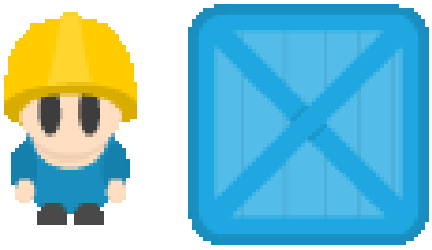
Académie de Montpellier – Lycée Nevers Montpellier

PROJET FINAL SPÉCIALITÉ
INFORMATIQUE ET SCIENCES DU NUMÉRIQUE

SOKOBAN



FLORIS Yann, Terminale S4

NADAL Lucas, Terminale S1

ANNÉE SCOLAIRE 2015/2016

SOMMAIRE

I. Introduction

A. Trouver un projet

Dans le cadre du projet de fin d'année d'I.S.N., bien choisir quel programme créer était important : il nous fallait quelque chose qui nous plaisait tous les 2, et d'un niveau de codage correct, mais qui représentait tout de même un défi. Programmer un jeu semblait donc une option logique. Devant utiliser TKInter, nous nous sommes orientés vers un jeu aux graphismes simples en 2D, comme un jeu de type puzzle. Nous nous sommes rapidement mis d'accord sur le Sokoban, jeu de puzzle/énigme japonais, dont les règles sont simples mais qui peut devenir un véritable casse-tête parfois. Le jeu tire son nom de son concept lui-même, un personnage doit ranger des caisses à une certaine place, et donc « sokoban » se traduit du japonais par « gardien d'entrepôt ».

B. Les règles

Wikipédia définit les règles du jeu tel que :

« Gardien d'entrepôt : le joueur doit ranger des caisses sur des cases cibles. Il peut se déplacer dans les quatre directions, et pousser (mais pas tirer) une seule caisse à la fois. Une fois toutes les caisses rangées […] le niveau est réussi et le joueur passe au niveau suivant, plus difficile en général. L'idéal est de réussir avec le moins de coups possibles (déplacements et poussées). »

Ces règles simples permettent donc plusieurs difficultés selon le niveau choisi. Coder ce jeu nous permet alors d'avoir différentes possibilités dans la création de niveaux. Cette idée nous a plus et nous en avons tiré un cahier des charges.

II. Cahier des charges

- Créer un niveau :
  De la forme d'un damier quadrillé, il peut contenir :
  - le joueur
  - une ou plusieurs caisse(s)
  - les objectifs (là où le joueur doit placer une caisse)
  - des murs délimitant ledit niveau.

- Limiter le joueur dans ses actions :
  - il se déplace dans 4 directions (case par case) : haut, droit, bas, gauche.
  - il peut pousser une caisse
  - les murs arrêtent le joueur et les caisses
  - puisqu'il ne peut pousser qu'une caisse à la fois, 2 caisses à côté l'arrêtent (une caisse a côté d'un mur aussi).

- Résolution d'un niveau :
  - un joueur réussit un niveau lorsque tous les objectifs ont une caisse dessus.
  - un joueur doit pouvoir redémarrer un niveau (il peut se bloquer et se retrouver dans une situation impossible).

- Interface :
  - une fenêtre d'accueil
  - une fenêtre permettant de choisir un niveau
  - un bouton « redémarrer »

III. Réalisation du projet

A. Première approche

Nous avons donc commencé par établir un algorithme simplifié en langage naturel pour en extraire les principales fonctions :

Accueil
Choisir un niveau, l'ouvrir
--Dans le niveau--
Les touches directionnelles dirigent le personnage
Quand un joueur est à côté d'un mur et se déplace vers lui, il ne déplace pas
Quand un joueur est à côté d'une caisse et se déplace vers elle, il déplace la caisse en même temps, sauf s'il y a à côté un mur ou une autre caisse
Si une caisse est sur un objectif, ce sera indiqué par une information visuelle (ex : la caisse change de texture)
Quand toutes les caisses d'un niveau sont sur les objectifs : on affiche « Réussi » et on passe au niveau suivant
Une touche « redémarrer » relance le niveau

Une fois les bases posées, nous avons donc commencé par la création d'une fenêtre TKInter et de nombreux tests : les Frames, Canvas, Labels par exemple afin d'obtenir le damier voulu. Nous nous sommes familiarisés avec les déplacements d'un objet, un carré simple, et l'association d'une touche à une action : augmenter ou réduire les coordonnées x et y en fonction de la touche pressée.

