exemple :

```
# Declaration d'une variable pion
pion = {'ligne' : 10, 'colonne' : 20, 'dessin' : 'rond'}
# Affichage du contenu de la structure
print(pion)
# Affichage de la valeur associée champ colonne
print(pion['colonne'])
# Modification de la valeur associée au champ colonne
pion['colonne'] = 19
```

Comment faire jouer l'ordinateur ?

Pour choisir le meilleur déplacement du pion, vous pouvez évaluer les différentes positions possibles.
Exemple d'évaluation :

- une case occupée ou noire aura une valeur négative (-1) ;
- les autres cases auront comme valeur le nombre de cases blanches vides qui leur sont adjacentes.

Le choix du programme ira vers la case ayant la plus grande évaluation. Cette façon de faire n'est pas nécessairement la meilleure. Peut-être trouverez-vous mieux ?

Pour interdire une case, vous pourrez utiliser le même principe. Une fonction évaluera la case la plus gênante pour l'adversaire et si possible la moins gênante pour soi. A vous de trouver une fonction approchant ces objectifs.

8 Représentation d'un labyrinthe

But

Le but du projet est de représenter une vue subjective des couloirs composants un labyrinthe, ainsi que de permettre un déplacement dans ce labyrinthe.
Les déplacements se feront à l'aide d'une zone de 6 boutons :

- quart de tour à gauche,
- avancer d'une case,
- quart de tour à droite,
- pas sur le coté à gauche,
- pas en arrière,
- pas sur le coté à droite.

Description des fonctionnalités obligatoire

- Affichage des lieux en fonction de la position du joueur dans le labyrinthe et de son orientation,
- Gestion des mouvements, en contrôlant leur validité (pas de passe-muraille),
- Détecter la fin d'une partie (le joueur a atteint la sortie).

Implémentation

Le labyrinthe est contenu dans un rectangle de moins de LONGUEUR×LARGEUR cases. Le labyrinthe sera stockée sous forme d' un fichier texte lu par le programme.

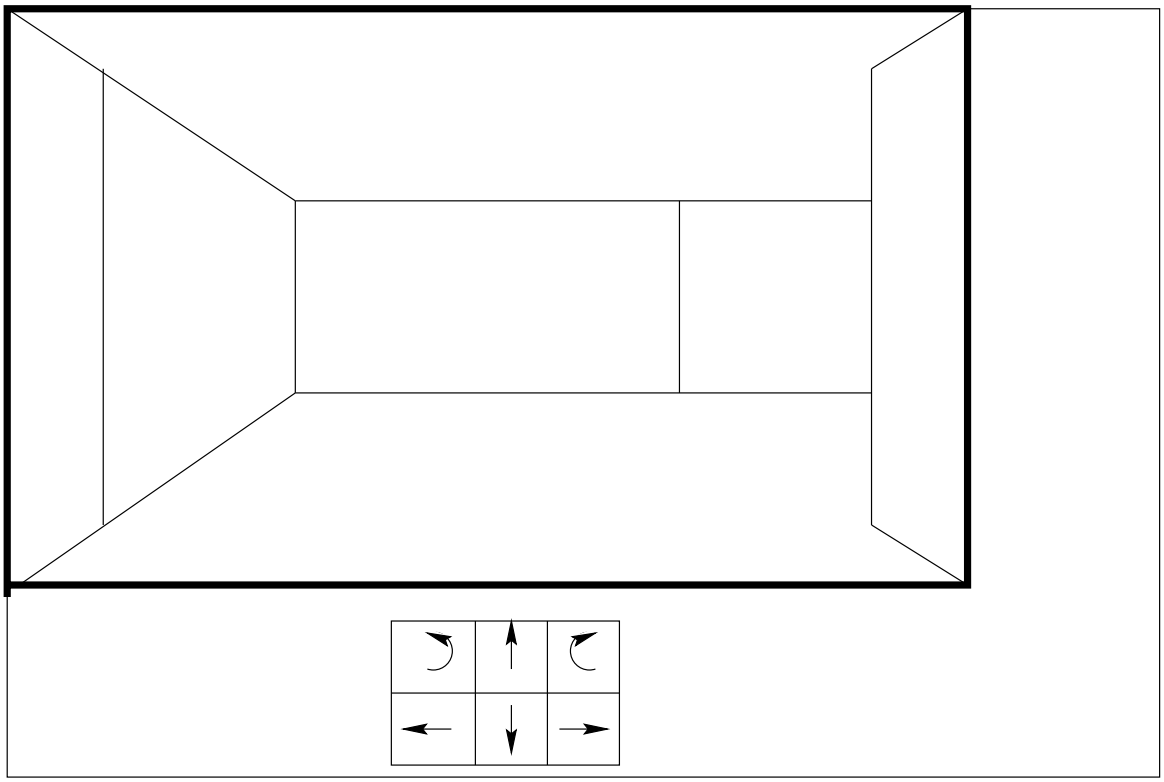
Les éléments du labyrinthe sont représenté dans le fichier texte par les caractères suivants :

