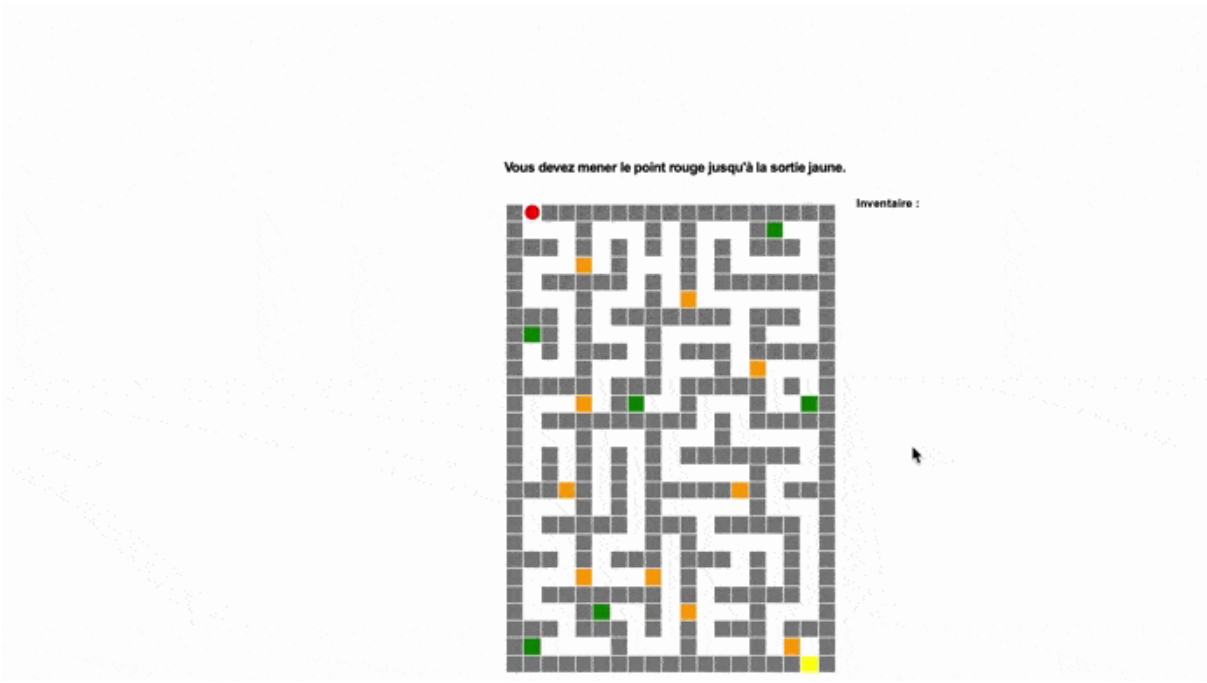
7.1.2. Présentation du projet

Lancelot entre dans le château au sommet du Python des Neiges, muni de son précieux sac de rangement et de sa torche fraîchement allumée aux feux de Beltane. Il doit trouver la statue de sainte Axerror, le chef-d'oeuvre de Gide de Rome, dit le « tyran malfaisant éternel .

Heureusement, pour l'aider dans sa quête, Merlin, son maître, lui a fourni un plan minutieux des salles et des couloirs du château. Ce plan lui sera fort utile, vu l'énormité du bâtiment, tant par sa taille que par le nombre de ses pièces !

Avant de partir, Merlin lui a donné la clef de la porte d'entrée du château et lui a prodigué moults conseils, dont celui de bien garder tous les objets qu'il trouvera lors de sa quête : ceux-ci lui permettront de répondre aux diverses énigmes que ne manqueront pas de poser les gardes postés devant les portes à l'intérieur du château.

Merlin a affirmé à son disciple que, s'il procède avec intelligence, sa quête sera satisfaite.