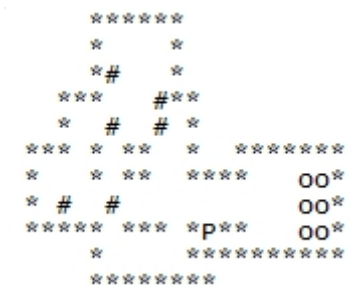
En dehors des heures de cours et tout au long du développement du programme, nous communiquions beaucoup nos idées et nos différentes lignes de codes par messages, ou appels vocaux comme Skype, mais aussi par le logiciel Github pour les échanges de fichiers, qui nous permettait de regrouper nos dossiers et de les comparer en ligne.

B. Problèmes et résolutions

Mais le premier et le principal problème a été la collision entre objets. Avec une forme telle que le carré, il faut détecter que 2 carrés (un mur et le joueur par exemple) se superposent s'il y a déplacement. L'idée était donc de déplacer le joueur, de relever ses coordonnées et de les comparer avec les coordonnées du mur : s'il y a superposition, on renvoie le joueur où il était avant le déplacement, sinon, il reste à ses coordonnées. La vitesse d'exécution de Python fait que si le joueur se déplace et revient car il est dans un mur, on ne le remarque pas à l'œil nu. On peut pour éviter cela malgré tout tester les coordonnées avant le déplacement, c'est-à-dire lorsque le joueur appuie sur « Flèche gauche », on teste les coordonnées actuelles + un déplacement à gauche. S'il n'y a pas d'obstacle, on applique le déplacement.
Cependant de nombreux bugs apparaissaient avec cette méthode, les collisions étaient détectées par exemple parfois trop tôt : le joueur s'arrêtait alors qu'il n'était pas à côté mais à une case d'écart du mur. Le même problème se posait bien sur pour la détection et le déplacement de caisses. Même la manière du calcul de centre des objets pour comparer les coordonnées posait problème. Nous avons donc changé de méthode et travaillé sur une plus simple pour les coordonnées. Le plateau du Sokoban étant un damier, nous avons numéroté les colonnes et lignes. Ainsi un objet est plus clairement localisé : il est par exemple en (2,3) = ligne 2, colonne 3 ; au lieu d'être en (150,150 ; 200,200) = 150,150 les coordonnées du pixel le plus en haut à gauche du carré et 200,200 les coordonnées les plus en bas en droite du carré). Ainsi plus de doute sur la position d'un objet.

A partir de ce problème résolu, il a donc été beaucoup plus facile de coder la suite. La caisse sur un objectif qui n'avait pas été traité avant répond aux mêmes pré-requis que le joueur et les murs : c'est de la vérification de coordonnées.

Mais arrivé à ce niveau là les lignes de codes devenaient nombreuses car les coordonnées de chaque mur étaient entrées une à une dans le code. Si nous voulions en créer plusieurs, cela allait prendre beaucoup de temps et de lignes. Nous nous sommes renseignés sur Internet sur les différentes méthodes utilisées en général dans les jeux comprenant différents niveaux. Nous avons trouvé une méthode lisant les fichiers textes (et en plus les lisant suivant la méthode des lignes et colonnes !) et transformant les caractères rencontrés en un objet défini. Par exemple, « * » dans le fichier texte devenait un mur dans TKInter. Même s'il utilisait des fonctions hors programme, nous nous sommes autorisés à les utiliser car elles étaient simples de compréhension et d'utilisation. Concrètement, la fonction *askopenfiles* demande d'ouvrir un fichier, comme son nom l'indique…



CAPTURE D'ECRAN D'UN NIVEAU EN FICHIER TEXTE

C. Fin du code, optimisation

Le cœur du code était écrit et fonctionnel, cependant il paraissait « brouillon ». Pour l'améliorer et parce que tous les deux nous étions intéressés par le codage plus avancé, nous avons regardé différents codes sources en Python de plusieurs jeux sur Internet. Nous avons remarqué très vite des fonctions récurrentes telles qu'*init* ou *self*. Nous avons donc approfondi pour comprendre ces fonctions et remarqué qu'elles seraient utiles à notre Sokoban. Les dernières vacances scolaires ont donc été en grande partie un remaniement du code original vers un code se rapprochant de codes « professionnels ». Nous sommes donc très fiers de la version finale de notre jeu même si plusieurs améliorations sont possibles.

