Jules Cléro, Kévin Sahin
L1 Informatique
Année : 2010/2011

Rapport de Projet Python

Thématique : Développement d'une application du jeu du Tangram

Code UE : L1INF26F0S

Sommaire

I/ Introduction

Le présent rapport a pour but d'expliquer l'analyse que nous avons faite du problème posé, les solutions que nous avons dans un premier temps expérimenté, puis dans un second temps les solutions que nous avons sélectionné. Dans cette optique, nous expliquerons l'algorithme, ainsi que les fonctions principales, et nous justifirons nos choix d'implémentation.

Énoncé :

« Réaliser un programme permettant de jouer au Tangram 7 pièces (5 triangles, 1 parallélogramme, 1 carré). Gérer les déplacements, les interactions, les rotations à la souris. »

Mais...qu'est ce que le jeu du Tangram ?

« *L'origine du mot « tangram » semble être occidentale : il serait composé de « tang », signifiant « chinois » en cantonais, et de « gram », rappelant le caractère dessiné des figures.* »

Le principe du jeu est simple : 7 formes géométriques basiques vous sont présentées, vous devez les utiliser afin de réaliser une figure plus complexe, tel qu'un canard, ou un lapin...Pour cela, vous aurez le droit de déplacer les figures, de les tourner, et enfin, pour le parallélogramme, de faire son symétrique.

Nous allons donc devoir implémenter des solutions techniques, des fonctions, et un algorithme nous permettant de satisfaire à ces exigences techniques.

C'est parti...

II/ Analyse du problème.

Étant donné le fonctionnement du jeu , le problème que nous allons avoir sera la gestion en général des figures : la création des figures, le déplacement, la rotation et la détection de fin de jeu, tout en respectant les contraintes techniques d'implémentation (à savoir l'unique utilisation de Turtle et de ses dérivés).

II.A. Déplacement :

Le déplacement d'une seule figure n'est pas problématique, il suffit d'utiliser les fonctions intégrées au module Turtle, tel que onclick, ondrag et goto.

Là où il y a un problème, c'est la gestion de plusieurs figures dans une même fenêtre. En effet, doit-on utiliser une seule tortue qui change de forme, ou bien faire en sorte d'en avoir plusieurs sur un même écran ?

De plus, il nous faudra en permanence savoir la position de chaque figure sur l'écran.

II.B. Rotation

Ici encore, la rotation est faisable de bien des façons : On peut tout d'abord faire tourner la tortue avec un simple rt(), voire un seth(), on peut aussi avoir plusieurs images de nos figures décalées de x degrés, voir même faire plein de begin_poly() end_poly()...

II.C. Création des figures :

Il y a plusieurs façons d'afficher une figure géométrique avec le module turtle : On peut utiliser une image au format gif, dont la tortue prendra la forme, on peut également dessiner avec la tortue l'image (fonction fd, rt...), ou bien utiliser les fonctions begin_poly() et end_poly() pour que la tortue prenne la forme géométrique contenue dans le polygone créé.

Nous avons donc eu l'embarras du choix, cependant, c'est en fait le problème majeur du projet, puisque cela implique une cohérence entre les choix que nous faisons pour les déplacements et la rotation.

III/ Les solutions testées, la solution retenue :

III.A. Les images :

Tout d'abord nous avons tenté de créer des figures géométriques à l'aide d'images au format gif. Nous avons cependant été vite limité dans l'utilisation de ces images, car il est impossible de leur imposer une rotation. Malgré cela, les images se déplaçaient correctement, et la détection de fin de jeu était aisée. Nous aurions pu choisir cette option, car il était possible de créer mettons 24 images par figures décalées de 15 degrés. ILnous aurait donc fallu 168 images en tout, ce qui nous paraissait exagéré.

III.B. Les stamps[1]

Pour pallier le problème avec les images, nous avons utilisé une tortue qui prenait successivement les différentes formes, puis on les collait sur l'écran quand l'utilisateur avait fini le déplacement (à l'aide de stamp). Le nom du curseur est p ; nous décrirons les procédés pour obtenir un curseur personnalisé dans une fenêtre Tkinter à l'aide d'un Canvas, dans le chapitre implémentation.

