Apprentissage de la programmation

Rapport technique

Introduction : La technologie fait aujourd'hui partie de notre quotidien, si bien que les nouvelles générations y sont sensibilisées dès leur plus jeune âge, sans pour autant en saisir l'essence ni comprendre leur fonctionnement.

Problématique : En tant qu'électroniciens aguerris, la société ***Solec Group®*** nous a confié la mission de réaliser pour un public de jeunes adolescents un jeu robotisé constitué d'un damier et d'un robot mobile, dont le déplacement est contrôlé au bon vouloir de l'utilisateur. Comment donc enseigner à de jeunes collégiens la logique de la programmation, de la manière la plus pédagogique possible tout en restant ludique ?

Avant-propos : Le projet initial faisait intervenir une table de traçage correspondant au robot, un ordinateur et une interface graphique permettant de piloter le robot. Compte-tenu des conditions actuelles ayant entraîné la fermeture de l'école, nous n'avons pas eu accès à la table de traçage et au matériel électronique habituellement mis à disposition au LEnsE.

Nous avons donc pris le parti suivant : nous n'avons pas développé la partie relative à l'électronique et *a fortiori* à la communication entre le vrai robot et l'ordinateur. À la place, nous avons choisi de développer uniquement le code informatique et l'interface graphique. Ceux-ci ont pour but de simuler ce qu'aurait été le fonctionnement du robot réel. Ainsi, la table de traçage est simulée par un damier visible sur l'interface graphique, sur laquelle on peut piloter le robot au moyen de différentes instructions.

L'interface est donc, en ce sens, beaucoup plus développée ici qu'elle ne l'aurait été si nous avions pu utiliser la machine.

Sommaire

I. Présentation du prototype et schéma fonctionnel

1. Présentation du prototype

Le but de notre projet est de mettre en place un outil pédagogique permettant à des enfants de se familiariser avec les bases de la logique inhérente à la programmation informatique. Grâce à une interface homme-machine accessible sur n'importe quel ordinateur doté du logiciel de programmation Python et de son extension TkInter, à la manière d'un jeu, les enfants pourront guider un robot sur un damier, en lui indiquant, par exemple, le trajet pas à pas et l'existence éventuelle d'obstacles sur celui-ci.

Comme déjà évoqué en introduction, le lien avec la machine ne sera pas développé ici. Le but étant uniquement de développer un code complet et une interface adaptée.

2. Schéma fonctionnel

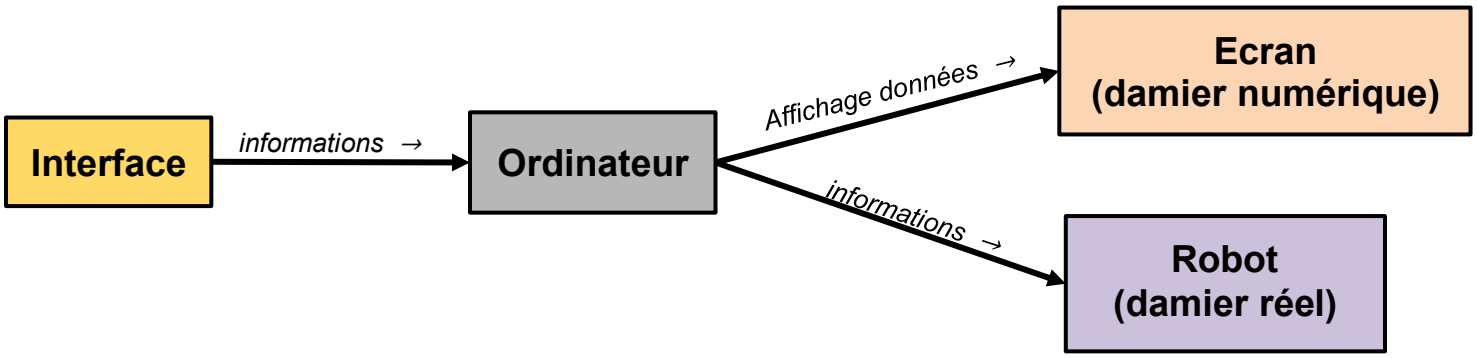


Figure I.2.a. : Schématisation de l'architecture générale du projet.

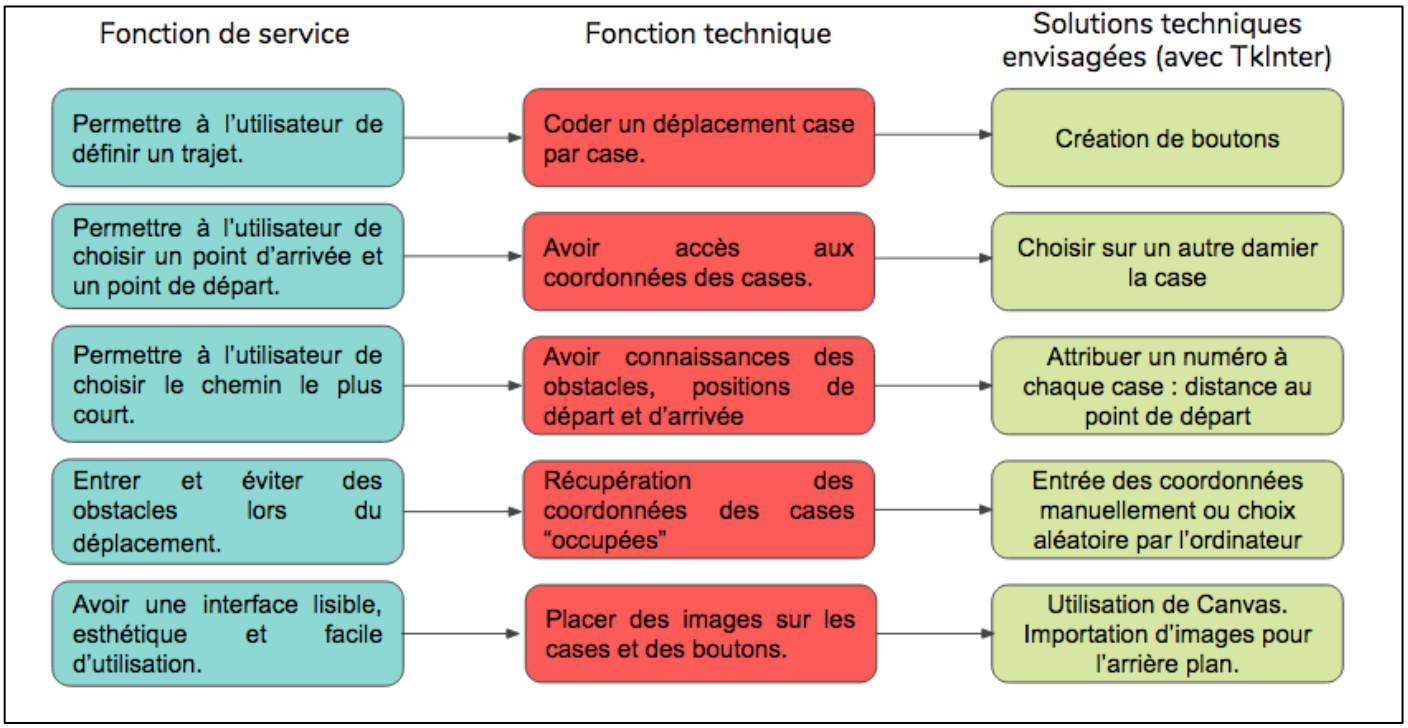


Figure I.2.b. : Schéma fonctionnel du prototype.

II. Tutoriel d'utilisation, explication des simulations pas à pas

Nous avons séparé notre jeu en quatre fonctions de base. Nous nous proposons ici de les développer une à une, à la manière d'un tutoriel. En effet, ce rapport technique a aussi pour vocation de servir de mode d'emploi aux futurs utilisateurs.

L'interface de jeu est la suivante :

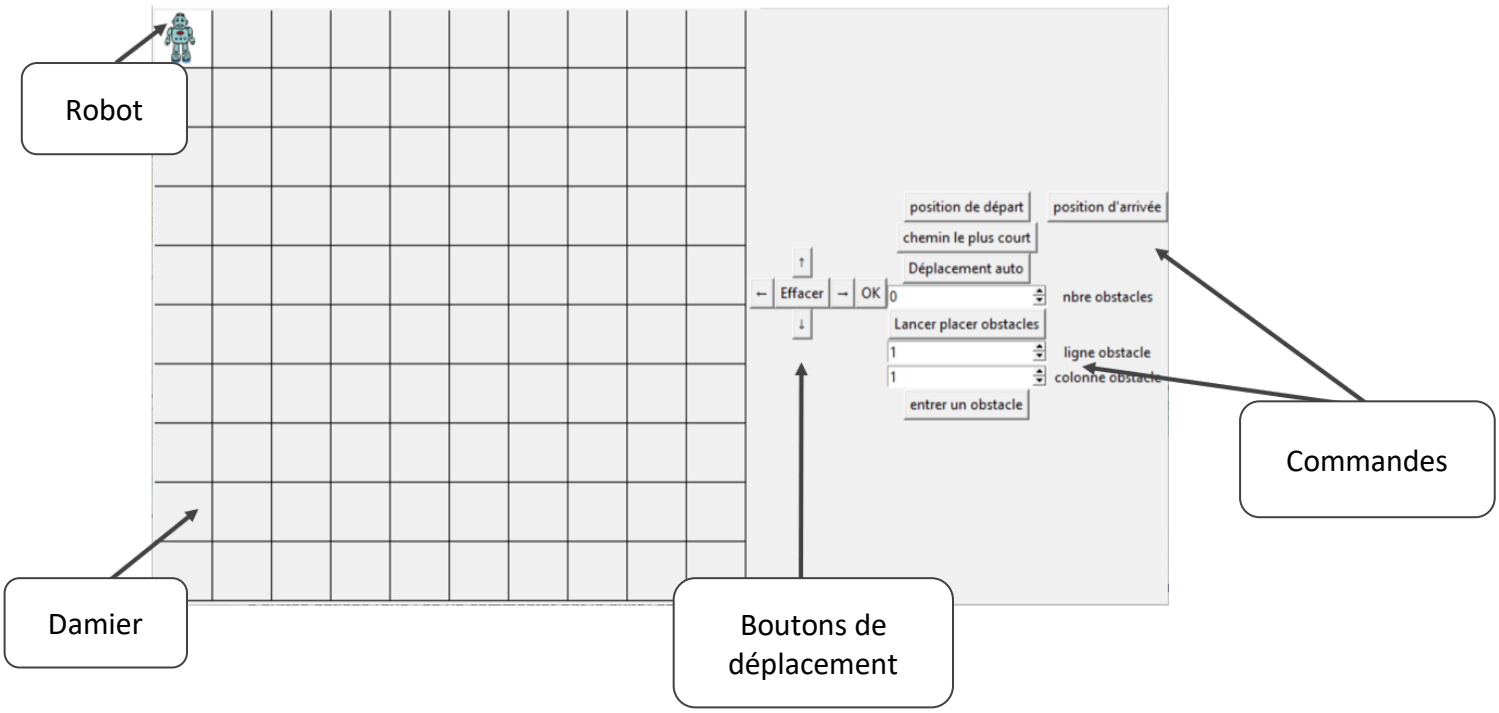


Figure II. : Aperçu de l'interface graphique qui s'ouvre lorsque l'on exécute le programme sous Python.