IV. Améliorations envisageables

Par rapport à l'algorithme en langage naturel d'origine, la seule fonction absente au final est le passage au niveau suivant. Nous ne maîtrisons malgré tout pas assez bien les fonctions hors programme scolaire pour que le programme Python ouvre seul un fichier niveau après une énigme complétée.

Dans le jeu en lui-même, quasiment aucun bug n'a été observé une fois le programme achevé.
Le seul bug apparu peut-être deux ou trois fois est à la fin des niveaux. Une fois le dernier objectif complété avec une caisse, sous Windows, la fenêtre de victoire n'est pas apparue. Ce n'est pas réellement un bug entravant le déroulement du jeu, puisque dans tous les cas le passage à un autre niveau est manuel.

Différentes options supplémentaires sont possibles :

- Un compteur de coups, il faut donc réussir un niveau avec le moins de coups possibles. Cela entraîne possiblement une fenêtre « High scores » avec le moins de coups faits sur chaque niveau.
- Un bouton « annuler le dernier coup » peut être disponible si le joueur se bloque. Mais dans ce cas moins de réflexion sur quelle caisse pousser, car chaque erreur peut être annulée.
- Des ajouts graphiques, une interface plus agréable que du texte sur un fond uni.

V. Conclusion et ressentis

Lucas : Cette année d'I.S.N. aura été pour moi très intéressante : habitué de l'utilisation de l'ordinateur, je voulais en découvrir les coulisses. Le codage en lui-même est un exercice qui me plaît particulièrement et ces connaissances me seront sûrement utiles pour moi qui m'oriente vers l'infographie 3D pour mes études supérieures.

Yann : Passionné depuis toujours par la programmation, ce projet encadré et en équipe m'a permis de m'intéresser à un langage informatique supplémentaire et m'a permis également d'affirmer mon envie de poursuivre mes études dans ce milieu. C'est pourquoi j'ai prévu d'entrer à Epitech Montpellier lors de la prochaine rentrée afin d'approfondir ma passion et travailler avec cette dernière à terme.

Nous remercions ensemble Mr Magne pour ses cours et conseils tout au long de cette année.

VI. Annexes

Sources : Wikipedia, sites de langages et programmation comme OpenClassrooms, banque d'images et différents codes sources de jeux en Python.

A. Code Python

```python
import tkinter as tk
from tkinter.filedialog import askopenfilenames
import pydoc
import os


def enum(**enums):
    return type('Enum', (), enums)
Objectif = enum(plein=True, vide=False)


class Menu(object): #Création objets menu
    def __init__(self, app): #Initialisation du menu
        self.app = app

    def Ouvrirlvl(self): #Bouton ouvrir...
        app.grid_forget()
        niveau_files = askopenfilenames(initialdir = "niveaux")
        self.app.niveau_files = list(niveau_files)
        self.app.start_next_niveau()

    def About(self): #Bouton A Propos
        AboutDialog()


class Direction(object): #Directions pour touches clavier
    left = 'Left'
    right = 'Right'
    up = 'Up'
    down = 'Down'


class AboutDialog(tk.Frame): #Fenêtre bouton A Propos
    def __init__(self, master=None):
        tk.Frame.__init__(self, master)
        self = tk.Toplevel()
        self.title("A Propos")

        info = tk.Label(self, text=("Sokoban sous Python 3.5 par Lucas Nadal & Yann Floris"))
        info.grid(row=0)

        self.ok_button = tk.Button(self, text="OK", command=self.destroy)
        self.ok_button.grid(row=1)


class CompleteDialog(tk.Frame): #Fenêtre niveau gagné
    def __init__(self, master=None):
        tk.Frame.__init__(self, master)
        self = tk.Toplevel()
        self.title("Félicitations !")

        info = tk.Label(self, text=("Vous avez fini ce niveau !", height=15, width=150))
        info.grid(row=0)

        self.ok_button = tk.Button(self, text="OK", command=self.destroy)
        self.ok_button.grid(row=1)