- `*` pour un mur
- `.` pour une case vide
- `@` pour le joueur au début de la partie
- `x` pour la sortie

L'affichage en vue subjective se fera en '*fil de fer*' à une case de distance. Ainsi la partie du fichier texte :

```
******
*.....
*@****
*.****
```

avec une orientation vers le nord devra correspondre à l'affichage d'un tournant vers la droite. Une présentation possible de cette situation correspond à l'écran :

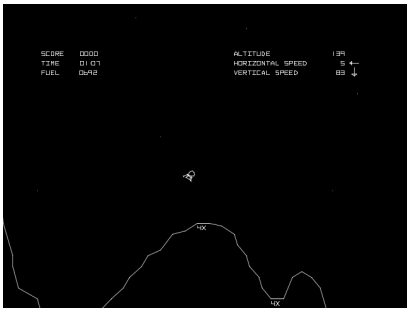


On devra gérer des salles, c'est à dire des murs éloignés de plus d'une case.

Améliorations possibles

- Affichage d'une carte construite au fur et a mesure du déplacement
- Possibilité de sauvegarde
- Gestion d'étages
- Génération aléatoire du labyrinthe
- Gestion d'habitants et de leur rencontre.

9 Lunar Lander – Pose la fusée



Le nom *Lunar lander* désigne un véhicule spatial destiné à se poser sur la lune, et également une série de jeux vidéos datant des années 70. Le concept de chacun de ces jeux est de contrôler une fusée afin de la faire aterrir en toute sécurité dans un environnement 2D représentant un paysage lunaire. Différentes versions existent en ligne, on peut en voir une démonstration ici.

Principe du jeu

Au début de la partie la fusée possède une vitesse nulle et se trouve à une certaine distance du sol. Elle commence immédiatement à accélérer vers le bas sous l'effet de la gravité. Le joueur contrôle le réacteur principal ainsi que les propulseurs latéraux. L'objectif est de poser la fusée en douceur et à la verticale, c'est-à-dire à faible vitesse, perpendiculairement au sol et sur une surface horizontale et régulière. Enfin, l'appareil disposant d'une réserve de carburant limitée, il faut veiller à ce que les manoeuvres ne drainent pas tout le carburant. Le jeu est gagné lorsque la fusée est posée dans les conditions décrites, et perdu si la fusée ne se pose pas sur une zone horizontale et plane, ne se pose pas à la verticale ou se pose trop vite (crash).

Notez que le jeu n'est pas immédiatement perdu si la fusée arrive à court de carburant : elle ne peut simplement plus utiliser ses propulseurs (ce qui limite grandement les chances de réussite, admettons-le). Les paramètres du jeu (vitesse maximum d'alunissage, consommation de carburant, force de la gravité, puissance des propulseurs etc.) pourront être choisi par l'utilisateur en début de jeu grâce à un menu. Ce menu pourra proposer un choix parmi plusieurs valeurs fixées ou laissera l'utilisateur saisir des valeurs quelconques. Le menu permettra également de choisir le niveau de jeu si plusieurs niveaux existent.