1. Fonction 1 : Choisir un point de départ et d'arrivée

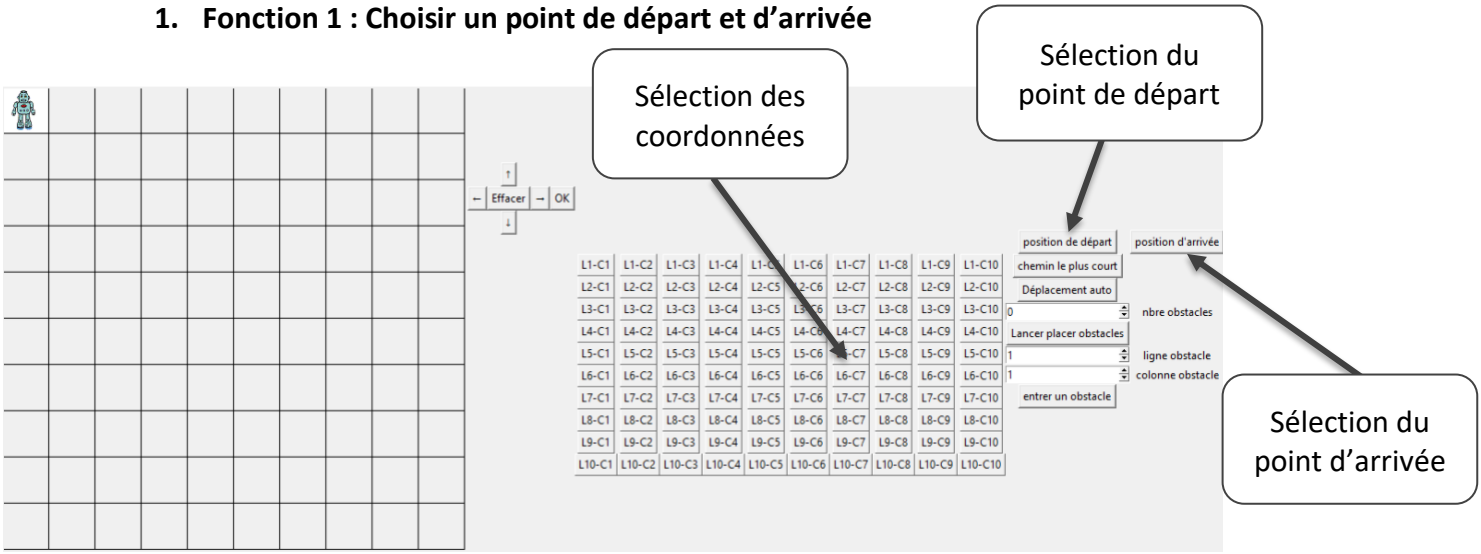


Figure II.1. : Aperçu de l'interface graphique lorsque l'on choisit un point de départ et un point d'arrivée pour le robot.

Le point de départ correspond à la position du robot choisie. Le point d'arrivée est matérialisé par un drapeau sur la case choisie. Ces deux commandes sont accessibles via les boutons "Position de départ" et "Position d'arrivée".

L'utilisateur est ensuite libre de :

1) Constituer le trajet manuellement à l'aide des flèches, comme détaillé dans la fonction 2.
2) Demander au programme de déterminer le trajet le plus court entre les deux points et de le déplacer en conséquence, comme nous le verrons dans la fonction 3.

2. Fonction 2 : Définir un trajet

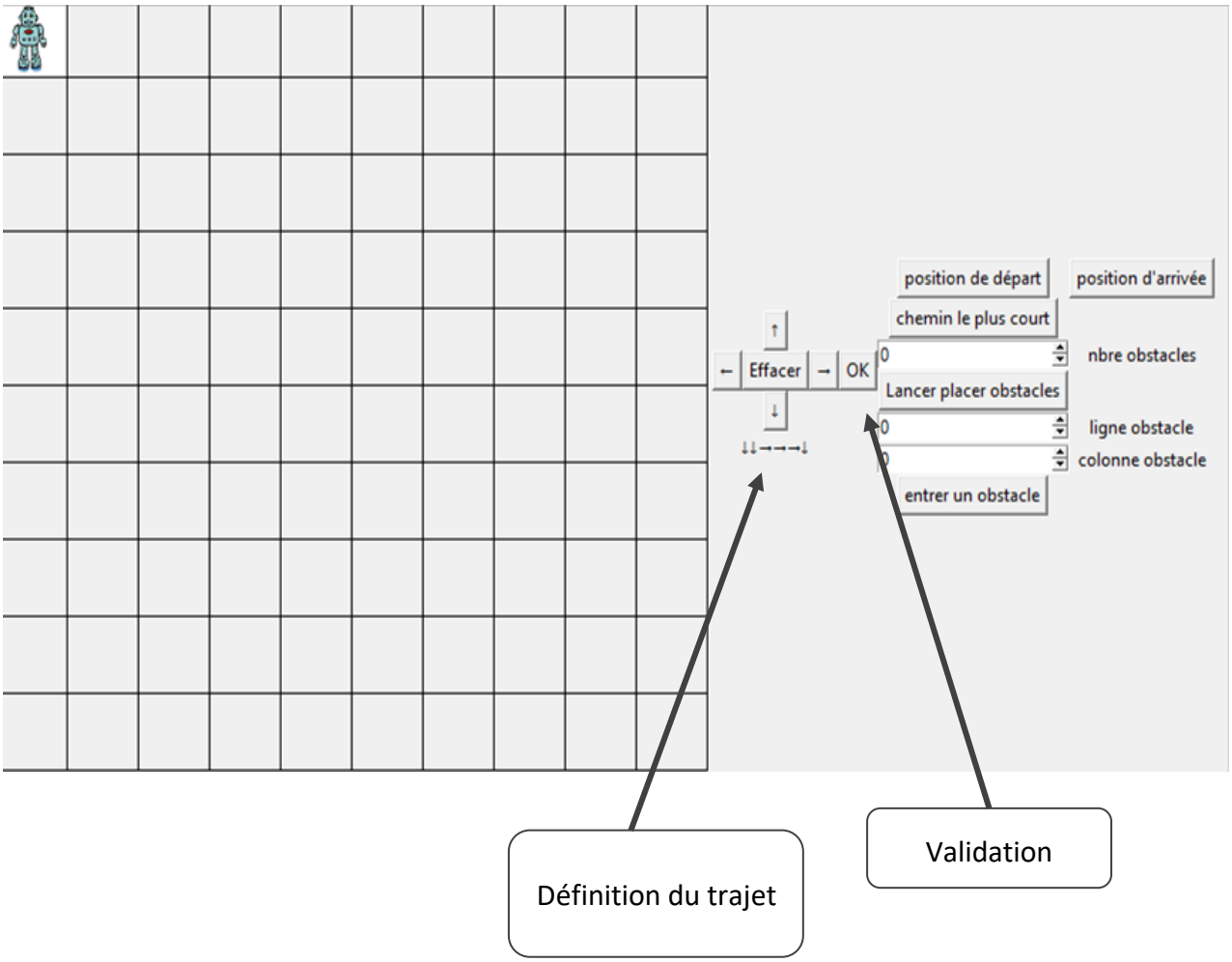


Figure II.2. : Aperçu de l'interface graphique lorsque l'on définit le trajet à suivre.

Au moyen des flèches de commande, l'utilisateur peut définir le trajet à suivre par le robot. Une fois le trajet défini, il lui suffit de cliquer sur "OK" pour lancer le robot.
Par défaut, la position de départ du robot est en haut à gauche du damier. Nous avons vu dans la section précédente qu'il existe une fonction permettant à l'utilisateur de choisir un point de départ et un point d'arrivée pour le robot n'importe où sur le damier.

3. Fonction 3 : Définir et éviter les obstacles

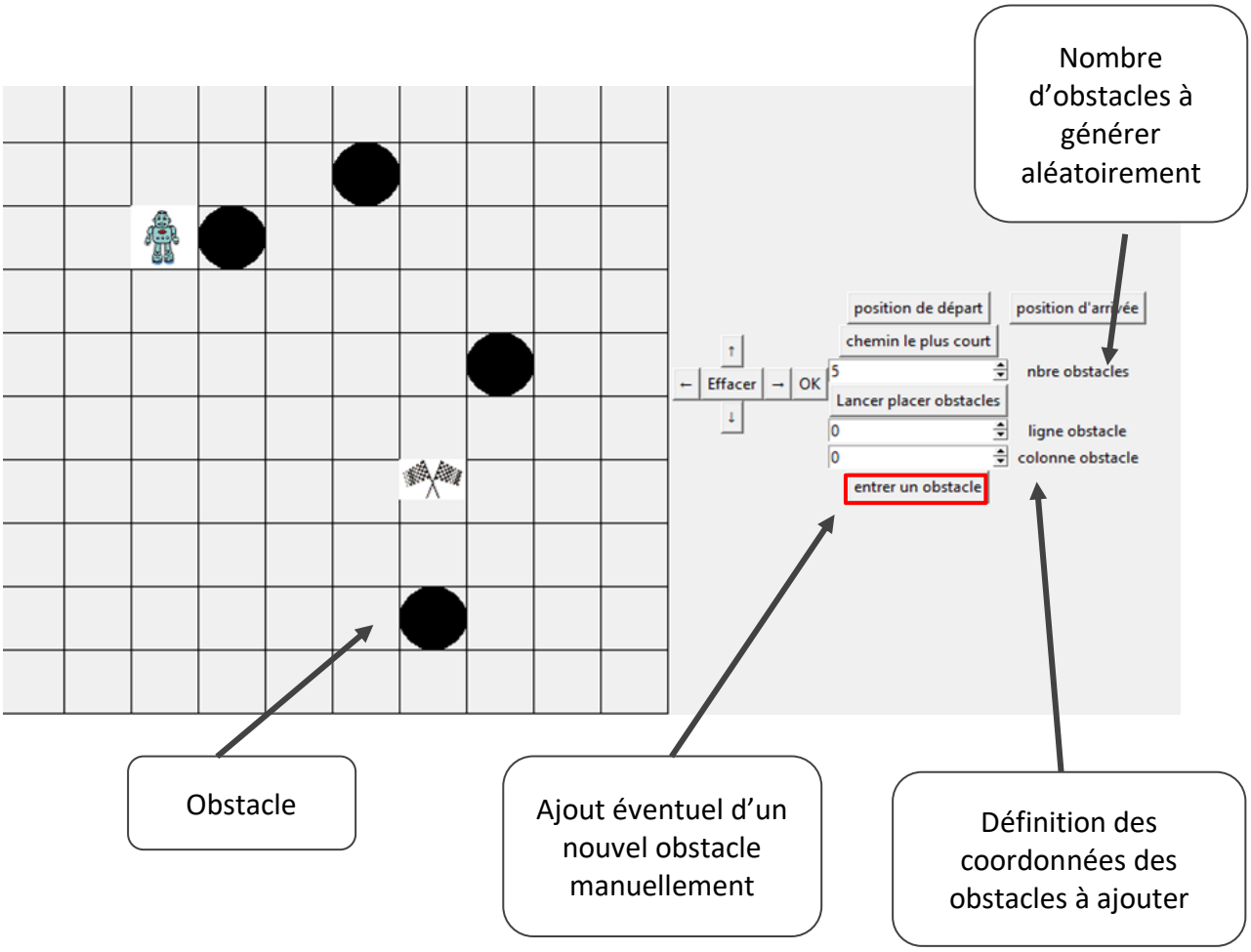


Figure II.3. : Aperçu de l'interface graphique lorsque l'on positionne des obstacles sur le parcours du robot.

