M1102: Introduction à l'algorithmique et à la programmation
Projet n°1
Le Labyrinthe

1 Présentation

L'objectif de ce projet est de programmer le jeu du Labyrinthe. Le Labyrinthe est un jeu de société qui se joue à deux, trois ou quatre joueurs. Le plateau du Labyrinthe (voir figure 1) est une sorte de grille 7×7 sur laquelle on place des cartes cartonnées. Chacune de ces cartes représente un morceau de labyrinthe qui peut contenir ou non un symbole représentant un trésor. Certaines cartes sont fixes (les quatre coins, puis une case sur deux sur tout le plateau), les autres cartes sont amovibles. Au début du jeu on dispose de manière aléatoire les cartes amovibles sur le plateau. Il y a une carte amovible de plus que de places sur le plateau. Chaque joueur se voit attribuer un pion de couleur qu'il doit placer dans le coin correspondant à cette couleur. Il se voit aussi attribuer un certain nombre de trésors qu'il doit retrouver dans le labyrinthe. Chaque tour de jeu se décompose en deux phases :

1. le joueur choisit un endroit pour insérer la carte amovible. Pour placer cette carte amovible, il pousse la ligne ou la colonne choisie d'un cran et récupère la carte *expulsée* par ce décalage. Si un pion se trouvait sur la carte expulsée, ce pion est placé sur la carte insérée. La seule restriction est qu'on ne peut pas remettre la carte amovible à l'emplacement d'où elle vient d'être expulsée.
2. le joueur déplace son pion en suivant les couloirs du labyrinthe en essayant d'atteindre le trésor qu'il doit trouver.

La partie se termine lorsqu'un des joueurs a réussi à trouver tous les trésors qui lui ont été attribués.

Figure 1 – Le plateau du labyrinthe

2 Travail à faire

Vous devrez développer, par groupe de trois personnes, un programme qui permet à des personnes (de un à quatre) de jouer au Labyrinthe. Voici quelques précisions sur les différents éléments du jeu.
Il y a trois type de cartes différentes dans le jeu

— les angles ┏ au nombre de vingt,
— les jonctions ┣ au nombre de dix-huit et
— les tout-droits ┃ au nombre de douze.

La figure 2 représente le plateau avec ses cartes fixes. Dans cette figure, on peut voir comment sont repérés les 4 directions du plateau et la numérotation que l'on adoptera dans la description du sujet. Il est à noter que toutes les cartes fixes sauf les quatre coins possèdent un trésor. Par contre, les quatre coins seront les points de départ respectifs des quatre joueurs.

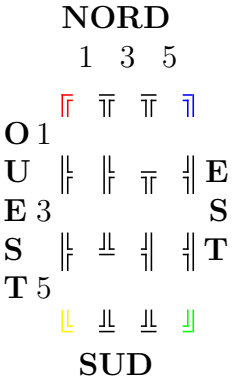


FIGURE 2 – Position des cartes fixes

3 Structure du projet

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. Cette structuration vous permettra d'utiliser les scripts d'affichage en mode texte et en mode graphique qui vous sont fournis. Le schéma présenté figure 3 représente la dépendance entre les différentes parties du projet.

3.1 les cartes

Dans le script `carte.py` vous trouvez la signature des fonctions qui vous sont demandées pour gérer les cartes. Une carte se caractérise par la présence ou non de murs sur ses quatre directions : nord, est, sud et ouest (voir figure 4).
De plus, une carte peut contenir un trésor et entre 0 et quatre joueurs. Les joueurs sont représentés par leur numéro (entre 1 et 4) appelé pion. Les opérations que l'on peut faire sur une carte sont les suivantes :

— mettre ou enlever un pion
— mettre ou enlever un trésor
— faire tourner la carte dans le sens horaire (ou anti-horaire). Par exemple la carte de la figure 4(c) est le résultat de la rotation dans le sens anti-horaire de la carte de la figure 4(a).