Ce projet, si vous le menez jusqu'au bout, va vous faire programmer un petit jeu du type jeu d'évasion (escape game) dans lequel le joueur commande au clavier les déplacements d'un personnage au sein d'un « château » représenté en plan. Le château est constitué de cases vides (pièces, couloirs), de murs, de portes, que le personnage ne pourra franchir qu'en répondant à des questions, d'objets à ramasser, qui l'aideront à trouver les réponses à ces questions et de la case de sortie / quête du château. Le but du jeu est d'atteindre cette dernière.

Le programme utilisera le module **turtle** comme interface graphique et comportera deux parties principales :

- le tracé du plan du château,
- la gestion du jeu sur le plan tracé (gestion des déplacements du personnage au sein du château, affichage des objets recueillis, gestion de l'ouverture des portes).

Les données nécessaires **(plan du château, objets, portes)** sont encodées dans **3 fichiers texte**. Vous disposerez d'un jeu de fichiers de données que nous vous proposons et devrez réaliser un programme de jeu qui met en œuvre ces données.

Vous pourrez ensuite si vous le souhaitez encoder votre propre château en préparant d'autres fichiers de données, qui tourneront avec le même programme. Votre château pourra aussi bien être du type labyrinthe (entièrement fait de couloirs étroits, sans portes) que du type escape game (un ensemble de pièces, entre lesquelles un personnage circule pour rassembler des objets lui permettant de répondre à des questions ou résoudre des énigmes).

Ce projet vous permettra d'écrire un code mais substantiel que ce que nous vous demandons ailleurs dans le cours, et va vous demander de manipuler de nombreux concepts vus tout au long du cours.

Il vous est demandé que votre programme puisse gérer divers jeux de données construits sur les mêmes principes que les nôtres. Cela permettra à chacun des participants de tester son programme sur les données proposées par d'autres.

NIVEAUX DE PROJET

Dans le cadre de ce projet, 4 niveaux de jeux peuvent être réalisés correspondant à des difficultés croissantes :

Niveau 1 : construction et affichage du plan du château. Ce niveau consiste à tracer correctement le plan du château à partir du fichier de données. La gestion du jeu après affichage du plan ne fait pas partie de ce niveau.

Niveau 2 : en plus du niveau 1, la gestion des déplacements au clavier, sans portes ni objets à ramasser. Ce niveau consiste à gérer les déplacements d'un personnage dans le plan construit au niveau 1. Ici on suppose qu'aucun objet ni porte n'est présent dans le plan : leur gestion n'est pas prise en compte.

- On déplace le personnage de case en case à l'aide des 4 flèches du clavier.
- Le personnage ne doit pas pouvoir traverser les murs ni sortir du plan.
- Si le personnage arrive sur la case quête / sortie du château, un message lui annonce qu'il a gagné.
- Option non demandée, si vous voulez aller plus loin : les cases déjà parcourues par le personnage peuvent être affichées dans une couleur spécifique, de façon à ce que la trace de son parcours soit conservée.

Niveau 3 : niveaux 1 + 2 + gestion des objets à ramasser. Dans ce niveau, il y a dans le labyrinthe des objets que le joueur doit collecter. Outre le fichier contenant le plan, un second fichier de données contient la liste des objets et la case où chaque objet se trouve.

- Les cases où se trouve un objet doivent apparaître dans une couleur spécifique.
- Lorsque le joueur se déplace sur une case contenant un objet, un message lui signale qu'il a trouvé un objet, et ce dernier s'ajoute à l'inventaire affiché en permanence à côté du plan. La case où était l'objet devient empty, puisque l'objet a été ramassé.
- Option non demandée, si vous voulez aller plus loin : on peut prévoir que la sortie du labyrinthe ne sera accessible qu'une fois tous les objets rassemblés.

Niveau 4 (escape game complet) : niveaux 1 + 2 + 3 + gestion des portes. Ce niveau consiste à ajouter dans la labyrinthe des portes que le joueur doit ouvrir en répondant à des questions. Le troisième fichier de données contient la liste des portes avec les questions et réponses associées.