L'utilisateur peut choisir s'il décide de placer des obstacles sur le chemin et si oui, à quelle position ils se trouvent. Ensuite, le programme permet au robot d'éviter ces obstacles lors de son déplacement.
Pour ajouter des obstacles, nous avons développé deux modes possibles.
Le mode **manuel** permet à l'utilisateur, à l'aide de deux commandes, une pour la ligne et une pour la colonne, d'indiquer la place voulue pour l'obstacle (la première ligne est la ligne 1 et la première colonne est la colonne 1).
Le mode **aléatoire** permet à l'utilisateur d'indiquer le nombre d'obstacles voulus (entre 1 et 10) et l'ordinateur les génère aléatoirement sur le damier en les matérialisant.

Après la mise en place d'obstacles, deux situations sont possibles :

1) Soit l'utilisateur rentre manuellement le trajet à suivre avec les flèches (en tenant ou non compte de l'éventuel point d'arrivée choisi).
2) Soit il demande au programme de suivre le chemin le plus court (fonction 4), ce qui implique d'avoir ici défini un point d'arrivée, tout en évitant les obstacles présents sur le trajet.

4. Fonction 4 : Sélectionner le chemin le plus court

Le programme permet, par une simple commande qu'il faut sélectionner, de déplacer non plus manuellement mais automatiquement le robot en suivant le chemin le plus court. Cela suppose donc d'avoir indiqué au préalable les positions de départ et d'arrivée, ainsi que les obstacles.

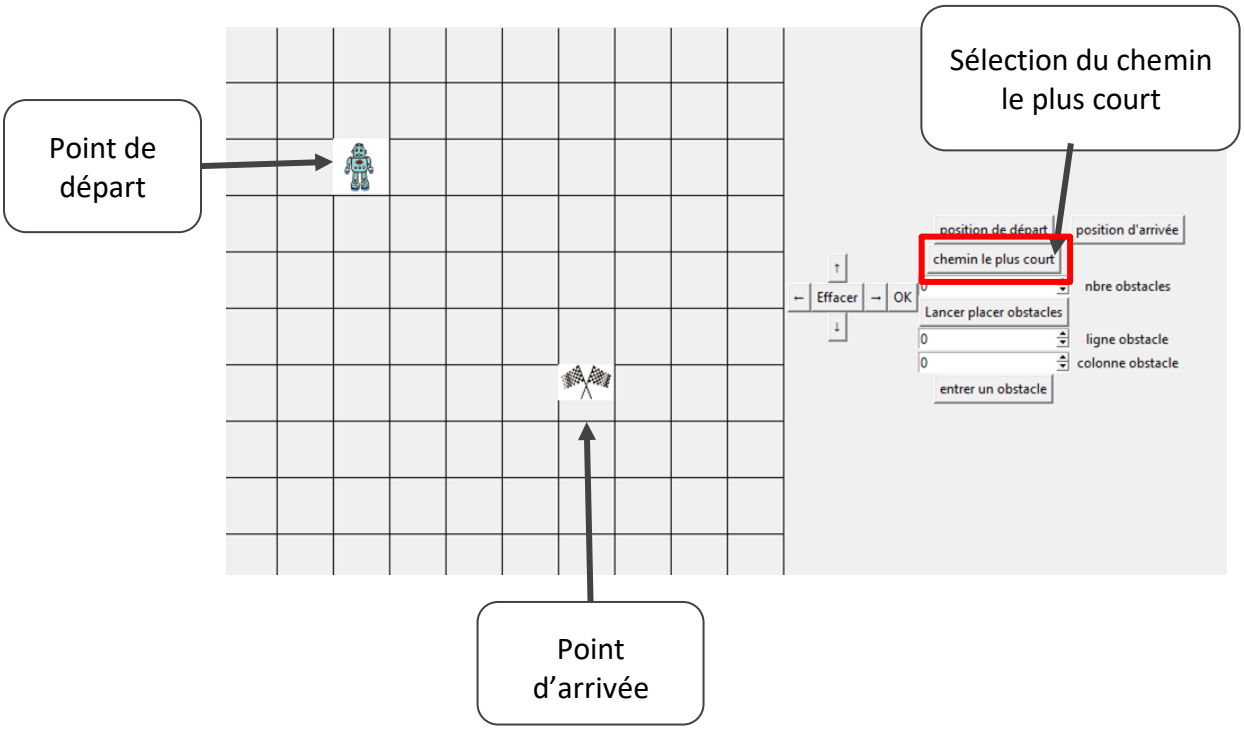


Figure II.4. : Aperçu de l'interface graphique lorsque l'on utilise l'option du chemin le plus court.

Il suffira de cliquer sur "Chemin le plus court" afin de lancer le déplacement du robot.

III. Explication des différents codes

Pour toutes les fonctions que nous allons expliciter ici, nous utilisons la bibliothèque Python TkInter.

1. Création du damier

Il s'agit de la fonction de base que nous utilisons ensuite dans toutes les autres fonctions. Nous avons décidé que pour que le robot se déplace, le plus simple est de recréer le damier en entier à chaque fois en positionnant simplement le robot sur une autre case.

```
def creergrille(ligneRobot, colonneRobot):
    platforme.delete(ALL)
    n = taille/nbLignes
    for i in range (nbLignes):
        for j in range(nbColonnes) :
            platforme.create_rectangle(i*n,j*n,i*n+n,j*n+n)
    robot = [ligneRobot*50,colonneRobot*50]
    platforme.create_image(robot[1],robot[0], anchor = NW, image=robotimage)
    platforme.update()
```

Figure III.1.a. : Code correspondant à la création du damier vide avec le robot.

Dans cette fonction, nous commençons par supprimer la grille déjà existante puis nous recréons les différents rectangles formant le damier (c'est le rôle de la double boucle `for`). Puis, en fonction de la position du robot prise en argument de la fonction, nous positionnons le robot sur la grille.
Dans cette fonction, il faut faire attention à définir les coordonnées du robot et des carrés en pixels.

Lors de l'exécution du programme, une grille comme celle-ci (figure ci-contre) s'ouvre. Chaque case est formée d'un carré distinct des autres. Ainsi chaque case est un élément différent. C'est donc plus simple pour positionner le robot sur la case de notre choix.
L'utilisateur ne peut pas interagir directement avec cette fenêtre. Cela permet simplement de visualiser les déplacements du robot.

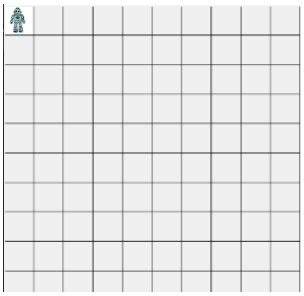


Figure III.1.b. : Aperçu de l'interface graphique obtenue après l'exécution du précédent code.

2. Design du damier

Le jeu est – rappelons-le – destiné à des enfants, nous avons donc essayé d'égayer l'interface graphique au moyen de sympathiques petits items disposés çà et là sur la surface du damier. On peut voir ci-dessous un champ d'arbres, de rochers et de champignons en tous genres sur lequel le robot va pouvoir vagabonder au bon gré du joueur. Ces images servent uniquement de « fond d'écran » et n'ont pas d'incidence sur le parcours du robot. Il s'agit simplement de mettre un peu de couleur dans le jeu.

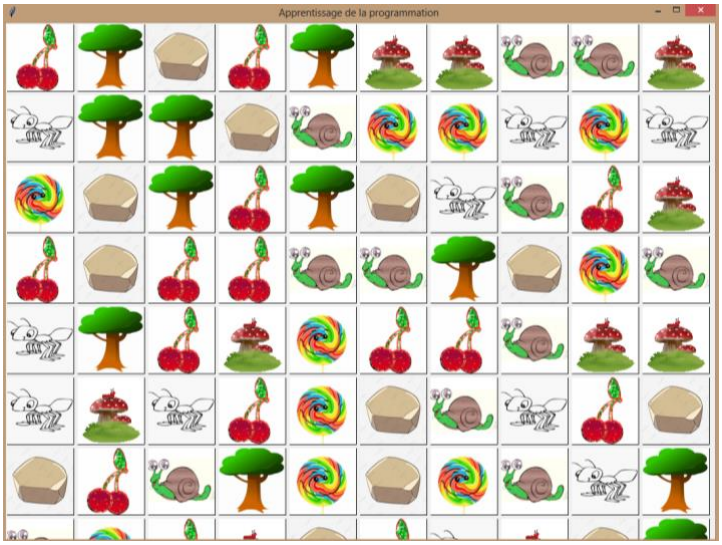


Figure III.2.a. : Aperçu de l'interface graphique avec ajout d'images sur les cases.