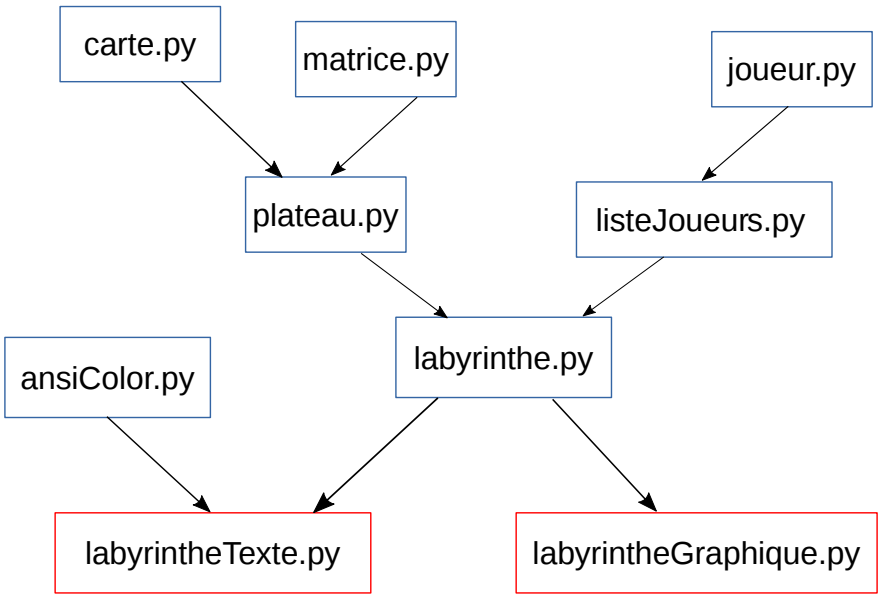
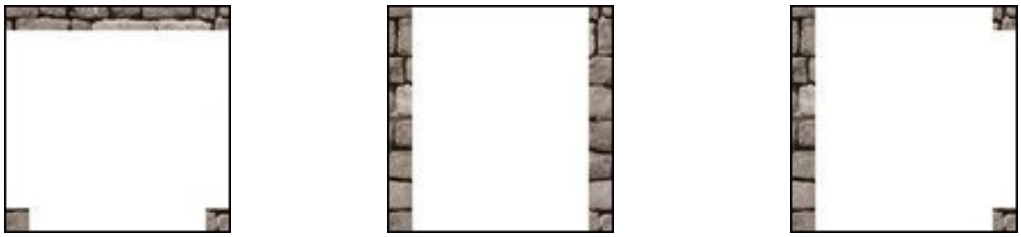


FIGURE 3 – Architecture du projet



(a) Une carte ne possédant qu'un mur au nord (b) Une carte possédant des murs à l'est et à l'ouest (c) Une carte ne possédant qu'un mur à l'ouest

FIGURE 4 – Exemples de cartes

Par ailleurs, quatre fonctions permettent de savoir si il y a un passage entre deux cartes contiguës du plateau. Vous pouvez choisir la représentation que vous souhaitez pour vos cartes, par contre, les différentes fonctions doivent être implémentées et **répondre aux spécifications demandées**.

3.2 Les joueurs

Un joueur possède un nom et une liste de trésors à trouver. Chaque trésor est représenté par un entier qui l'identifie. La fonction Joueur crée un joueur avec son nom et une liste de trésor à trouver vide. Une fonction permet d'ajouter un trésor à trouver au joueur, de connaître le prochain trésor à trouver du joueur, d'enlever le trésor qui a été trouvé, de connaître le nombre de trésors à trouver et de connaître son nom.
Vous pouvez choisir la représentation que vous souhaitez pour vos joueurs, par contre, les différentes fonctions doivent être implémentées et **répondre aux spécifications demandées**.

3.3 La liste de joueurs

La liste de joueurs, comme son nom l'indique gère la liste des joueurs qui participent à la partie. Une fonction va permettre de distribuer aléatoirement les trésors aux joueurs de la

partie. Cette structure de données doit gérer le joueur courant (retrouver les informations le concernant) que l'on doit pouvoir changer
Vous pouvez choisir la représentation de vos matrices à condition d'implémenter toutes les fonctions demandées **en suivant leur spécification**.