- Les cases où se trouve une porte doivent apparaître dans une couleur spécifique.
- Lorsque le joueur tente d'accéder à une porte, la question associée lui est posée.
- S'il répond correctement, la porte s'ouvre et il peut alors la franchir. La case de la porte apparaît alors comme une case vide.
- S'il ne répond pas ou s'il donne une mauvaise réponse, la porte reste fermée et infranchissable.

7.1.3. En pratique

SYSTÈME DE COORDONNÉES

Le module **turtle** utilise un système de coordonnées dont **l'origine (0, 0)** est au centre de la fenêtre. L'axe des abscisses est orienté vers la droite, l'axe des ordonnées est orienté vers le haut.

Dans ce qui suit, nous parlerons de **pixel turtle** pour parler de coordonnées dans la fenêtre turtle. Nous utiliserons **les pixels turtle de (-240, -240) à (240, 240).**

Pour faciliter les discussions, nous vous demandons de numéroter les cases du plan du château :

⇒ en numérotant les lignes de haut en bas et les colonnes de gauche à droite,
⇒ en commençant la numérotation des lignes comme des colonnes à partir de 0 (convention habituelle de Python).

La case (0, 0) est donc située en haut et à gauche du plan et la case (0, 5) est située 5 cases à sa droite. la case (6, 5) sera la 6e case depuis la gauche, sur la 7e ligne depuis le haut.

Point important : Lorsque vous calculerez les coordonnées nécessaires pour tracer une case, souvenez-vous que le numéro de ligne augmente quand l'ordonnée « y » (pour la fenêtre turtle) diminue.

OCCUPATION DES ZONES DE LA FENÊTRE TURTLE

Vous aurez besoin de distinguer trois zones différentes dans la fenêtre turtle :

⇒ un bandeau en haut pour l'affichage d'annonces, qui peut par exemple faire 40 pixels turtle de hauteur, que vous pourrez définir par les coordonnées en pixels turtle du début de la ligne de texte,

⇒ en-dessous, une large zone à gauche pour l'affichage du plan, qu'il sera pratique de définir par les coordonnées de son coin inférieur gauche (abscisse et ordonnée minimales de la zone) et de son coin supérieur droit (abscisse et ordonnée maximales de la zone),

⇒ la partie restant à droite sera la colonne d'affichage de l'inventaire, que vous pourrez définir par les coordonnées en pixels turtle du début de la première ligne de texte de l'inventaire.

DESCRIPTION DES FICHIERS DE DONNÉES

Les données correspondant à un jeu figurent dans trois fichiers texte dont la structure est décrite ci-dessous.

Un fichier texte plan_chateau.txt qui contient les données du plan du château.

Il comporte un certain nombre de lignes, toutes de la même longueur, représentant une ligne de cases du plan. Chaque ligne du fichier est une suite d'entiers séparés par des espaces, où chaque entier représente une case. Les valeurs de ces entiers codent chaque nature de case :

- valeur 0 pour une case vide,
- valeur 1 pour un mur (infranchissable),
- valeur 2 pour la case de sortie/victoire,
- valeur 3 pour une porte qui sera franchissable en répondant à une question,
- valeur 4 pour une case contenant un objet à collecter.

Un fichier texte dico_objets.txt qui contient une liste d'objets associés aux cases sur lesquelles on les trouve.

Chaque ligne sera du type :

- (x, y), "objet"

Ainsi, chaque ligne contient :

- un couple d'entiers positifs ou nuls (numéro de ligne, numéro de colonne) indiquant la case où se trouve l'objet,
- puis une virgule et une espace,
- puis une chaînes de caractères décrivant l'objet.

Par exemple :

- (12, 3), "un oreiller magique"

signifiera que la case de coordonnées (12, 3) contient l'objet « un oreiller magique ».

Un fichier texte dico_portes.txt qui contient une liste de questions/réponses associées aux portes.

Chaque ligne sera du type :

- (x, y), ("question", "réponse")

Ainsi, chaque ligne contient :

un couple d'entiers positifs ou nuls (numéro de ligne, numéro de colonne) indiquant la case où se trouve la porte,

- puis une virgule et une espace,
- puis un couple de chaînes de caractères avec une question et une réponse.