Nous avons d'abord essayé de créer un unique `canvas` avec TkInter, c'est-à-dire un tableau inclus dans une fenêtre, dans lequel on peut positionner des objets (les images). Nous avons pu créer un quadrillage et faire bouger le robot dessus. Toutefois, à chaque fois que l'on essayait d'attribuer une image à une case, celle-ci s'affichait puis disparaissait à mesure que la boucle `for` avançait. En fait, il se trouve que dans un `canvas` on ne peut afficher qu'une seule image .jpeg (ou .gif) à la fois. On peut afficher en permanence des objets déjà inclus dans la librairie TkInter : lignes et formes géométriques simples, par exemple les disques qui servent à représenter les obstacles …

Nous avons alors créé 10 x 10 `canvas` différents correctement positionnés dans la fenêtre pour ne plus être confrontés au problème. Ensuite à chaque `canvas` est attribuée aléatoirement une image parmi sept autres savamment choisies pour plaire aux enfants. Les images ont été préalablement chargées dans une liste et redimensionnées de façon à ce qu'elles aient toutes la même taille. Les étapes de la construction de ce tableau d'image sont décrites par la fonction Python `créergrille` ci-après :

```
from PIL import ImageTk
from tkinter import *
from time import sleep
from random import *
from PIL import Image

nbLignes = 10
nbColonnes = 10
taille = 100
ligneDepart=0
colonneDepart=0
ligneArrivee=0
colonneArrivee=0

fenetre = Tk()
fenetre.title("apprentissage de la programmation")

robotimage = PhotoImage(file="imagerobot2.gif",master=fenetre)
img = PhotoImage(file='ppt.gif')

image1= PhotoImage(file='image1.gif')
image2= PhotoImage(file='image2.gif')
image3= PhotoImage(file='image3.gif')
image4 = PhotoImage(file='image4.gif')
image5 = PhotoImage(file='image5.gif')
image6 = PhotoImage(file='image6.gif')
image7 = PhotoImage(file='image7.gif')

Limages=[image1,image2,image3,image4,image5,image6,image7]

def creergrille(ligneRobot, colonneRobot):

    Lplatforme=[[0]*100]*100
    for i in range (nbLignes):
        for j in range(nbColonnes) :

            Lplatforme[i][j] = Canvas(fenetre, width = taille, heigh = taille) #création d'un espace de dessin de 50 par 50
            Lplatforme[i][j].create_rectangle(0, 0, taille,taille,fill ='white', width =1)
            Lplatforme[i][j].grid(column = j,row = i) # position de la fenêtre

            Lplatforme[i][j].create_image(50,50, anchor=CENTER, image=Limages[randint(0,len(Limages)-1)], state='normal')

creergrille(0,0)

fenetre.mainloop()
```

Figure III.2.b. : Code permettant d'insérer les images dans les différentes cases du damier.

On remarquera, à la ligne 49, que les images sont placées de façon aléatoire dans les cases grâce à la fonction `randint`, fonction qui renvoie tout simplement un entier aléatoire (en l'occurrence entre 1 et 7) et que l'on va placer en indice de la liste des images.

Malheureusement, pendant que deux membres de l'équipe essayaient de résoudre le problème de l'affichage d'images sur un seul `canvas`, les deux autres membres avaient largement avancé sur les autres parties du codes (que vous allez découvrir plus loin) et prenant comme repère initial un seul `canvas`.
Le manque de temps ne nous a malheureusement pas permis de recombiner les deux implémentations, c'est pourquoi sur les simulations finales, on ne voit pas ces petites images qui auraient dû illustrer le fond du damier.

3. Choisir un point de départ et d'arrivée

Cette fonction permet à l'utilisateur de choisir sur une grille la position de départ du robot. Cela permet en plus de choisir la position à laquelle il veut que le robot aille au cours du déplacement. Une fois cette position d'arrivée choisie, un drapeau est positionné sur cette case.

Nous avons deux boutons dans l'interface graphique permettant de choisir ces positions. Voici le code qui définit les boutons dans l'interface graphique.

Figure III.3.a. : Aperçu des boutons ajoutés sur l'interface.

```
222    Button(autresboutons,text = "position de départ",command = positiondepart).grid(column = 0, row = 0)
223    Button(autresboutons,text = "position d'arrivée",command = positionarrivee).grid(column = 1, row = 0)
```

Figure III.3.b. : Code permettant d'ajouter les boutons de sélection sur l'interface.

Quand l'utilisateur appuie sur l'un de ces boutons, un rectangle composé de 10 x 10 cases (figure ci-contre) s'ouvre. L'utilisateur n'a plus qu'à sélectionner l'un de ces boutons pour que le robot (ou le drapeau) aille sur la case souhaitée. De plus, dès que l'utilisateur a sélectionné une case, ce rectangle se ferme à nouveau.

L0-C0	L0-C1	L0-C2	L0-C3	L0-C4	L0-C5	L0-C6	L0-C7	L0-C8	L0-C9
L1-C0	L1-C1	L1-C2	L1-C3	L1-C4	L1-C5	L1-C6	L1-C7	L1-C8	L1-C9
L2-C0	L2-C1	L2-C2	L2-C3	L2-C4	L2-C5	L2-C6	L2-C7	L2-C8	L2-C9
L3-C0	L3-C1	L3-C2	L3-C3	L3-C4	L3-C5	L3-C6	L3-C7	L3-C8	L3-C9
L4-C0	L4-C1	L4-C2	L4-C3	L4-C4	L4-C5	L4-C6	L4-C7	L4-C8	L4-C9
L5-C0	L5-C1	L5-C2	L5-C3	L5-C4	L5-C5	L5-C6	L5-C7	L5-C8	L5-C9
L6-C0	L6-C1	L6-C2	L6-C3	L6-C4	L6-C5	L6-C6	L6-C7	L6-C8	L6-C9
L7-C0	L7-C1	L7-C2	L7-C3	L7-C4	L7-C5	L7-C6	L7-C7	L7-C8	L7-C9
L8-C0	L8-C1	L8-C2	L8-C3	L8-C4	L8-C5	L8-C6	L8-C7	L8-C8	L8-C9
L9-C0	L9-C1	L9-C2	L9-C3	L9-C4	L9-C5	L9-C6	L9-C7	L9-C8	L9-C9

Figure III.3.c. : Aperçu de l'interface lors de la sélection des positions de départ et d'arrivée.

Nous allons maintenant expliquer les différents codes derrière ces fonctions.

Pour le bouton associé aux coordonnées de départ, nous avons créé deux fonctions qui fonctionnent conjointement. Il s'agit du même type de fonction pour la position d'arrivée.

La fonction qui s'exécute en premier est la fonction `positiondepart` (figure ci-dessous). Cette fonction permet de renvoyer dans la deuxième fonction les coordonnées de la position de départ. Nous avons eu beaucoup de difficultés à créer cette fonction et nous avons dû regarder beaucoup de documentation. C'est en consultant l'ouvrage *Apprendre la programmation par le jeu* que nous avons trouvé comment faire pour que, lorsque l'on appuie sur l'une des cases, nous arrivons à garder les coordonnées en mémoire.

```
def coordonneeDepart(l1,c1):
    global ligneDepart, colonneDepart
    ligneDepart=l1
    colonneDepart=c1
    position.destroy()
    creergrille(l1,c1)

def positiondepart(*arg):#définition grille pour séléctionner position d'arrivée
    global position
    position = Frame(boutonDeplacement)
    position.grid(column = 4, row = 5)
    global nbLignes
    global nbColonnes
    for ligne in range (nbLignes):
        for colonne in range (nbColonnes):
            exec("Button(position, text='L%s-C%s' % ({}, {}), borderwidth=1,command=lambda:coordonneeDepart({}, {})).grid(row={}+3, column={}+3)".format(ligne, colonne,ligne, colonne,ligne, colonne))
```

Figure III.3.d. : Code permettant de sélectionner la position de départ du robot.

Cette manière de faire est très intuitive pour l'utilisateur car il n'a pas besoin de connaître le système de coordonnées utilisé pour pouvoir positionner où il veut son robot.

4. Définir un trajet

Cette fonction permet à l'utilisateur de choisir manuellement le chemin qu'il veut que le robot suive. Par exemple, s'il appuie sur la flèche vers la gauche puis sur « OK », le robot se déplacera d'une case vers la gauche. Il s'agit d'une fonction très intuitive pour les utilisateurs. La personne voit ainsi directement l'impact de ce qu'elle demande sur le déplacement du robot.

Pour cela, nous avons créé les six boutons (figure ci-contre et code ci-dessous). Chaque bouton est relié à une fonction. Ainsi lorsque l'utilisateur clique sur le bouton « OK », le robot effectue le déplacement demandé à l'aide des autres boutons.

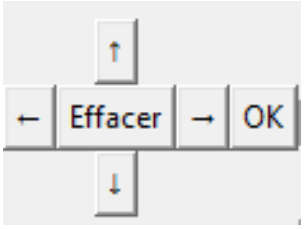


Figure III.4.a. : Aperçu des boutons ajoutés sur l'interface.

```
Button(boutonDeplacement,text = "↑",command = deplacementhaut).grid(column = 1, row = 0)
Button(boutonDeplacement,text = "→",command = deplacementdroite).grid(column = 2, row = 1)
Button(boutonDeplacement,text = "←",command = deplacementgauche).grid(column = 0, row = 1)
Button(boutonDeplacement,text = "↓",command = deplacementbas).grid(column = 1, row = 2)
Button(boutonDeplacement,text = "Effacer",command = findeplacement).grid(column = 1, row = 1)
Button(boutonDeplacement,text = "OK",command = lancerDeplacement).grid(column = 3, row = 1)
```

Figure III.4.b. : Code permettant d'ajouter les boutons de déplacement sur l'interface.

L'utilisateur peut même supprimer le dernier mouvement. De plus, nous avons ajouté une boucle de contrôle indiquant à l'utilisateur quand un mouvement est impossible pour le robot (par exemple, quand il y a un obstacle ou s'il sort du plateau).

Nous allons maintenant expliquer les différents codes derrière ces boutons.

```
248  def deplacementhaut(*arg):
249      global chemin, cheminbis
250      chemin += "↑"
251      cheminbis.append(8)
252      sv.set(chemin)
253  def deplacementbas(*arg):
254      global chemin, cheminbis
255      chemin += "↓"
256      cheminbis.append(2)
257      sv.set(chemin)
258  def deplacementdroite(*arg):
259      global chemin, cheminbis
260      chemin += "→"
261      cheminbis.append(6)
262      sv.set(chemin)
263  def deplacementgauche(*arg):
264      global chemin, cheminbis
265      chemin += "←"
266      cheminbis.append(4)
267      sv.set(chemin)
268  def findeplacement(*arg):
269      global chemin, cheminbis
270      chemin = chemin[:-1]
271      cheminbis = cheminbis[:-1]
272      sv.set(chemin)
```

Figure III.4.d. : Code permettant de commander les déplacements dans les différentes directions de l'espace 2D.

Nous avons ici les fonctions associées aux boutons de déplacement.
Nous avons créé, au début, une chaîne de caractère `chemin` et ainsi, à chaque fois que l'utilisateur appuie sur l'un des boutons de déplacement, nous ajoutons le caractère associé à ce déplacement à `chemin`.
Pour supprimer le dernier mouvement, nous venons supprimer le dernier caractère dans `chemin`. Cette chaîne `chemin` est celle que nous allons afficher en-dessous des boutons de commande, pour que l'utilisateur voit ce qu'il a fait, comme indiqué sur la figure ci-contre de l'interface graphique.

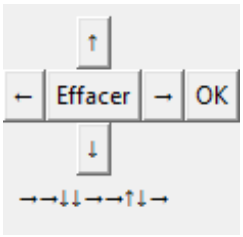


Figure III.4.c. : Aperçu des boutons de l'interface et de la trace du déplacement lorsque l'on clique sur les flèches.