```

```python
class Niveau(object): #Lecture fichiers niveau, association objets/caractères
    mur = '*'
    objectif = 'o'
    caisse_sur_objectif = '@'
    caisse = '#'
    joueur = 'P'
    sol = ' '


class Image(object): #Associations objets/images
    mur = os.path.join('images/mur.gif')
    objectif = os.path.join('images/objectif.gif')
    caisse_sur_objectif = os.path.join('images/caisse-sur-objectif.gif')
    caisse = os.path.join('images/caisse.gif')
    joueur = os.path.join('images/joueur.gif')
    joueur_sur_objectif = os.path.join('images/joueur-sur-objectif.gif')


class Application(tk.Frame):
    def __init__(self, master=None):
        tk.Frame.__init__(self, master)
        self.grid()
        self.master.title("Sokoban sous Python 3.5")
        self.master.resizable(0,0)
        icon = tk.PhotoImage(file=Image.caisse)
        self.master.tk.call('wm', 'iconphoto', self.master._w, icon)
        self.creer_menu()

        self.DEFAULT_SIZE = 200
        self.frame = tk.Frame(self, height=self.DEFAULT_SIZE, width=self.DEFAULT_SIZE)
        self.frame.grid()
        self.default_frame()

        self.joueur_position = ()
        self.joueur = None

        self.current_niveau = None
        self.niveau_files = []
        self.niveau = []
        self.caisses = {}
        self.objectifs = {}

    def key(self, event):
        directions = {Direction.left, Direction.right, Direction.up, Direction.down}
        if event.keysym in directions:
            self.move_joueur(event.keysym)

def creer_menu(self): #Création menu fenêtre accueil
    menu = tk.Menu(self.master)
    user_menu = Menu(self)
    self.master.config(menu=menu)

    file_menu = tk.Menu(menu)
    menu.add_cascade(label="Fichier", menu=file_menu)
    file_menu.add_command(label="Redémarrer", command=self.restart_niveau)
    file_menu.add_command(label="Ouvrir...", command=user_menu.Ouvrirlvl)
    file_menu.add_command(label="Quitter", command=menu.quit)

    help_menu = tk.Menu(menu)
    menu.add_cascade(label="Aide", menu=help_menu)
    help_menu.add_command(label="A Propos", command=user_menu.About)

def default_frame(self): #Création fenêtre d'Accueil
    start_width = 30
    start_label = tk.Label(self.frame, text="Bienvenue !\n", font='Helvetica', height=10, width=130)
    start_label.grid(row=0, column=0)

    start_label2 = tk.Label(self.frame, text="Pour jouer, choissisez un\nniveau dans Fichier -> Ouvrir...\n", font='Helvetica', height=10, width=100)
    start_label2.grid(row=1, column=0)

    start_label3 = tk.Label(self.frame, text="PROJET ISN 2016 - YANN FLORIS ET LUCAS NADAL\n",font='Helvetica', height=5, width=100)
    start_label3.grid(row=3, column=0)

def clear_niveau(self): #Fermer fenêtre de jeu
    self.frame.destroy()
    self.frame = tk.Frame(self)
    self.frame.grid()
    self.niveau = []

def start_next_niveau(self): #Passage niveau suivant (mauvais fonctionnement, arret du niveau)
    self.clear_niveau()
    if len(self.niveau_files) > 0:
        self.current_niveau = self.niveau_files.pop()
        niveau = open(self.current_niveau, "r")
        self.grid()
        self.load_niveau(niveau)
        self.master.title("Sokoban sous Python 3.5 par Lucas Nadal & Yann Floris")
    else:
        self.current_niveau = None
        self.master.title("Sokoban sous Python 3.5")
        self.default_frame()
        CompleteDialog()

def restart_niveau(self): #Fonction redémarrer du menu
    if self.current_niveau:
        self.niveau_files.append(self.current_niveau)
        self.start_next_niveau()
```

```
156
157     def load_niveau(self, niveau): #Génération du niveau basé sur le texte
158         self.clear_niveau()
159
160         for row, line in enumerate(niveau):
161             niveau_row = list(line)
162             for column,x in enumerate(niveau_row):
163                 if x == Niveau.joueur:
164                     niveau_row[column] = Niveau.sol
165
166                 elif x == Niveau.objectif:
167                     self.objectifs[(row, column)] = Objectif.vide
168
169                 elif x == Niveau.caisse_sur_objectif:
170                     self.objectifs[(row, column)] = Objectif.plein
171
172             self.niveau.append(niveau_row)
173
174             for column, char in enumerate(line):
175                 if char == Niveau.mur:
176                     mur = tk.PhotoImage(file=Image.mur)