3.4 La matrice

Il s'agit simplement d'une version étendue des matrices que vous avez manipulées en TD. Seules des fonctions permettant d'effectuer des décalages de lignes ou de colonnes sont à implémenter.
Vous pouvez choisir la représentation de vos matrices à condition d'implémenter toutes les fonctions demandées **en suivant leur spécification**.

3.5 La plateau

Il s'agit simplement d'une matrice (telle que définie dans la section 3.4) de cartes (telles que définies dans la section 3.1). Dans le plateau, on va gérer l'initialisation du plateau de jeu avec un placement aléatoire des cartes. Cette structure de données gère aussi les objets (pions de joueurs et trésors) posés sur le plateau afin de les repérer, les poser ou les enlever. Enfin les fonctions de test d'accessibilité dans le labyrinthe se trouvent dans ce fichier.
Vous ne devez pas implémenter de structures de données particulières pour le plateau, le plateau est une matrice de cartes.

3.6 Vue générale de la structure du programme

Afin de mieux comprendre les fonctions qui vous sont demandées dans les fichiers `labyrinthe.py` et `labyrintheTexte.py`, nous allons présenter une vue générale du déroulement d'un tour de jeu. La structuration du programme est faite de telle sorte que `labyrintheTexte.py` ne se préoccupe que des interactions avec l'utilisateur et de l'affichage et `labyrinthe.py` ne se préoccupe que de la manipulation des différents objets du labyrinthe. La figure 5 présente schématiquement la manière dont un tour de jeu s'effectue et comment les différentes fonctions que vous avez à implémenter s'articulent.
Dans cette figure, les ovales non grisés représentent des fonctions que vous aurez à implémenter. Cette figure illustre le déroulement d'un tour de jeu. Lors de la phase 1, le joueur peut entrer un ordre au clavier grâce à la fonction `saisirOrdre()`. Cet ordre est passé sous la forme d'un couple de deux valeurs expliqué plus loin. Cet ordre est passé à la fonction `executerActionPhase1()` du fichier `labyrinthe.py` qui va mettre à jour le labyrinthe en fonction de l'ordre donné. Le résultat de cette fonction sera utilisé par l'affichage pour rendre compte à l'utilisateur de ce qui s'est passé afin qu'il puisse entrer un nouvel ordre en connaissance de cause. Cette boucle est recommencée jusqu'à ce que l'utilisateur donne l'ordre d'insérer la carte à jouer. Cet ordre va provoquer le passage dans la phase 2 du jeu. Dans cette dernière, c'est la fonction `saisirDeplacement()` qui permet au joueur d'indiquer sa case de destination souhaitée. La valeur de retour de cette fonction est une nouvelle fois un couple de valeurs. Ce couple est passé à la fonction `accessibleDistJoueurCourant()` qui va vérifier l'accessibilité de la destination choisie pour le joueur courant. Cette fonction utilise la fonction correspondante du plateau. Elle va retourner soit un chemin vide (case non accessible) soit le chemin à parcourir pour que le joueur courant accède à cette case. Dans le premier cas, l'utilisateur devra choisir une autre case destination. Dans le second cas, une fonction prédéfinie va parcourir le chemin pour créer une animation en terme d'affichage. Une fois cette animation terminée la fonction `finirTour()`, que vous devrez écrire, va