Nous avons également une liste `cheminbis` qui nous permet ensuite de venir déchiffrer le parcours quand on appuie sur le bouton « OK ».

Cette fonction `lancerdeplacement` (figure ci-contre) est associée au bouton « OK ». Elle permet de démarrer le déplacement.
On vient tout simplement regarder les éléments un à un de `cheminbis` puis on recrée la grille après chaque déplacement demandé par l'utilisateur.
Nous avons veillé à mettre un temps de lattence pour que le robot ne se déplace pas trop vite et qu'ainsi l'utilisateur ait le temps de voir le robot se déplacer. De plus, les flèches décrivant le chemin dans l'interface graphique apparaissent les unes après les autres une fois qu'on a appuyé sur « OK », en même temps que le robot se déplace.

```
273  def lancerDeplacement(*arg):
274      global chemin,cheminbis
275      chemin = ''
276      sv.set(chemin)
277      robot=[ligneDepart,colonneDepart]
278      creergrille(robot[0],robot[1])
279      for i in range (len(cheminbis)) :
280          if cheminbis[i]==2:
281              robot[0]+=1
282              chemin += '↓'
283              sv.set(chemin)
284          elif cheminbis[i] == 6:
285              robot[1]+=1
286              chemin += '→'
287              sv.set(chemin)
288          elif cheminbis[i] == 4:
289              robot[1] -=1
290              chemin += '←'
291              sv.set(chemin)
292          elif cheminbis[i]==8:
293              robot[0] -= 1
294              chemin += '↑'
295              sv.set(chemin)
296          if not 0<= robot[0] <=nbColonnes or not 0<= robot[1] <=nbLignes:
297              print ('le robot est sortie du damier')
298              return
299          for i in range(len(listeobstacles)):
300              if listeobstacles[i][0]==robot[1] and listeobstacles[i][1] == robot[0]:
301                  print('le robot ne peut pas aller sur cette case')
302                  return
303          if robot[0]==ligneArrivee and robot[1]==colonneArrivee:
304              print('vous êtes arrivés à destination !')
305              return
306          sleep(0.3)
307          creergrille(robot[0],robot[1])
```

Figure III.4.e. : Code permettant de lancer le déplacement après sélection du parcours choisi.

L'utilisateur peut ainsi choisir le chemin qu'il veut que le robot exécute. Il peut modifier ce chemin et suivre en temps réel le déplacement du robot ainsi que le trajet fléché associé à ce déplacement.

5. Définir les obstacles

Cette capacité du programme est divisée en deux fonctions différentes : l'une permettant de positionner manuellement les obstacles sur les cases voulues et l'autre permet de placer aléatoirement n obstacles (n compris entre 1 et 10).

Ces deux fonctions apparaissent dans la fenêtre graphique comme présentée ci-contre. Nous avons tout en haut un menu déroulant où l'on rentre le nombre d'obstacles aléatoires souhaités. En dessous, lorsque nous cliquons sur le bouton « lancer placer obstacles », les obstacles sont placés sur la grille. Les trois boutons du dessous permettent d'ajouter manuellement des obstacles. Nous choisissons le numéro de la ligne puis celui de la colonne et enfin on appuie sur le bouton « entrer un obstacle » pour que celui-ci soit pris en compte.

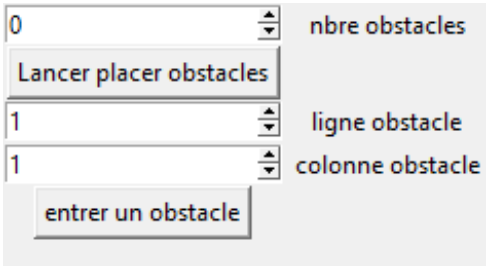


Figure III.5.a. : Aperçu de l'interface de commande des obstacles.

Les lignes de code ci-dessous permettent de définir les différents boutons décris précédemment.

```
227    #création des fenêtres de déroulement
228    s = Spinbox(autresboutons, from_=0, to=10)
229    s.grid(column=0, row=4)
230    stitre=Label(autresboutons, text="nbre obstacles")
231    stitre.grid(column=1, row=4)
232    Button(autresboutons,text = "Lancer placer obstacles",command =placerobst).grid(column=0,row=5) #bouton validation
233
234    sligne = Spinbox(autresboutons, from_=0, to=10)
235    sligne.grid(column=0, row=6)
236    slignetitre=Label(autresboutons, text='ligne obstacle')
237    slignetitre.grid(column=1, row=6)
238    scolonne = Spinbox(autresboutons, from_=0, to=10)
239    scolonne.grid(column=0, row=7)
240    scolonnetitre=Label(autresboutons, text='colonne obstacle')
241    scolonnetitre.grid(column=1, row=7)
242    Button(autresboutons,text = "entrer un obstacle",command = entrerObstacle).grid(column = 0, row = 8)
```

Figure III.5.b. : Code permettant de commander la création d'obstacles sur l'interface.

Les obstacles **aléatoires** sont régis par deux fonctions : `placerobst` et `obstaclesaleatoires`. La fonction `placerobst` récupère le nombre d'obstacles entrés par l'utilisateur dans le menu déroulant et applique la fonction `obstaclesaleatoires` autant de fois qu'il y a d'obstacles à placer. La fonction `obstaclesaleatoires` choisit aléatoirement une case dans `listecase`, qui est une liste de toutes les cases du damier et l'ajoute à `listeobstacles` (liste qui comprendra tous les obstacles et qui nous servira par la suite). Ensuite, cette fonction crée l'obstacle sur le damier. La façon de créer l'obstacle est délicate, puisqu'il faut regarder la « valeur » en pixels de chaque trait du contour de la case afin de placer la forme choisie (ici un rond noir pour un obstacle) au centre de la case. A la ligne 209, nous avons donc mis en commentaire ce que chaque paramètre signifiait afin de s'y retrouver.

Les obstacles **manuels** sont contrôlés à l'aide de la fonction `entrerObstacle`. Elle récupère les valeurs attribuées en ligne et en colonne, ajoute cet obstacle à la liste et créer l'obstacle sur le damier. Grâce au remplissage dans les deux fonctions de la liste `listeobstacles`, il est possible pour l'utilisateur de rentrer à la fois des obstacles aléatoires et manuels.

```
#fonctions : place le nombre d'obstacles aléatoires voulus
def placerobst(*arg):
    global creergrille
    nbreacquis = int(s.get())#récup nombre d'obstacles voulus dans liste déroulement
    for i in range (nbreacquis):#créer 1 obstacle aléatoire à chaque fois
        obstaclesaleatoires()
    creergrille = creergrille2


listeobstacles=[] #liste qui contiendra les positions des obstacles en pixels
listecases=[] #liste qui contient les cases du damier
n = taille/nbLignes
for i in range (nbLignes):#remplissage de la liste pour pouvoir choisir aléatoirement dedans
    for j in range (nbColonnes):
        listecases.append([i,j])

def obstaclesaleatoires():
    global listeobstacles, listecases,n
    casealeat=choice(listecases)#on choisit une case aléatoirement
    if (casealeat[0]*50==robot[1] and casealeat[1]*50==robot[0]) or (casealeat[1]*50==ligneArrivee*50 and casealeat[0]==colonneArrivee):#si obstacle se retrouve sur le robot ou s
        return
    listeobstacles.append(casealeat)
    platforme.create_oval(casealeat[0]*50,casealeat[1]*50,casealeat[0]*50+50,casealeat[1]*50+50,fill='black')#création de l'obstacle sur le damier (0=côté gauche, 1=haut, 2=côté

#fonction pour récupérer obstacle entré manuellement
def entrerObstacle(*arg):
    global creergrille, listeobstacles, ligneDepart,colonneDepart
    n=taille//nbLignes
    ichoisi= int(sligne.get())#récupère valeur de la ligne choisie
    jchoisi=int(scolonne.get())#récupère valeur de la colonne choisie
    listeobstacles.append([jchoisi-1,ichoisi-1])#le -1 vient du fait que pour l'utilisateur la première ligne est la ligne 1
    creergrille = creergrille2
    creergrille(ligneDepart,colonneDepart)
```

Figure III.5.c. : Code permettant d'ajouter les obstacles sur le damier, manuellement et automatiquement.

6. Déplacement automatique

Nous avons essayé de coder le déplacement automatique du robot si on lui indique la position de départ, celle d'arrivée et celles des obstacles sur le damier et que nous appuyons ensuite sur « Déplacement auto ».

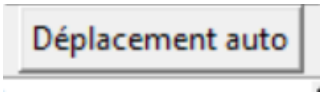
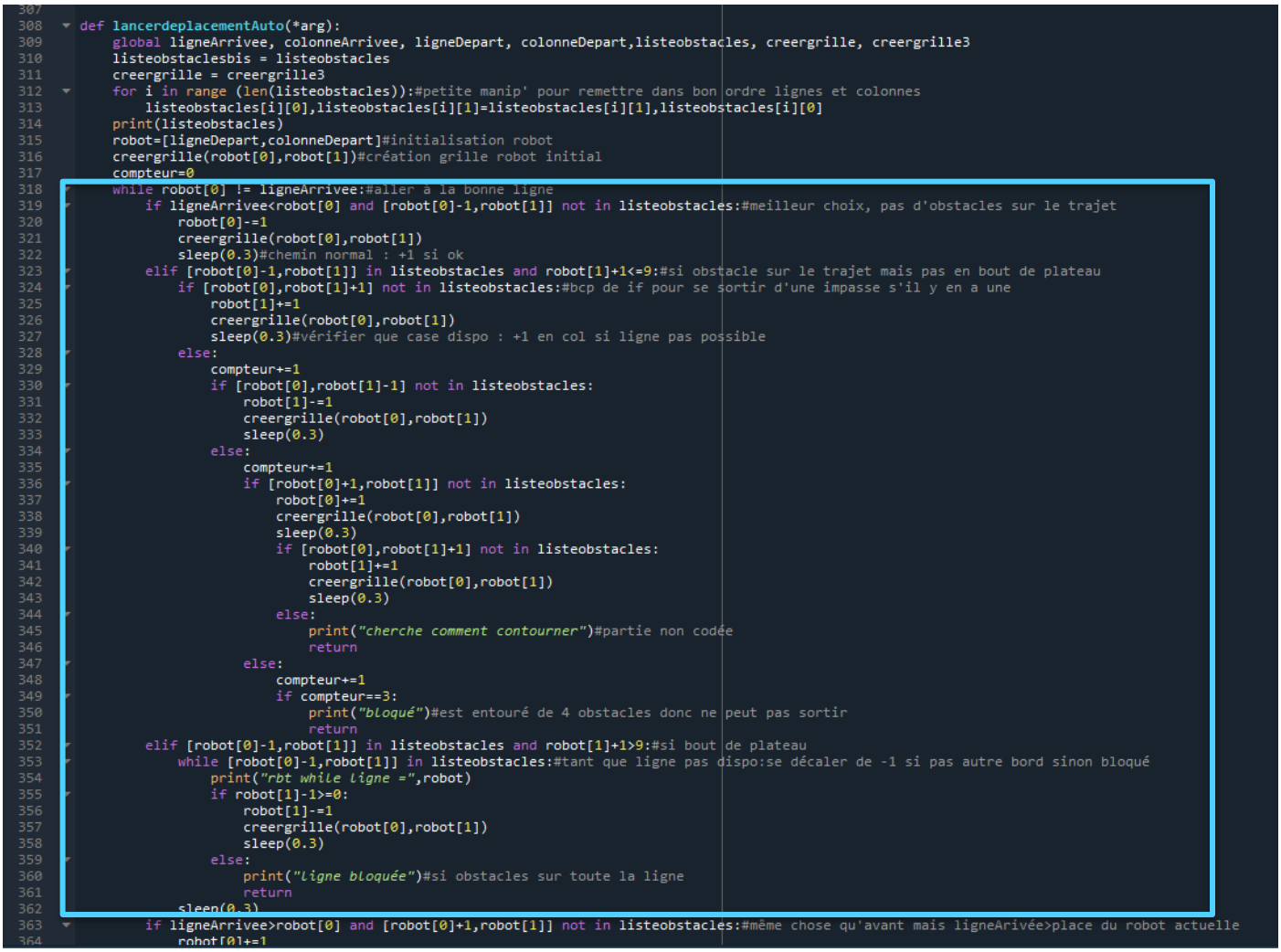