Par exemple :

- (21, 12), ("Capitale de la Belgique ?", "Bruxelles")

signifiera que pour franchir la porte située en case (21, 12), il faudra répondre « Bruxelles » à la question « Capitale de la Belgique ? ».

CONSTANTES ET JEU DE DONNÉES TYPE

Pour vous aider à concevoir et à tester votre programme, un jeu de données comportant trois fichiers types sur le format spécifié plus haut est donné ici :

fichier plan_chateau.txt

fichier dico_objets.txt

fichier dico_portes.txt

Il s'agit d'un escape game très simple avec des questions portant sur Python. Nous vous suggérons de placer ces trois fichiers dans le même dossier que votre programme, pour pouvoir les appeler sans devoir décrire leur place dans l'arborescence.

Nous supposons aussi que la porte d'entrée du château est ouverte (elle est donc vue comme une case vide), qu'elle est en position (0, 1) et que le personnage s'y trouve initialement.

Un fichier CONFIGS.py vous est fourni : il donne les dimensions, couleurs et quelques constantes utilisées par notre programme de référence. Vous pouvez télécharger ce

fichier et copier/coller ces définitions dans votre script ou mieux placer ce fichier CONFIGS.py dans le même répertoire que votre script et ajouter à ce dernier la ligne

from CONFIGS import *

dont l'effet en pratique sera le même qu'un copier/coller.

ZONE_PLAN_MINI = (-240, -240) # Coin inférieur gauche de la zone d'affichage du plan

ZONE_PLAN_MAXI = (50, 200) # Coin supérieur droit de la zone d'affichage du plan

POINT_AFFICHAGE_ANNONCES = (-240, 240) # Point d'origine de l'affichage des annonces

POINT_AFFICHAGE_INVENTAIRE = (70, 210) # Point d'origine de l'affichage de l'inventaire

 # Les valeurs ci-dessous définissent les couleurs des cases du plan

COULEUR_CASES = 'white'

COULEUR_COULOIR = 'white'

COULEUR_MUR = 'grey'

COULEUR_OBJECTIF = 'yellow'

COULEUR_PORTE = 'orange'

COULEUR_OBJET = 'green'

COULEUR_VUE = 'wheat'

**COULEURS = [COULEUR_COULOIR, COULEUR_MUR, COULEUR_OBJECTIF, COULEUR_PORTE, **

COULEUR_OBJET, COULEUR_VUE]

COULEUR_EXTERIEUR = 'white'

 # Couleur et dimension du personnage

COULEUR_PERSONNAGE = 'red'

RATIO_PERSONNAGE = 0.9 # Rapport entre diamètre du personnage et dimension des cases

POSITION_DEPART = (0, 1) # Porte d'entrée du château

 # Désignation des fichiers de données à utiliser

fichier_plan = 'plan_chateau.txt'

fichier_questions = 'dico_portes.txt'

fichier_objets = 'dico_objets.txt'

Nous supposons donc que la fenêtre turtle est contenue dans un rectangle **(de coin inférieur gauche en (-240, -240) et de coin supérieur droit (240, 240)))** et est composée de 3 zones :

- la zone affichage du plan (de coin inférieur gauche en (-240, -240) et de coin supérieur droit (50, 200)),
- la zone annonces avec les textes qui seront affichés en (-240, 240) (chaque annonce devra effacer la précédente),
- la zone affichage de l'inventaire (de coin supérieur gauche en (70, 210) et de coin supérieur droit en (240, 210))

7.2.1. Niveau 1 : construction et affichage du plan du château

Le niveau 1 parle de lecture du plan à partir du fichier `chateau.txt`, de la construction de la matrice correspondante pour stocker ce plan, et de son affichage grâce au module `turtle`.

Pour cela différentes fonctions doivent être programmées.

FONCTION DE LECTURE DU FICHIER CONTENANT LE PLAN

Vous devez écrire une première fonction **`lire_matrice(fichier)`**, qui recevra en argument le nom d'un fichier texte contenant le plan à tracer. Elle ouvrira ce fichier et renverra en sortie une matrice, c'est-à-dire une liste de listes, soit une liste dont chaque élément sera lui-même une liste représentant une ligne horizontale de cases du plan.