Ce premier essai consistait à faire une fonction maitresse sélection, qui récupérait en paramètre la position x,y d'un clic. Si cette position correspondait à la position d'une des images contenues dans la liste tabl, le curseur allait à cette position et prenait la forme de l'image qui y était précédemment. Le déplacement de la souris était lié à une fonction move qui déplaçait ce curseur. Cependant, à la fin du placement, on marquait la nouvelle position de la figure à l'aide de la fonction stamp. Néanmoins on remarque bien que les positions successives restent affichées à l'écran, puisque le curseur doit prendre plusieurs formes, le stamp à l'écran est obligatoire pour chaque figure. Il nous a donc fallu implémenter une fonction de rafraichissement (refresh), qui effaçait l'écran puis le mettait à jour en redessinant chaque figure. De plus, une fonction where permet de savoir en permanence où se trouve chaque figure.

Le problème majeur de cette solution est le clignotement intempestif de l'écran de jeu du à la ré-actualisation de tous les éléments à l'écran.

1 Voir le code en annexe.

III.C. La solution retenue

Suite à ce problème de rafraichissement, nous avons fait des recherches plus approfondies sur le module Turtle. Nous avons donc découvert qu'il était possible de gérer plusieurs curseurs dans une même fenêtre. Nous avons donc procédé ainsi, on crée sept curseurs différents qui représentent chacun une unique figure. Les figures sont faites grâce à la méthode begin_poly() end_poly() décrite dans le chapitre précédent.

La solution pour les rotations est seth(), pour le déplacement ondrag, et la sélection avec onclick, cela est permis grâce au fait qu'on ait plusieurs curseurs.

III.D. L'algorithme

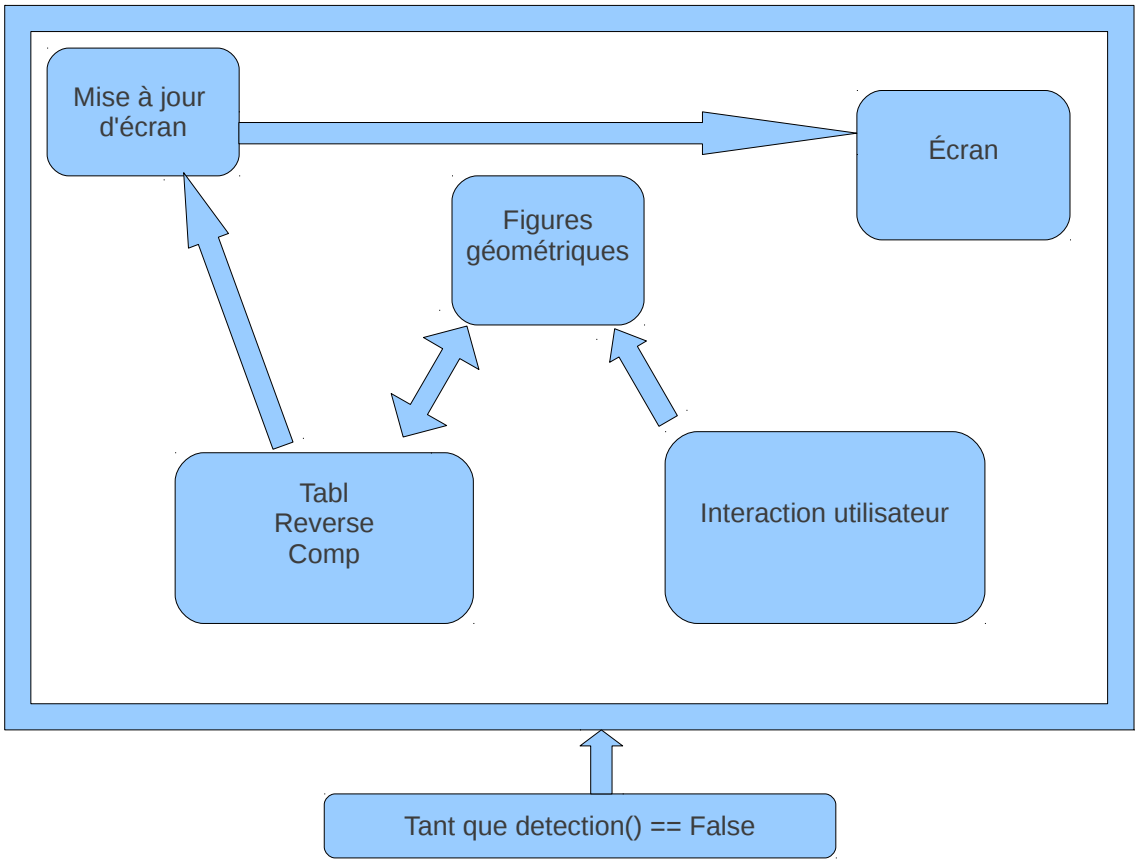
On dessine les figures géométriques, et on leur attribue une tortue chacune (sauf pour le symétrique du parallélogramme).