177                     w = tk.Label(self.frame, image=mur)
178                     w.mur = mur
179                     w.grid(row=row, column=column)
180
181                 elif char == Niveau.objectif:
182                     objectif = tk.PhotoImage(file=Image.objectif)
183                     w = tk.Label(self.frame, image=objectif)
184                     w.objectif = objectif
185                     w.grid(row=row, column=column)
186
187                 elif char == Niveau.caisse_sur_objectif:
188                     caisse_sur_objectif = tk.PhotoImage(file=Image.caisse_sur_objectif)
189                     w = tk.Label(self.frame, image=caisse_sur_objectif)
190                     w.caisse_sur_objectif = caisse_sur_objectif
191                     w.grid(row=row, column=column)
192                     self.caisses[(row, column)] = w
193
194                 elif char == Niveau.caisse:
195                     caisse = tk.PhotoImage(file=Image.caisse)
196                     w = tk.Label(self.frame, image=caisse)
197                     w.caisse = caisse
198                     w.grid(row=row, column=column)
199                     self.caisses[(row, column)] = w
200
201                 elif char == Niveau.joueur:
202                     joueur_image = tk.PhotoImage(file=Image.joueur)
203                     self.joueur = tk.Label(self.frame, image=joueur_image)
204                     self.joueur.joueur_image = joueur_image
205                     self.joueur.grid(row=row, column=column)
206                     self.joueur_position = (row, column)
```

```
207
208     def move_joueur(self, direction): #Délpacement joueur
209         row, column = self.joueur_position
210         prev_row, prev_column = row, column
211
212         blocked = True
213         if direction == Direction.left and self.niveau[row][column - 1] is not Niveau.mur and column > 0: #Si joueur va à gauche et block suivant n'est pas mur
214             blocked = self.move_caisse((row, column - 1), (row, column - 2)) #Vérifier si caisse poussée est bloquée
215             if not blocked: #Si non bloquée déplacer à gauche
216                 self.joueur_position = (row, column - 1)
217
218         elif direction == Direction.right and self.niveau[row][column + 1] is not Niveau.mur: #Si joueur va à droite et block suivant n'est pas mur
219             blocked = self.move_caisse((row, column + 1), (row, column + 2)) #Vérifier si caisse poussée est bloquée
220             if not blocked: #Si non bloquée déplacer à droite
221                 self.joueur_position = (row, column + 1)
222
223         elif direction == Direction.down and self.niveau[row + 1][column] is not Niveau.mur:
224             blocked = self.move_caisse((row + 1, column), (row + 2, column))
225             if not blocked:
226                 self.joueur_position = (row + 1, column)
227
228         elif direction == Direction.up and self.niveau[row - 1][column] is not Niveau.mur and row > 0:
229             blocked = self.move_caisse((row - 1, column), (row - 2, column))
230             if not blocked:
231                 self.joueur_position = (row - 1, column)
232
233         all_objectifs_plein = True #Tous objectifs remplis
234         for objectif in self.objectifs.values(): #Pour chaque objectif
235             if objectif is not Objectif.plein: #Si au moins 1 objectif pas plein
236                 all_objectifs_plein = False #Alors tous objectifs pas pleins
237
238         if all_objectifs_plein: #Si tous objectifs remplis
239             self.start_next_niveau() #Mettre fin au niveau
240             return
241
242         row, column = self.joueur_position
243
244         if not blocked:
245             self.joueur.grid_forget() #Effacer de la mémoire la position précédente du joueur
246
247             if self.niveau[row][column] is Niveau.objectif: #Si joueur est sur case objectif
248                 joueur_image = tk.PhotoImage(file=Image.joueur_sur_objectif) #Afficher image joueur sur objectif
249             else:
250                 joueur_image = tk.PhotoImage(file=Image.joueur) #Sinon afficher image joueur
251
252             self.joueur = tk.Label(self.frame, image=joueur_image)
253             self.joueur.joueur_image = joueur_image
254             self.joueur.grid(row=row, column=column)
255
256     def move_caisse(self, location, next_location): #Pousser les caisses