Prenons l'exemple de ce plan comprenant 4 lignes et 6 colonnes avec les cases vides blanches, des murs gris, un objet orange et une porte verte :

Plan d'un chateau très simple

Il est représenté par le fichier suivant, composé d'entiers séparés par des espaces et de retours à la ligne :

```
1 0 1 1 1 1
1 0 0 0 0 1
1 0 4 0 0 1
1 1 1 3 1 1
```

La fonction de lecture du fichier renverra la liste de listes suivante :

```
[[1, 0, 1, 1, 1, 1], [1, 0, 0, 0, 0, 1], [1, 0, 4, 0, 0, 1], [1, 1, 1, 3, 1, 1]]
```

FONCTIONS D'AFFICHAGE DU PLAN

Pour tracer le plan du château à partir de la matrice obtenue, vous écrirez une fonction `afficher_plan(matrice)` utilisant le module `turtle` ; cette fonction recevra en entrée la matrice telle que décrite ci-dessus.

Pour réaliser cette fonction, nous vous conseillons de suivre les étapes suivantes :

1. Commencer par une fonction **calculer_pas(matrice)** qui calcule la dimension à donner aux cases pour que le plan tienne dans la zone de la fenêtre `turtle` que vous avez définie : diviser la largeur et la hauteur de la zone en question par le nombre de cases qu'elle doit accueillir, retenir la plus faible de ces deux valeurs.

   Par exemple, pour une zone turtle d'affichage pour le plan de 290 (-240 à 50) de large et 440 (-240 à 200) de haut, si le plan fait 20 cases en largeur et 44 cases en hauteur, les carrés auront une taille de 10 pour ne pas sortir de la zone (ici le souci, si la taille est supérieure à 10, sera la hauteur du plan).
2. Définir une fonction **coordonnees(case, pas)** qui calcule les coordonnées en *pixels turtle* du coin inférieur gauche d'une case définie par ses coordonnées (numéros de ligne et de colonne). Souvenez-vous que les lignes sont numérotées de haut en bas, alors que l'axe des coordonnées verticales va de bas en haut. Par exemple : avec un château de 44 lignes (lignes 0 à 43), la case inférieure gauche du château (c'est-à-dire la case `(43, 0)`) pour une sous-fenêtre turtle allant de `(-240, -240)` à `(50, 200)` vaudra `(-240, -240)`.
3. Définir une fonction **tracer_carre(dimension),** traçant un carré dont la dimension en *pixels turtle* est donnée en argument.
4. Définir une fonction **tracer_case(case, couleur, pas),** recevant en arguments un couple de coordonnées en indice dans la matrice contenant le plan, une couleur, et un pas (taille d'un côté) et qui va appeler la fonction `tracer_carre` pour tracer un carré d'une certaine couleur et taille à un certain endroit.
5. Définir enfin une fonction **afficher_plan(matrice),** qui va appeler la fonction `tracer_case` pour chaque ligne et chaque colonne du plan, par deux boucles imbriquées. Le principe est : pour chaque élément ligne de la matrice, pour chaque élément colonne de cet élément ligne, tracer une case à l'emplacement correspondant, dans une couleur correspondant à ce que dit la matrice.

CORPS DU PROGRAMME AU NIVEAU 1

Il restera alors à écrire la suite d'instructions qui va :

- fixer les valeurs nécessaires (noms des fichiers de données, couleurs des divers types de cases, points définissant les différentes zones de l'écran),
- appeler les différentes fonctions nécessaires pour créer la matrice et tracer le plan,

et à réaliser un premier script complet, contenant l'ensemble des éléments (docstrings, import, définitions des constantes, des fonctions et corps du programme) tel qu'expliqué dans le manuel des bonnes pratiques

7.2.2. Niveau 2 : gestion des déplacements

Bravo si vous avez terminé ce niveau 1. Vous pouvez passer au niveau 2. Ici, nous vous demandons d'ajouter à votre programme la gestion du déplacement d'un personnage.