Figure III.6.a. : Aperçu du bouton de commande du déplacement automatique.

Nous avons procédé par étapes en commençant par coder un trajet sans obstacles. Nous avons choisi de le faire se déplacer d'abord en lignes puis en colonnes jusqu'à sa position d'arrivée (son trajet forme un « L »). Nous avons donc séparé la fonction en deux grandes parties : une boucle qui tourne jusqu'à ce que la ligne soit la bonne et une boucle qui tourne jusqu'à ce que la colonne soit la bonne. Nous avons redivisé ces boucles en deux conditions : suivant si la ligne d'arrivée était supérieure ou inférieure à la ligne du robot actuelle et de même sur les colonnes. Ci-dessous, une capture d'écran de la première partie du code (boucle des lignes, ligne d'arrivée inférieure à la ligne actuelle du robot) dans le carré bleu. Nous allons détailler cette partie, les autres fonctionnant de la même façon mais sur la ligne d'arrivée supérieure à la ligne actuelle ou sur les colonnes.



```
307
308 def lancerdeplacementAuto(*arg):
309     global ligneArrivee, colonneArrivee, ligneDepart, colonneDepart,listeobstacles, creergrille, creergrille3
310     listeobstaclesbis = listeobstacles
311     creergrille = creergrille3
312     for i in range (len(listeobstacles)):#petite manip' pour remettre dans bon ordre lignes et colonnes
313         listeobstacles[i][0],listeobstacles[i][1]=listeobstacles[i][1],listeobstacles[i][0]
314     print(listeobstacles)
315     robot=[ligneDepart,colonneDepart]#initialisation robot
316     creergrille(robot[0],robot[1])#création grille robot initial
317     compteur=0
318     while robot[0] != ligneArrivee:#aller à la bonne ligne
319         if ligneArrivee<robot[0] and [robot[0]-1,robot[1]] not in listeobstacles:#meilleur choix, pas d'obstacles sur le trajet
320             robot[0]-=1
321             creergrille(robot[0],robot[1])
322             sleep(0.3)#chemin normal : +1 si ok
323         elif [robot[0]-1,robot[1]] in listeobstacles and robot[1]+1<=9:#si obstacle sur le trajet mais pas en bout de plateau
324             if [robot[0],robot[1]+1] not in listeobstacles:#bcp de if pour se sortir d'une impasse s'il y en a une
325                 robot[1]+=1
326                 creergrille(robot[0],robot[1])
327                 sleep(0.3)#vérifier que case dispo : +1 en col si ligne pas possible
328             else:
329                 compteur+=1
330                 if [robot[0],robot[1]-1] not in listeobstacles:
331                     robot[1]-=1
332                     creergrille(robot[0],robot[1])
333                     sleep(0.3)
334                 else:
335                     compteur+=1
336                     if [robot[0]+1,robot[1]] not in listeobstacles:
337                         robot[0]+=1
338                         creergrille(robot[0],robot[1])
339                         sleep(0.3)
340                         if [robot[0],robot[1]+1] not in listeobstacles:
341                             robot[1]+=1
342                             creergrille(robot[0],robot[1])
343                             sleep(0.3)
344                         else:
345                             print("cherche comment contourner")#partie non codée
346                             return
347                     else:
348                         compteur+=1
349                         if compteur==3:
350                             print("bloqué")#est entouré de 4 obstacles donc ne peut pas sortir
351                             return
352         elif [robot[0]-1,robot[1]] in listeobstacles and robot[1]+1>9:#si bout de plateau
353             while [robot[0]-1,robot[1]] in listeobstacles:#tant que ligne pas dispo:se décaler de -1 si pas autre bord sinon bloqué
354                 print("rbt while ligne =",robot)
355                 if robot[1]-1>=0:
356                     robot[1]-=1
357                     creergrille(robot[0],robot[1])
358                     sleep(0.3)
359                 else:
360                     print("ligne bloquée")#si obstacles sur toute la ligne
361                     return
362             sleep(0.3)
363         if ligneArrivee>robot[0] and [robot[0]+1,robot[1]] not in listeobstacles:#même chose qu'avant mais ligneArivée>place du robot actuelle
364             robot[0]+=1
```

Figure III.6.b. : Code permettant de simuler le déplacement automatique du robot.

Le code ci-dessus est divisé en plusieurs parties. Le principe est d'évaluer le déplacement selon plusieurs critères. La première évaluation consiste à regarder si un obstacle est présent sur la case suivante, et si ce n'est pas le cas, le robot peut avancer normalement.

Si c'est le cas et qu'il n'est pas en bout de plateau (ligne 323 du code), il se décale d'une colonne. Nous avons voulu prendre en compte le cas où une impasse était présente, c'est pourquoi des lignes 324 à 350 du code, on évalue chaque case autour du robot pour en trouver une libre. Il s'avère que cette technique rajoute tellement de lignes de codes et les complique que nous avons décidé de ne pas les répéter pour les autres parties du code. C'est à cette limite que nous avons arrêté de développer le code, en ce sens qu'il ne prend pas en compte tous les types d'impasses que l'on peut trouver sur le plateau (3 cases occupées, un bord de plateau + deux cases occupées, etc.). Nous avons procédé avec une autre méthode pour coder le chemin le plus court et ainsi avoir un déplacement automatique du robot fonctionnel.

Si le robot a un obstacle devant lui et qu'il est en bord de plateau, il se décale d'une colonne vers la gauche (ligne352). Ici, nous avons ajouté une variante si l'utilisateur essayer de bloquer le robot avec une ligne entière bloquée : le robot va évaluer toute la ligne et retourner « ligne bloquée ». Cette option, si l'on bloque une ligne ou une colonne entière, fonctionne.

7. Sélectionner le chemin le plus court

Cette fonction lance le déplacement du robot de la case de départ à la case d'arrivée en évitant les obstacles. De plus ce programme a été optimisé pour qu'il s'agisse de l'un des chemins les plus courts.

Figure III.7.a. : Aperçu du bouton de sélection sur chemin le plus court.

Comme pour les fonctions précédentes, il s'agit d'un bouton placé dans l'interface graphique et relié à une fonction qui permet de calculer le chemin le plus court puis qui permet le déplacement du robot.

```
224    Button(autresboutons,text = "chemin le plus court",command = cheminlepluscourt).grid(column = 0, row = 1)
```

Figure III.7.b. : Code permettant d'ajouter le bouton de commande du chemin le plus court sur l'interface.

Avant de voir comment fonctionne la fonction `cheminlepluscourt`, nous allons voir sur quel principe elle repose.

Nous allons associer de proche en proche un numéro aux cases. Ainsi, nous commençons par placer notre robot à sa position de départ. Sur le schéma ci-contre, nous le notons R mais dans le programme on lui associera le numéro 1 (nous avons décalé les numéros attribués aux cases de 1 par rapport au schéma ci-contre). Ensuite nous positionnons les différents obstacles : ici des cases bleues mais dans le programme on leur donnera la valeur de 0.

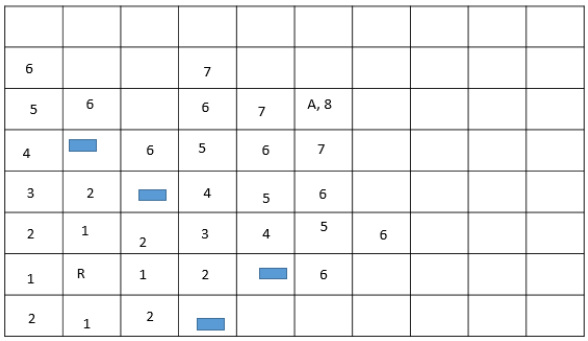


Figure III.7.c. : Schématisation du processus d'évitement des obstacles sur le damier.

Puis de proche en proche nous allons associer un numéro à chaque case. Par exemple, nous partons du point de départ et nous regardons toutes les cases où il peut aller en un coup et nous leur associons le numéro 1. Puis à partir de ces cases nous regardons où le robot peut aller sans revenir sur ses pas et ainsi de suite jusqu'à ce qu'on arrive à la position d'arrivée.

Nous remarquons ici qu'il existe de nombreux moyens d'arriver à la case d'arrivée en huit coups. Nous allons donc la suite remonter la grille de la position d'arrivée à la position de départ en ne prenant que l'une des solutions.
Nous avons ainsi obtenu le chemin le plus court en évitant les obstacles.

Concernant le code en lui-même, nous commençons par créer une liste de n par n qui contiendra les numéros associés à chaque case. Nous associons à chacune de ces case un nombre arbitraire ici 100. A chaque fois nous testerons que la case est bien toujours à 100, sinon cela signifiera qu'on lui a déjà assigné un numéro et qu'il ne faut donc pas le modifier.