Implémentation

Mouvement de la fusée

Le principal enjeu de ce projet est de simuler un comportement physiquement crédible de la fusée, en particulier l'effet d'accélération dû à la gravité et aux propulseurs. La fusée est attirée par la planète, et s'en rapproche donc de plus en plus vite si le joueur ne fait rien. De même, activer les propulseurs ne provoque pas un changement de direction brutal mais progressif : par exemple, si la fusée est verticale et se dirige vers la surface et que le propulseur est allumé de manière prolongée, elle ralentira peu à peu avant éventuellement de repartir vers le haut. Autrement dit, le déplacement de la fusée prend en compte sa vitesse et sa direction actuelle, ce qui donne un effet d'inertie. Pour obtenir ce comportement,on utilisera l'algorithme suivant :

- À tout moment, le déplacement de la fusée est représenté par un point (couple de coordonnées sur l'écran) et par un vecteur décrivant sa vitesse actuelle (également un couple de coordonnées).

- Les différentes forces qui s'exercent sur la fusée (gravité, propulsion dûe aux réacteurs... éventuellement vent !) sont elles aussi représentées par des vecteurs.
- À chaque unité de temps, on additionne au vecteur décrivant la vitesse de la fusée la *somme* de tous les vecteurs décrivant les forces qui s'exercent sur elle. C'est ainsi que l'on calcule l'accélération de la fusée (seconde loi de Newton).

Quant aux capacités de déplacement de la fusée, on peut distinguer trois degrés de complexité :

- Uniquement un propulseur principal dirigé vers le bas (fusée ne se déplaçant que verticalement) ;
- Ajout de propulseurs latéraux déplaçant la fusée vers la gauche ou la droite mais sans rotation (propulseur vertical toujours dirigé vers le bas). Il s'agit de la version facile du jeu, présentée dans la deuxième partie de la vidéo liée ci-dessus ;
- Propulseurs latéraux faisant pivoter la fusée (propulseur principal non nécessairement vertical). C'est la version difficile du jeu.

Modélisation du terrain

Le terrain lunaire est vu en coupe verticale. La topographie du paysage est aléatoire, mais doit obligatoirement présenter au moins une zone d'aterissage viable. Dans un premier temps, il est possible de mettre au point les principaux éléments du jeu avec un terrain uniformément plat, puis ajouter la gestion du relief. On citera par ordre de complexité :

- Terrain plat ;
- Terrain formé de lignes brisées ;
- Terrain aléatoire sans "surplombs" ;
- Terrain quelconque (incluant surplombs, grottes, etc.).

Comme pour les modes de déplacement, il est demandé de commencer par le modèle le plus simple. Vous décrirez la méthode de génération de la topographie choisie dans le rapport.

Interface et améliorations possibles

L'interface doit permettre de démarrer le jeu, de recommencer en cas d'échec ou de réussite, de quitter le jeu. Les informations indispensables au joueur doivent être affichés en permanence pendant la partie : le vecteur vitesse de la fusée, l'altitude et la quantité de carburant restante.

De nombreuses améliorations sont possibles, mais elles ne doivent être abordées que si le reste du projet fonctionne déjà **parfaitement** :

- Affichage d'un score (nombre de réussites successives sans crash ou score dépendant de la douceur de l'alunissage ou du temps écoulé par exemple).
- Autres choix d'options ou de paramètres de difficulté, etc.
- Gestion de plusieurs niveaux (c'est-à-dire paysages) différents, de niveaux aléatoires, défilement du paysage si le niveau ne tient pas entièrement sur l'écran, possibilité d'effectuer un zoom.
- Autres éléments perturbateurs (vent fixe ou variable, objets mobiles gênant le pilotage, voire hostiles, passagers à recueillir).
- Améliorations visuelles (animation des moteurs, dessin de la fusée, couleurs, représentation des crash...).