Le personnage sera représenté par un petit rond (fonction prédéfinie turtle.dot). Il pourra se déplacer case par case dans les 4 directions (haut, bas, droite, gauche).

- Lorsque le personnage tentera un déplacement vers une case de mur ou vers l'extérieur du plan, rien ne se produira.
- Lorsqu'il tentera un déplacement vers une case vide, il avancera d'une case (en pratique, il s'agira de retracer la case de départ pour effacer le personnage, et de replacer ce dernier sur la case de destination).

Pour réaliser cette partie du code nous vous suggérons les parties suivantes :

FONCTION DE DÉPLACEMENT DU PERSONNAGE

Nous vous suggérons de définir une fonction principale de gestion des déplacements, qui sera assez simple pour le niveau 2, et que vous viendrez compléter si vous passez aux niveaux 3 (gestion des objets) et 4 (gestion des portes).

Pour le niveau 2, donc, nous avons besoin d'une fonction **deplacer(matrice, position, mouvement)** qui reçoit en arguments, outre la matrice représentant le château :
- `position` : un couple définissant la position où se trouve le personnage,
- `mouvement` : un couple définissant le mouvement demandé par le joueur.

Pour l'instant (niveau 2) cette fonction aura les effets suivants :

- Si le mouvement souhaité fait sortir le personnage des limites du plan, rien ne se passera.

- Si le mouvement souhaité mène le personnage sur un mur, rien ne se passera.
- Si le plan que vous utilisez comprend des objets ou des portes, vous traiterez les objets comme des cases vides et les portes comme des cases de mur.

Note

Au niveau 2, le traitement des portes et des objets trouvés ne sont pas traités, et sont « vus » comme des murs et des cases vides. Ces éléments seront traités correctement aux niveaux 3 et 4, expliqués plus loin.

SAISIE DES DÉPLACEMENTS

Pour gérer la commande du personnage au clavier, nous allons utiliser de la programmation événementielle. Le principe est d'associer à un événement un certain traitement. Ici, il faudra associer au fait de pousser sur une touche du clavier (les flèches de votre clavier) un traitement qui gère le déplacement associé de votre personnage.

Pour ceci, nous allons utiliser des fonctions `turtle` qui n'ont pas été vues dans le MOOC, et nous vous donnons donc ci-dessous les portions de code nécessaires :

Pour faire avancer le personnage grâce aux touches du clavier

1. Nous allons utiliser :

- la fonction `turtle.listen()`pour demander à Python « d'écouter » ce qui se passe sur le clavier,
- la fonction `turtle.onkeypress()`pour associer une touche du clavier au déclenchement d'une fonction (notons que cette fonction ne peut avoir de paramètres),
- la fonction `turtle.mainloop()`pour placer le programme en position d'attente d'une action.

Vous placerez donc, dans la partie traitement (corps) de votre programme, le bloc suivant, qui associe aux 4 flèches du clavier (qui sont identifiées dans `turtle` avec les noms (chaînes de caractères) `"Left"` (flèche vers la gauche), `"Right"` (flèche vers la droite), `"Up"` (flèche vers le haut) et `"Down"` (flèche vers le bas)) les 4 fonctions respectives `deplacer_gauche`, `deplacer_droite`, `deplacer_haut` et `deplacer_bas`. Nous verrons au point suivant que dans notre programme, il faudra définir chacune de ces 4 fonctions pour qu'elle déclenche le traitement requis chaque fois que la touche clavier correspondante est enfoncée. Ainsi, lorsqu'une des touches de flèche sera pressée (« onkeypress »), la fonction associée sera appelée.

```
turtle.listen()        # Déclenche l'écoute du clavier

turtle.onkeypress(deplacer_gauche,  "Left")        # Associe à la touche Left une fonction appelée deplacer_gauche

turtle.onkeypress(deplacer_droite,  "Right")

turtle.onkeypress(deplacer_haut,  "Up")

turtle.onkeypress(deplacer_bas,  "Down")

turtle.mainloop()        # Place le programme en position d'attente d'une action du joueur
```