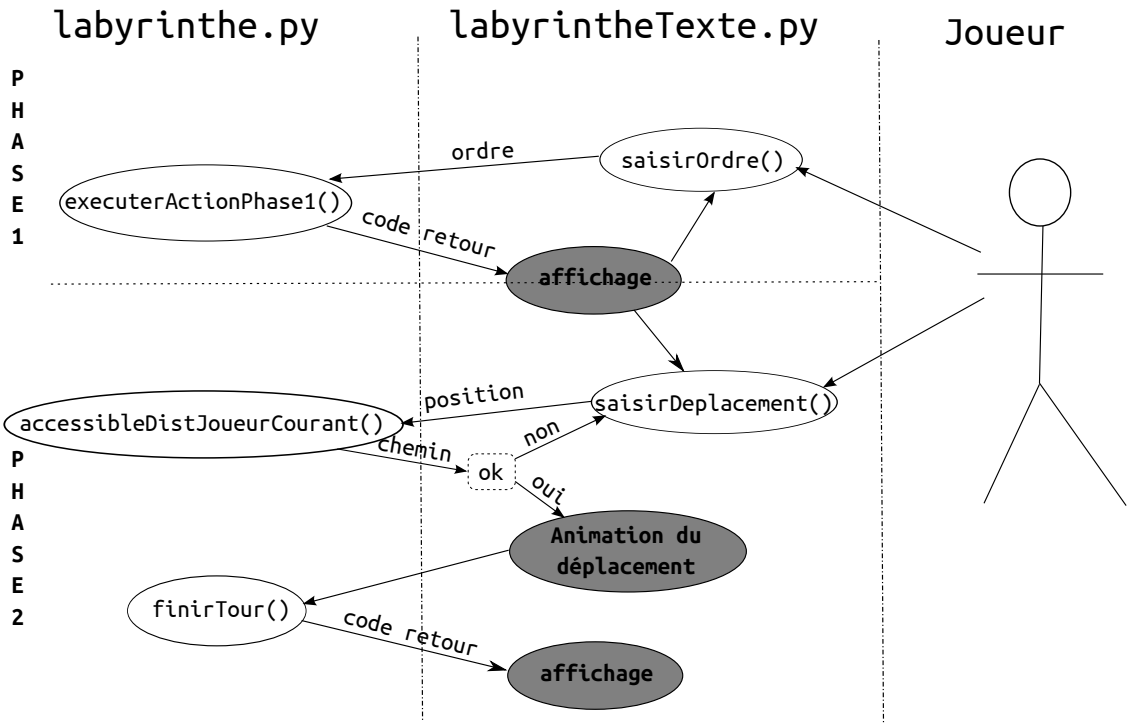


FIGURE 5 – Organisation générale du programme

— mettre à jour le labyrinthe en vérifiant si le joueur a trouvé son trésor sur la case d'arrivée,
— déterminer si la partie est terminée et
— passer au joueur suivant si la partie n'est pas terminée.

Les deux fonctions de saisie (`saisirOrdre()` et `saisirDeplacement()`) retournent des couples de valeurs dont voici la signification.

— `saisirOrdre()` retourne
  — soit `(-1,-1)` si l'ordre tapé par le joueur n'est pas correct ou inconnu
  — soit `('T','T')` lorsque le joueur choisit de tourner la carte amovible
  — soit un couple `(sens,rangee)` où `sens` est l'une des quatre lettres `'N'`, `'E'`, `'S'`, `'O'` indiquant une des quatre directions nord, est, sud ou ouest et `rangee` est l'un des trois caractères `'1'`, `'3'` ou `'5'`. Ce couple indique donc le sens et la rangée d'insertion de la carte amovible.
— `saisirDeplacement()` retourne un couple d'entiers donnant le numéro de la ligne et de la colonne de la case d'arrivée choisie par le joueur. Cette fonction peut retourner `(-1,-1)` si le joueur entre des valeurs non valides.

Pour la partie `labyrintheGraphique.py` le principe est le même, la seule différence est la manière dont les joueurs saisissent leurs commandes (via la souris), mais les mêmes fonctions du fichier `labyrinthe.py` sont utilisées, ce qui montre l'intérêt de bien séparer la partie mise à jour du labyrinthe, de la partie interaction avec l'utilisateur.

3.7 Le labyrinthe

Les fonctions de `labyrinthe.py` permettent de gérer le jeu du labyrinthe. Concernant la structure de données, la seule contrainte est que le plateau d'un labyrinthe doit être une matrice de cartes et doit contenir une liste de joueurs telle que définie dans la section 3.2. Il faut aussi maintenir les informations permettant de savoir dans quelle phase de jeu ce joueur se trouve (voir section 1).