Puis dans la boucle `while`, nous effectuons le programme tant qu'on n'a pas attribué de nombre à la case d'arrivée. Puis à l'aide d'un compteur, nous associons le nombre du compteur à toutes les cases en contact direct avec une case numéroté compteur -1. Nous avons rajouté une condition qui arrête la fonction su l'utilisateur a voulu bloquer le robot ou si l'arrivée n'est pas accessible, pour éviter que la fonction tourne en boucle.

```
def cheminlepluscourt(*arg):
    global listeobstacles, ligneDepart,ligneArrivee, colonneArrivee,colonneDepart
    listeobstaclesbis = listeobstacles
    robot=[ligneDepart,colonneDepart]
    creergrille(robot[0],robot[1])
    for i in range (len(listeobstacles)):
        listeobstacles[i][0],listeobstacles[i][1]=listeobstacles[i][1],listeobstacles[i][0]
    L=[]
    for i in range(nbLignes):
        vecteur = []
        for j in range (nbColonnes):
            vecteur.append(100)
        L.append(vecteur)
    for i in range(len(listeobstacles)):
        L[listeobstacles[i][0]][listeobstacles[i][1]]=0
    L[ligneDepart][colonneDepart] = 1
    compteur = 2
    while L[ligneArrivee][colonneArrivee]==100 :
        for i in range(nbLignes):
            for j in range(nbColonnes):
                if L[i][j] == compteur -1 :
                    if i+1 <10:
                        if L[i+1][j] == 100:
                            L[i+1][j] = compteur
                    if i-1 >= 0:
                        if L[i-1][j] == 100:
                            L[i-1][j] = compteur
                    if j+1 <10:
                        if L[i][j+1] == 100:
                            L[i][j+1] = compteur
                    if j-1 >= 0:
                        if L[i][j-1] == 100:
                            L[i][j-1] = compteur
        compteur = compteur +1
        if compteur >= 100:
            print("il n y a pas de chemin possible")
            return()
    listechemin = [[ligneArrivee,colonneArrivee]]
    compteur = L[ligneArrivee][colonneArrivee]
    while compteur > 1:
        retrouverchemin(compteur,L, listechemin)
        compteur = compteur -1
    for i in range (len(listeobstacles)):
        listeobstacles[i][0],listeobstacles[i][1]=listeobstacles[i][1],listeobstacles[i][0]
    for i in range(len(listechemin)):
        creergrille(listechemin[len(listechemin)-1-i][0],listechemin[len(listechemin)-1-i][1])
        sleep(0.3)
```

Figure III.7.d. : Code permettant de calculer le chemin le plus court entre deux points en évitant les obstacles sur le damier.

Ensuite dans une deuxième boucle `while`, nous reconstituons le chemin en faisant appel à la fonction `retrouverchemin` que nous détaillons ci-après. Puis finalement avec la dernière boucle `for`, nous faisons en sorte de déplacer le robot.

La fonction `retrouverchemin` ci-contre permet de remonter le chemin à partir de la case d'arrivée jusqu'à la case de départ. A chaque fois, il regarde où il se trouve et regarde dans les cases autour où il peut aller en diminuant le numéro de 1. Et dès qu'il a trouvé une solution, il sort du programme de sorte qu'on n'obtienne qu'un seul chemin à la fin.

```
def retrouverchemin(compteur, L, listechemin):
    n = len(listechemin)-1
    if listechemin[n][0]-1>=0 and L[listechemin[n][0]-1][listechemin[n][1]] == compteur -1:
        listechemin.append([listechemin[n][0]-1,listechemin[n][1]])
        return
    if listechemin[n][0]+1<10 and L[listechemin[n][0]+1][listechemin[n][1]] == compteur -1:
        listechemin.append([listechemin[n][0]+1,listechemin[n][1]])
        return
    if listechemin[n][1]-1>=0 and L[listechemin[n][0]][listechemin[n][1]-1] == compteur -1:
        listechemin.append([listechemin[n][0],listechemin[n][1]-1])
        return
    if listechemin[n][1]+1<10 and L[listechemin[n][0]][listechemin[n][1]+1] == compteur -1:
        listechemin.append([listechemin[n][0],listechemin[n][1]+1])
        return
```

Figure III.7.e. : Code permettant d'effectuer le chemin en sens inverse.

Ainsi dans la situation décrite à droite, le robot va suivre la flèche bleue. Il est donc très facile pour l'utilisateur d'utiliser cette fonction. On voit bien de plus sur cet exemple que le robot a pris le chemin le plus court.

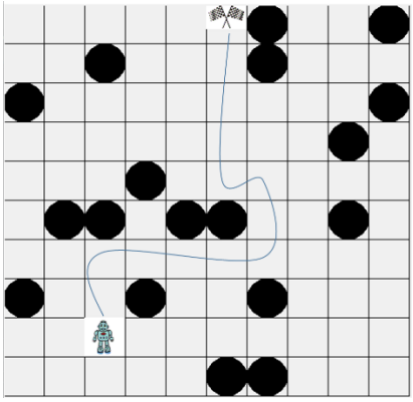


Figure III.7.f. : Aperçu de l'interface graphique lorsque le robot effectue le chemin le plus court entre deux points en évitant les obstacles sur le damier.

IV. Suivi des évolutions du projet

1. Répartition et planification des tâches

Nous avons réussi à tenir les engagements présentés lors de la première séance concernant la répartition des tâches. Le travail s'est essentiellement fait en binômes : Clara et Jérémy d'une part, Clotilde et Soizic d'autre part.

	Clotilde	Soizic	Clara	Jérémy
Mise en place du damier, création des fonctions de déplacement principales, chemin le plus court	**x**			
Gestion des obstacles et du chemin le plus court		**x**		
Design et interfaçage			**x**	**x**

Concernant le développement des programmes, nous avons été obligés de prendre des décisions au cours du projet et de parfois enlever certaines parties trop difficiles à mettre en œuvre.

Pour le déplacement automatique du robot, au vu des complications du code et de la variété du type d'impasses possibles pour le robot, nous avons décidé de laisser la fonction en suspens. Elle fonctionne donc sans obstacles, avec un obstacle seul sur son trajet et si une ligne ou une colonne est bloquée. Nous nous sommes concentrés pour développer la fonction du chemin le plus court afin de répondre au cahier des charges, ce qui était l'option la plus intéressante pour notre projet.

Concernant le design de l'interface, le binôme dédié a passé une séance à le développer. Le code est fonctionnel mais n'est pas basé sur la même structure que le reste du programme. En effet, le damier nouvellement créé est composé de 100 fenêtres individuelles, i.e. les Canvas, au lieu d'une fenêtre et 100 cases originellement. Nous avons donc décidé de ne pas rassembler les programmes au vu du temps restant, préférant présenter deux programmes qui fonctionnent séparément plutôt qu'un programme ne fonctionnant pas parfaitement.

2. Utilisation active de l'espace de travail partagé Basecamp

L'outil Basecamp nous a accompagnés tout au long des séances de projet. Son utilisation très intuitive a été pour nous un atout majeur dans la gestion des tâches et leur planification d'une séance à une autre.

Nous programmions à la fin de chaque séance les tâches à faire dans la rubrique To-Dos. Le fait de cocher au fur et à mesure les actions accomplies nous permettait d'avoir un aperçu global des éléments à terminer. De plus, nous pouvions partager des documents et les remplacer par une nouvelle version à chaque fois que quelqu'un modifiait quelque chose, ce qui nous évitait de faire plusieurs fois la même chose sans nous concerter.

Nous proposons ici un aperçu de l'emploi du temps et des objectifs que nous nous étions fixés au cours des cinq semaines de projet.

19/03	Réalisation du diaporama afin de répondre à l'appel d'offre. Répartition des tâches, élaboration d'un planning prévisionnel.
23/03	Prise en main de Python et découverte de TkInter. Formation à distance. Élaboration d'un premier damier.
30/03	Gestion et codage du déplacement pas à pas du robot sur le damier. Développement des premières options de base de déplacement. Création de la fonction de génération d'obstacles. Calcul de la fonction donnant le chemin le plus court.
02/04	Amélioration du design de l'interface : création d'un arrière-plan contenant des images pour chaque case. Prise en compte des obstacles visibles lors du déplacement.
27/04	Inclusion du programme du chemin le plus court dans le programme global. Rédaction des livrables.

Figure IV.2. : Aperçu de notre interface de travail sur BaseCamp.

V. Retour d'expérience des différents membres de l'équipe

Ce contexte particulier nous a montré à quel point le contact humain, auquel on ne prête pas forcément attention dans la vie de tous les jours à l'école, était essentiel pour développer un projet commun. Le fait de ne pas se voir, à cause du télétravail a constitué pour nous une expérience à laquelle il a été difficile de s'adapter. En effet, il s'est parfois avéré difficile de communiquer au sujet de certains points de blocage dans les codes car nous ne pouvions pas nous voir. Un partage d'écran ne remplace pas le contact avec les autres ! Cela a été particulièrement compliqué lorsque nous avons tenté de mettre en commun les différentes parties du code.

Toutefois, les outils disponibles en ligne, qu'il s'agisse d'eCampus ou de Basecamp nous ont permis de conserver un lien entre nous, en partageant notre travail et en faisant le point sur les activités en cours de réalisation.

Finalement, nous nous attendions, avant la crise, à une formation individuelle à la gestion de projet. Il se trouve que celle-ci n'a pas été pleinement mise en œuvre, compte tenu de la situation difficile. Cela nous a fait prendre conscience de la nécessité de se voir physiquement pour avancer. Ainsi, un travail à distance, bien qu'effectué en équipe, n'est pas véritablement ce que l'on pourrait appeler un projet.

Clara : Ce projet m'a permis de me re-familiariser avec Python et de découvrir l'outil d'interfaçage TkInter. J'ai beaucoup aimé travailler sur le design de l'interface puisque cela permet de laisser parler sa créativité. De plus, nous nous sommes tous très bien entendus au sein de l'équipe et à la fin de chaque séance, j'avais vraiment l'impression que nous avions considérablement avancé par rapport à la séance précédente. Même si nous étions parfois bloqués, nous avons toujours réussi à trouver une solution. C'était donc d'autant plus stimulant et cela donnait envie d'avancer encore davantage.

Clotilde : J'ai apprécié travailler sur ce projet pour de nombreuses raisons. Premièrement je trouve que nous avons formé une bonne équipe. Nous sommes tous organisés et perfectionnistes, de plus, nous nous entendons bien. C'est donc toujours avec plaisir que j'ai travaillé sur ce projet. D'un point de vue technique, ce projet m'a permis de me familiariser avec Python et avec la bibliothèque Tkinter. J'ai de plus à travers l'élaboration du chemin le plus court pu progresser dans ma façon de programmer. J'ai également appris à travers ce projet à créer une interface graphique, chose que je n'avais jamais faite avant.

Jérémy : Devoir mener à bien un projet qui nous est imposé est à mon sens une tâche extrêmement stimulante : contrairement à un projet personnel, cela nous apprend à sortir de notre zone de confort. J'ai dû réapprendre à utiliser Python alors que depuis deux ans je parle le Matlab. J'ai utilisé une librairie dont je ne connaissais pas l'existence avant la réalisation du projet (un saut dans l'inconnu…). Le codage est toujours source d'angoisse : on a toujours peur de ne pas trouver la source d'erreur. Mais quelle satisfaction quand tout fonctionne enfin ! Bien que le confinement ne nous ait pas permis de tester notre projet en réel, je suis très content du résultat. L'équipe que nous formions était très soudée et efficace, l'ai eu l'honneur de travailler avec des camarades fort agréables et extrêmement studieuses !