Après avoir initialisé l'écran, on stocke la position de chaque tortue ainsi que son inclinaison en permanence. Pour le parallélogramme il faudra aussi penser à récupérer une valeur booléenne pour savoir quel parallélogramme est affiché.

On attend une interaction de la part de l'utilisateur (un clic droit pour faire tourner la figure, un clic gauche prolongé pour la déplacer, un double clic pour le symétrique du parallélogramme).

On fait cela tant que la figure n'est pas complétée, tout en vérifiant si elle l'est à chaque lâché de clic.

Ci-dessous le diagramme de « l'algorithme » de notre programme :



IV/ L'implémentation

Nous avons donc à présent l'algorithme et les solutions pour l'implémentation de notre programme. Nous allons donc pouvoir entrer dans le vif du sujet : l'implémentation.

Notre programme est composé de trois blocs principaux :

- ✗ le bloc menu principal
- ✗ le bloc tangram
- ✗ le bloc fonction

Le premier bloc contient donc la définition de la fenêtre principale. C'est ici que l'on va définir les premières variables comme color_default ou helped dont on va très vite avoir besoin, mais c'est ici que l'on va préparer notre jeu. En effet, il nous faut lire dans le fichier sav_res.txt la résolution courante du jeu et préparer les variables qui dépendent de la

résolution (comme par exemple l'emplacement des objets dans la fenêtre ainsi que les tableaux de niveaux). Ces tableaux de niveaux, contiennent toutes les positions de chaque figure terminées ainsi que les positions de fin (celles ou les figures sont terminées), ils contiennent ainsi la valeur finale de comp et reverse ainsi que l'emplacement du fond d'écran dédié à chaque niveau. Le tableau carré fait cependant exception puisque qu'il ne sert en fait qu'au positionnement des figures de jeu et lors de l'exécution de la fonction restart(). Ensuite, puisque tout est prêt on peut créer la fenêtre principale qui permet au joueur de lancer le jeu, de voir le fichier crédit dans gedit ou bien de quitter le tangram. Pour l'ergonomie, nous avons fait une petite animation qui affiche pendant 2,5 secondes le nom du jeu (cela est possible grâce à la fonction wait qui lance une liste d'instructions qui place tout les objets de cette première fenêtre. Cela est possible en déclarant une fonction lambda.

Arrêtons nous un peu sur cette dernière que nous avons beaucoup utilisée dans ce jeu de Tangram. Bien souvent, on a besoin de lancer une fonction avec plusieurs arguments qui peuvent varier mais cela n'est pas toujours possible. Par exemple, lorsque l'on clique sur un bouton on ne peut uniquement lancer une procédure (qui n'a pas d'argument), une solution pour passer des arguments à celle-ci si besoin serait donc de déclarer une fonction lancement() qui exécute une uniquement instruction : le lancement de notre fonction visée avec les arguments correct. Cette méthode peut très vite devenir lourde à écrire, c'est pourquoi il existe des fonctions *lambda* qui permettent la déclaration de fonctions à la volée. Ainsi la commande lancée par notre bouton pourra être : command = lamda : level(canard) tandis que command = level(canard) aurait renvoyé une erreur.

Le jeu est prêt cliquons donc sur jouer ! Une fenêtre va s'ouvrir et le niveau par défaut est celui du canard. Mais avant cela, les figures ont été créées. Le curseur de base que nous avons appelé p a dessiné chacune des figures et elles ont ensuite été enregistrées en temps que curseur à l'aide de la méthode register_shape(''nom_du_curseur''). Une fois cela fait, chaque shape est assigné à son curseur, les curseurs ont le préfixe "C_" à leur nom. On lit ensuite les couleurs sauvegardées par l'utilisateur et on les met dans les variables "C_color" qui sont de type StringVar pour pouvoir être pilotées à l'aide du menu. Le bloc se termine par des binding de touche pour que le jeu puisse démarrer.