2. Vous devrez alors définir les 4 fonctions **(deplacer_gauche(), deplacer_droite(), deplacer_haut(), deplacer_bas())** que déclenchent les 4 flèches du clavier. Mais si le joueur appuie de manière répétée et rapide sur une touche, ou en continu en gardant le doigt enfoncé, la fonction associée risque d'être lancée plusieurs fois en parallèle, ce qui poserait de grandes difficultés. Aussi, nous allons encadrer le bloc d'instructions de la fonction par des instructions visant à désactiver provisoirement la touche concernée. Pour cela, nous allons associer provisoirement la touche à la valeur **None** qui représente une absence de valeur (et donc de traitement associé).

   Vous placerez donc dans votre code une définition de fonction du type qui suit, en ce qui concerne la flèche gauche, et des fonctions similaires pour les 3 autres flèches :

```
def  deplacer_gauche():

        turtle.onkeypress(None,  "Left")      # Désactive la touche Left

        ...  # traitement associé à la flèche gauche appuyée par le joueur

        turtle.onkeypress(deplacer_gauche,  "Left")        # Réassocie la touche Left à la fonction deplacer_gauche
```

3. Il reste un point auquel il faut veiller sur le fonctionnement de la fonction **turtle.listen().**

   Nous aurons besoin, pour le niveau 4, que le programme pose une question au joueur lorsque celui-ci veut franchir une porte. Pour cela, nous vous proposerons d'utiliser la fonction **turtle.textinput()** qui affiche une fenêtre de saisie puis attend que le joueur y entre une chaîne de caractères. Cette fenêtre de saisie lance en fait son propre **turtle.listen()** associé à la fenêtre de saisie, ce qui interrompt le **turtle.listen()** que nous avons lancé auparavant. Il faut donc, après l'utilisation de **turtle.textinput(),** relancer le **turtle.listen().**

En pratique, retenez ceci : si vous utilisez la fonction **turtle.textinput(),** vous devrez placer une nouvelle ligne **turtle.listen()** juste après, pour que le programme continue à détecter l'appui sur les touches du clavier.

```
reponse = turtle.textinput("Question", dico_portes[case][0])
turtle.listen()
```

Note :

La méthode `onkeypress,` utilisée pour associer une fonction python à une touche clavier, ne permet malheureusement pas que cette fonction ait des paramètres. Ceci est ennuyeux d'autant plus que les 4 fonctions **(deplacer_gauche(), deplacer_droite(), deplacer_haut(), deplacer_bas())** que nous devons associer aux touches nécessitent d'avoir des informations comme le plan du château et la position du joueur. Exceptionnellement ici votre code devra donc manipuler les variables correspondantes comme des variables globales en ajoutant au début du code de chacune des 4 fonctions

```
global matrice, position
```

où `matrice, position` sont les variables contenant plan et position du personnage. Ajoutez d'autres variables globales si nécessaire à votre code.

7.2.3. Niveau 3 : collecte d'objets dans le labyrinthe

Pour le niveau 3, nous vous demandons d'ajouter au programme tel que rédigé au niveau 2 la gestion des objets, présents dans le château, que le personnage collecte quand il passe sur la case correspondante. Pour cela les éléments supplémentaires suivants devront être codés.

FONCTION DE LECTURE DES FICHIERS CONTENANT LES OBJETS

Vous aurez besoin d'une fonction **creer_dictionnaire_des_objets(fichier_des_objets)** créant un dictionnaire d'objets à partir du fichier correspondant, présenté précédemment. Cette fonction recevra en argument le nom du `fichier_des_objets`, et renverra un dictionnaire comportant :

- en clefs : les couples `(ligne, colonne)` désignant les cases où se trouvent les objets,
- en valeurs : les chaînes de caractères correspondant aux objets.