Soizic : J'ai trouvé ce projet challengeant et intéressant à développer, même à distance. Le fait de pouvoir se servir d'une interface graphique a été à la fois indispensable pour se rendre compte de ce que qu'on faisait et utile pour la suite de maîtriser quelques outils de Tkinter qui permettent de créer des boutons, des menus déroulants, etc. Programmer et réfléchir à plusieurs à une solution est aussi entraînant et nous sommes (sans vouloir nous jeter des fleurs) plutôt organisés donc c'était un projet vraiment sympa à suivre !

VI. Bilan des acquis et des compétences individuelles

1. Retour individuel sur la formation à distance

Nous faisons ici le bilan des compétences acquises par les différents membres de l'équipe. Ce projet nous a en effet permis de revoir l'utilisation de Python et de nous former à la création d'interfaces graphiques avec TkInter. Comme nous n'avons pas eu de cours à proprement parler, nous avons dû suivre, pour certains, des cours en ligne via des MOOCs et nous former nous-mêmes grâce aux ressources disponibles sur certains sites dédiés.

Outils de formation à distance	Jérémy	Clotilde	Soizic	Clara
Cours en ligne "Le Python pour tous"		x		
Quatre niveaux de cours pour Python - Georgia Tech	x			
Tutoriels YouTube		x	x	x
Liens Internet (voir ci-dessous)	x	x	x	x

2. Bibliographie commentée

Nous proposons ici une bibliographie commentée des divers liens auquel nous avons eu accès et qui nous ont aidés dans notre développement de projet. Cela pourra peut-être, en plus du forum, de permettre aux futurs 2As de trouver plus facilement les liens utiles pour leur projet...

Lien : https://www.youtube.com/watch?v=8J8wWxbAdFg
Ce tutoriel permet de poser les bases de la programmation d'un jeu vidéo Python. Il est très exhaustif et illustre la création d'un jeu dans lequel le joueur élimine des monstres morts-vivants en tous genres. N'allons pas jusque-là. Nous nous sommes intéressés en particulier à la manipulation de fenêtres d'affichages : ouverture, fermeture, format...

Lien : https://www.youtube.com/watch?v=bzSpU9s2Pto
Ici le youtubeur explique dans une première partie comment créer un jeu de dames (donc un quadrillage blanc et bleu) avec la librairie Tkinter. Dans une seconde partie il s'agit de placer des points (les dames !) sur le damier et de les faire bouger. Ce code nous a été très utile car très simple à comprendre et à mettre en œuvre.

Lien:https://www.youtube.com/watch?v=psaDHhZ0cPs&list=PLMS9Cy4Enq5JmIZtKE5OHJCI3jZfpASbR
Rappels de base sur l'implémentation Python : les syntaxes, les boucles, les fonctions etc … Peut-être très utile quand on s'est habitué à parler MatLab. Un peu de changement de temps en temps ne fait pas de mal.

Lien:https://python.doctor/page-tkinter-interface-graphique-python-tutoriel
https://python.developpez.com/faq/?page=Tk
Documentation très complète sur l'utilisation de Tkinter : manipulation de fenêtres, menus, boutons, canvas… Une mine d'informations ! Mais il faut chercher longtemps avant de trouver notre bonheur.

Lien : poitiers.fr/ENS/doku/doku.php/stu:python_gui:tuto_images

Article intéressant pour se familiariser avec la gestion des images dans Python : afficher une image, rectifier la taille...

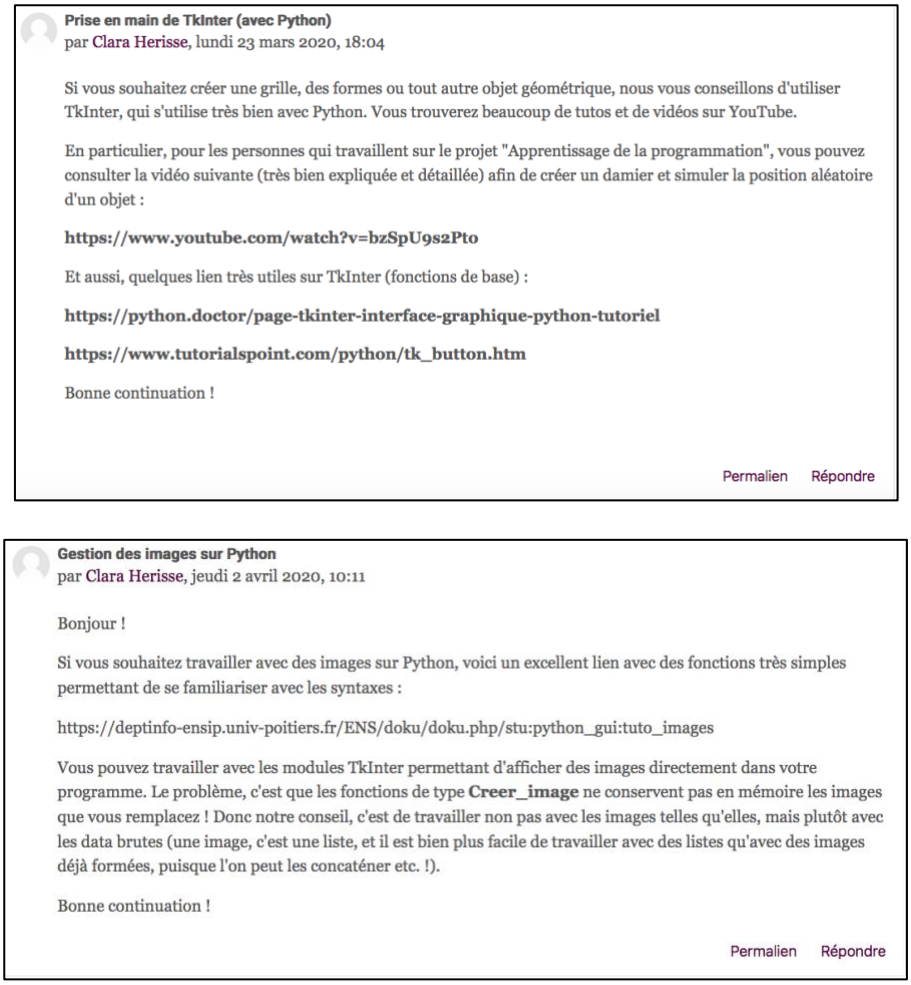
Lien: tutorialspoint.com/python/tk_button.htm
Tutoriel très simple sur l'utilisation de la fonction button, fonction qui nous aura particulièrement servi pendant l'implémentation des codes Python. Contrôler la taille du bouton, la position, insérer du texte...,

Lien : http://tkinter.fdex.eu/doc/caw.html
Ajouter des images dans un Canvas. Cet article nous a permis de gérer l'aspect esthétique du damier : taille du canvas, couleur, position des objets à l'intérieur...

3. Contributions aux forums d'eCampus

Tout au long du projet, il nous a été demandé de contribuer aux forums ouverts sur eCampus afin de partager nos expériences sur l'utilisation de TkInter, qu'elles soient positives ou négatives. Ces forums nous ont d'ailleurs permis de glaner quelques liens supplémentaires afin de progresser dans notre travail.
Nous avons trouvé cette initiative très intéressante car, en dépit de l'absence de contact humain, cela permet de maintenir le lien entre les étudiants. En temps normal, en séance, il est fréquent que nous nous aidions les uns les autres lorsqu'un groupe rencontre un problème par exemple. Ces forums avaient vocation à remplacer ce contact direct.

Nous concernant, nous avons communiqué les liens des vidéos permettant de créer un damier et avons partagé notre expérience sur la gestion des images sur Python, notamment à l'aide Canvas.



Prise en main de TkInter (avec Python)
par Clara Herisse, lundi 23 mars 2020, 18:04

Si vous souhaitez créer une grille, des formes ou tout autre objet géométrique, nous vous conseillons d'utiliser TkInter, qui s'utilise très bien avec Python. Vous trouverez beaucoup de tutos et de vidéos sur YouTube.

En particulier, pour les personnes qui travaillent sur le projet "Apprentissage de la programmation", vous pouvez consulter la vidéo suivante (très bien expliquée et détaillée) afin de créer un damier et simuler la position aléatoire d'un objet :

https://www.youtube.com/watch?v=bzSpU9s2Pto

Et aussi, quelques lien très utiles sur TkInter (fonctions de base) :

https://python.doctor/page-tkinter-interface-graphique-python-tutoriel

https://www.tutorialspoint.com/python/tk_button.htm

Bonne continuation !

Permalien Répondre

Gestion des images sur Python
par Clara Herisse, jeudi 2 avril 2020, 10:11

Bonjour !

Si vous souhaitez travailler avec des images sur Python, voici un excellent lien avec des fonctions très simples permettant de se familiariser avec les syntaxes :

https://deptinfo-ensip.univ-poitiers.fr/ENS/doku/doku.php/stu:python_gui:tuto_images

Vous pouvez travailler avec les modules TkInter permettant d'afficher des images directement dans votre programme. Le problème, c'est que les fonctions de type **Creer_image** ne conservent pas en mémoire les images que vous remplacez ! Donc notre conseil, c'est de travailler non pas avec les images telles qu'elles, mais plutôt avec les data brutes (une image, c'est une liste, et il est bien plus facile de travailler avec des listes qu'avec des images déjà formées, puisque l'on peut les concaténer etc. !).

Bonne continuation !

Permalien Répondre

Figure V. : Aperçu de notre contribution aux forums d'eCampus.

VII. Conclusion générale

Nous avons été très heureux de pouvoir développer un projet aussi ludique et amusant. Nous y avons pris beaucoup de plaisir, puisque c'est un jeu que nous pouvons utiliser nous-mêmes. Au-delà de la dimension ludique, la société *Solec Group®* aura formé nos esprits d'ingénieurs à développer au maximum nos propres initiatives techniques. Le parfait ingénieur sait répondre avec audace à des problématiques qui touchent au cœur l'apprentissage de demain. Il sait rencontrer d'un même front, triomphe après défaite, en conservant tout son courage et sa tête, sans jamais oublier que son plus grand plaisir… c'est d'apprendre ! Car comme l'a dit Edgar Poe, « ce n'est pas dans la connaissance qu'est le bonheur, mais dans l'acquisition de la connaissance. »

Et maintenant, à vous de jouer ! La vidéo explicative de notre projet est disponible à l'adresse suivante :
https://www.youtube.com/watch?v=YsRO7-Q2GKk
Le code est téléchargeable sur n'importe quel ordinateur possédant Python et sa bibliothèque associée TkInter. N'oubliez pas de télécharger, en plus du code, les images contenant dans le dossier .zip.