Par ailleurs, le labyrinthe doit gérer les différentes phases de jeu que l'on va découper en trois fonctions. Attention les fonctions de ce fichier ne font aucune entrée-sortie (pas de saisies au clavier ou à la souris ni d'affichage). Elles permettent simplement de mettre à jour le labyrinthe en fonction des commandes qui auront été demandées.

- `executerActionPhase1()` : cette fonction s'occupe des opérations que le joueur peut effectuer lors de la phase 1 du jeu, c-à-d tourner la carte à jouer ou l'insérer dans une des colonnes ou lignes autorisées. Les paramètres de cette fonction, en plus du labyrinthe sont donc l'action à effectuer, tourner ou insérer. Pour le cas de l'insertion, l'action sera la direction où insérer la carte et le deuxième paramètre donnera le numéro la colonne ou de la ligne. Le code retour de la fonction permet de savoir si l'action demandée était possible et si tout s'est bien passé. Cette fonction effectue les mises à jour nécessaires dans le labyrinthe, notamment si le joueur a inséré la carte, il faut changer de phase de jeu.
- `accessibleDistJoueurCourant()` : cette fonction ne fait qu'appeler la fonction d'accessibilité du plateau avec les bons paramètres pour vérifier que le joueur courant peut accéder la case dont les coordonnées sont passées en paramètres. Si c'est le cas, la fonction retourne le chemin à suivre dans le labyrinthe **mais sans effectuer le déplacement**. Si la case n'est pas accessible, la fonction retourne `None`. Ce sera l'interface d'affichage qui s'occupera d'effectuer le déplacement en suivant le chemin fourni par votre fonction.
- `finirTour()` : cette fonction va prendre en compte toutes les modifications à effectuer une fois que le joueur courant aura terminé son déplacement :
  - vérifier s'il a trouvé son trésor,
  - vérifier s'il a gagné,
  - changer de joueur courant.

3.8 Le jeu du labyrinthe en mode texte

Dans le fichier `labyrintheTexte.py` seulement deux fonctions sont à implémenter. Chacune de ces fonctions permet d'interagir avec l'utilisateur pour recueillir les actions qu'il souhaite faire.

- `saisirOrdre(lmt)` permet de donner l'ordre de tourner la carte à jouer ou de l'insérer dans une des lignes ou colonnes. La fonction doit vérifier la validité de l'ordre.
- `saisirDeplacement(lmt)` permet au joueur courant de choisir sa case de destination. La fonction doit vérifier l'accessibilité de la case et retourner le chemin à prendre pour l'atteindre.

Les autres fonctions de ce fichier gèrent l'affichage en mode texte du jeu. La fonction `demarrer(lmt)` contient la boucle principale du jeu et utilise les fonctions que vous avez implémentées. **Vous n'avez pas à modifier ce code pour que votre jeu fonctionne**. Pour jouer au Labyrinthe, il vous suffira de taper la commande `./labyrintheTexte.py`.

3.9 Le jeu du labyrinthe en mode graphique

Dans le fichier `labyrintheGraphique.py` vous n'avez rien à implémenter. Ce script utilise la bibliothèque `Pygame` (`www.pygame.org`).
La méthode `demarrer(lmt)` joue un peu le même rôle que la fonction du même nom dans le labyrinthe en mode texte. La principale différence résidant dans le mode d'interaction de l'utilisateur qui utilise le principe de la programmation événementielle. Si toutes vos fonctions respectent les spécifications données, votre labyrinthe en mode graphique doit fonctionner sans modifier ce fichier.

Il vous suffira de taper `./labyrintheGraphique.py` pour jouer. Attention cependant à ce que le répertoire `images` soit bien présent et contiennent les images requises.

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 implémentera 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. Les versions objet de `labyrintheGraphique.py` et `labyrintheTexte.py` vous seront fournies au mois de janvier.
- 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 **18 janvier 2018** avant 23h55. Le **19 janvier** sera organisée une soutenance de 15 minutes au cours de laquelle vous présenterez votre travail et ferez une petite démonstration.