Ainsi le fichier d'objets suivant :

```
(12, 3), "un oreiller"
(3, 15), "une paire de ciseaux"
```

donnera le dictionnaire :

```
{(12,3): "un oreiller", (3, 15): "une paire de ciseaux"}
```

Note :

La lecture des données reçues du fichier des objets (et on verra plus loin, aussi le fichiers des portes) peut être facilitée par l'utilisation de verbe `eval`. Par exemple, ayant une chaîne de caractères `chaine = '(1, 2), "bonjour"'`, après l'instruction :

```
a, b = eval(chaine)
```

`a vaudra (1, 2) et b vaudra "bonjour".`

COMPLÉMENT À LA FONCTION DE DÉPLACEMENT DU PERSONNAGE

Au niveau 2, vous avez défini une fonction gérant le déplacement, qui examine si le personnage est envoyé vers une case vide ou vers un mur. Vous devez maintenant ajouter un autre cas à cette fonction, celui où le personnage est envoyé vers une case contenant un objet, et donc définir une fonction **ramasser_objet** :

- L'objet disparaîtra de la case (à la fois dans le plan et à l'affichage), qui devra donc prendre la couleur des cases vides.
- Le personnage avancera sur la case demandée.

- Une annonce du type « Vous avez trouvé : une clef à molette » s'affichera dans le bandeau d'affichage des annonces.
- L'objet s'ajoutera à l'inventaire des objets collectés affiché dans la colonne d'affichage de l'inventaire.

Pour cela, vous aurez sans doute besoin de passer de nouveaux paramètres à la fonction gérant le déplacement : la matrice contenant le plan (puisque vous serez amené à modifier la nature de la case contenant l'objet lorsque celui-ci sera ramassé) et l'ensemble contenant l'inventaire des objets ramassés (puisque l'objet sera ajouté à cet ensemble).

7.2.4. Niveau 4 : Le jeu escape game complet avec questions-réponses

Le niveau 4 demande de réaliser le jeu complet tel qu'annoncé en début d'énoncé. Pour cela les éléments supplémentaires suivants devront être codés.

FONCTION DE LECTURE DES FICHIERS CONTENANT LES QUESTIONS/RÉPONSES

Vous aurez besoin de lire le fichier contenant les questions-réponses associées aux portes. Cela peut être réalisé par la fonction de lecture du fichier des objets **(creer_dictionnaire_des_objets)** que vous aurez écrite au niveau précédent, puisque la structure des fichiers est presque la même.

Ainsi le fichier de questions/réponses suivant :

```
(12, 3) ("3 + 2 = ?", "5")
(3, 15) ("Quel était le prénom d'Henri IV ?", "Henri")
```

donnera le dictionnaire :

```
{(12,3): ("3 + 2 = ?", "5"),
 (3, 15): ("Quel était le prénom d'Henri IV ?", "Henri")}
```

COMPLÉMENT À LA FONCTION DE DÉPLACEMENT DU PERSONNAGE

Vous allez maintenant devoir ajouter le cas où la case de destination souhaitée est une porte. Vous aurez besoin d'une fonction **poser_question(matrice, case, mouvement)** pour :

- afficher dans le bandeau d'annonces `Cette porte est fermée.`,

- poser au joueur la question correspondant à l'emplacement de la porte et saisir sa réponse,
- si la réponse est bonne, remplacer la porte par une case vide, afficher dans le bandeau d'annonce que la porte s'ouvre, et avancer le personnage,
- si la réponse est mauvaise, l'annoncer et ne pas déplacer le personnage.

Pour poser une question, vous pouvez utiliser la fonction **turtle.textinput().** Elle prend en argument 2 chaînes de caractères, une qui sera le titre de la fenêtre d'input (par exemple « Question ») et une autre qui sera le texte affiché dans la fenêtre d'input (la question associée à la porte).

Comme cela a été signalé plus haut, utiliser la fonction **turtle.textinput()** interrompt l'écoute du clavier déclenchée par **turtle.listen(),** et vous devrez placer sur la ligne suivante une nouvelle instruction **turtle.listen()** pour recommencer à surveiller le clavier.