257         row, column = location
258         next_row, next_column = next_location
259
260         if self.niveau[row][column] is Niveau.caisse and self.niveau[next_row][next_column] is Niveau.sol: #Si block suivant caisse est vide
261             self.caisses[(row, column)].grid_forget()
262             caisse = tk.PhotoImage(file=Image.caisse)
263             w = tk.Label(self.frame, image=caisse)
264             w.caisse = caisse
265             w.grid(row=next_row, column=next_column)
266
267             self.caisses[(next_row, next_column)] = w
268             self.niveau[row][column] = Niveau.sol
269             self.niveau[next_row][next_column] = Niveau.caisse
270
271         elif self.niveau[row][column] is Niveau.caisse and self.niveau[next_row][next_column] is Niveau.objectif: #Si block suivant caisse est objectif
272             self.caisses[(row, column)].grid_forget()
273             caisse_sur_objectif = tk.PhotoImage(file=Image.caisse_sur_objectif)
274             w = tk.Label(self.frame, image=caisse_sur_objectif)
275             w.caisse = caisse_sur_objectif
276             w.grid(row=next_row, column=next_column)
277
278             self.caisses[(next_row, next_column)] = w
279             self.niveau[row][column] = Niveau.sol
280             self.niveau[next_row][next_column] = Niveau.caisse_sur_objectif
281             self.objectifs[(next_row, next_column)] = Objectif.plein
282
283         elif self.niveau[row][column] is Niveau.caisse_sur_objectif and self.niveau[next_row][next_column] is Niveau.sol: #Si block suivant caisse sur objectif est vide
284             self.caisses[(row, column)].grid_forget()
285             caisse = tk.PhotoImage(file=Image.caisse)
286             w = tk.Label(self.frame, image=caisse)
287             w.caisse = caisse
288             w.grid(row=next_row, column=next_column)
289
290             self.caisses[(next_row, next_column)] = w
291             self.niveau[row][column] = Niveau.objectif
292             self.niveau[next_row][next_column] = Niveau.caisse
293             self.objectifs[(row, column)] = Objectif.vide
294
295         elif self.niveau[row][column] is Niveau.caisse_sur_objectif and self.niveau[next_row][next_column] is Niveau.objectif: #Si block suivant caisse sur objectif est objectif
296             self.caisses[(row, column)].grid_forget()
297             caisse_sur_objectif = tk.PhotoImage(file=Image.caisse_sur_objectif)
298             w = tk.Label(self.frame, image=caisse_sur_objectif)
299             w.caisse_sur_objectif = caisse_sur_objectif
300             w.grid(row=next_row, column=next_column)
301
302             self.caisses[(next_row, next_column)] = w
303             self.niveau[row][column] = Niveau.objectif
304             self.niveau[next_row][next_column] = Niveau.caisse_sur_objectif
305             self.objectifs[(row, column)] = Objectif.vide
306             self.objectifs[(next_row, next_column)] = Objectif.plein
307
308         if self.bloque(location, next_location):
309             return True
310         return False
311
312     def bloque(self, location, next_location): #Bloquage caisse
313         row, column = location
314         next_row, next_column = next_location
315
316         if self.niveau[row][column] is Niveau.caisse and self.niveau[next_row][next_column] is Niveau.mur: #Si block suivant caisse est mur
317             return True
318         elif self.niveau[row][column] is Niveau.caisse_sur_objectif and self.niveau[next_row][next_column] is Niveau.mur: #Si block suivant caisse sur objectif est mur
319             return True
320         elif (self.niveau[row][column] is Niveau.caisse_sur_objectif and #Si block suivant caisse sur objectof est caisse ou caisse sur objectif
321                 (self.niveau[next_row][next_column] is Niveau.caisse or
322                         self.niveau[next_row][next_column] is Niveau.caisse_sur_objectif)):
323             return True
324         elif (self.niveau[row][column] is Niveau.caisse and#Si block suivant caisse est caisse ou caisse sur objectif
325                 (self.niveau[next_row][next_column] is Niveau.caisse or
326                         self.niveau[next_row][next_column] is Niveau.caisse_sur_objectif)):
327             return True
328
329
330 app = Application()
331 app.bind_all("<Key>", app.key)
332 app.mainloop()
333
```