A la fin du bloc jeu, la fonction restart() a été appelée, cette fonction entraine la mise à jour de tabl aux valeurs du carré de base et appel init() et color_init() pour mettre à jour les positions et les couleurs.

Décrivons donc à présent le bloc fonction ; nous allons faire une petite description de chaque procédure (dans l'ordre d'apparition dans le code) de notre programme et en expliquer la place au sein de celui-ci. Nous mettrons de côté restart() init() et color_init() dont nous venons de parler.

La première est la fonction distance() qui se contente de retourner la valeur de la distance entre deux points passée en paramètre, on s'en sert uniquement dans la fonction detection(). Cette dernière est l'une des plus importantes dans ce programme puisqu'elle permet d'indiquer au joueur si ce dernier a gagné ou non. Pour se faire, on initialise une variable booléenne à True et l'on regarde chaque figure géométrique pour vérifier que tout est bien en place. Si quelque chose ne va pas, test passe à False. Voyons maintenant ce que détection vérifie : tout d'abord les correspondances entre tabl et tabl_end mais il faut bien faire attention à certains cas. En effet pour le carré, le parallélogramme ou le triangle moyen, il n'y a pas de problème, on fait la distance entre l'emplacement actuel et l'emplacement final de l'un de ces éléments et si leur distance est inférieure à 5, test reste à True et l'élément est replacé à la position qui lui correspond dans tabl_end pour qu'il n'y ait pas de décalage dans la figure finale. Par contre pour ce qui est des deux grands triangles et des deux petits ce n'est pas la même histoire. Prenons les deux grands triangles (c'est exactement la même chose pour les deux petits), leur indice de position dans tab et dans tab_end sont respectivement 0 et 1 mais il faut vérifier si ni l'un ni l'autre n'est placé à l'une des deux positions disponibles pour cette forme. Nous allons donc procéder ainsi : pour chaque triangle, si sa distance avec tabl_end[0] ou tabl_end[1] est inférieure ou égale à 5, alors on compare ces deux distances pour savoir à quelle position doit être réajusté le triangle tout en vérifiant que les deux distances ne soient pas très proches car dans ce cas les deux triangles vont à cette position. Pour mieux comprendre, observez le niveau du canard et vous verrez que les deux grands triangles ont leur base qui se touche, sans ce dernier test le réajustement ne se fait pas. Une fois que nous avons vérifié les positions on passe aux rotations. Ici même calvaire que pour les grands et petits triangles mais ici on ne fait que vérifier que pour un indice donné la valeur dans comp et comp_end est bien la même. Dans le cas contraire test passe à False. Il ne nous reste

plus qu'à vérifier la valeur de reverse et si après tout cela ,test est encore True on affiche une fenêtre de félicitations au joueur qui lui permet, soit de quitter le jeu, soit de recommencer le niveau (pour ensuite changer en sélectionnant celui qu'il veut dans le menu).

Nous arrivons maintenant à sav_color() et sav_res() ces deux fonctions marchent de la même façon, l'une concerne la sauvegarde des couleurs des formes, l'autre permet le changement de résolution. Elles se comportent ainsi : grâce à l'importation du module pickle on ouvre un fichier texte (sav_res.txt ou sa_color.txt) à l'aide de la fonction with, ensuite on crée un objet pickler et grâce à sa méthode dump on pourra écrire le contenu d'une variable dans le fichier souhaité. Il faut aussi savoir que la lecture se fait de la même façon mais on utilise la méthode load du pickler. Cependant, la fonction sav_res() a aussi une autre utilité ; en effet, si le joueur veut changer de résolution le jeu devra être redémarré. La fonction sav_res() après son exécution affiche donc un message demandant de redémarrer l'application, celui-ci contient un bouton permettant de fermer l'application mais le joueur devra relancer celle-ci de lui-même.

Passons à présent aux fonctions rotations, "aux" au pluriel puisque il y en a une pour chaque figure géométrique. Vous devez donc vous demander pourquoi nous n'en avons pas fait qu'une seule, et bien car après beaucoup d'essais nous n'y sommes pas arrivé. En effet, la fonction se déclenche lorsque l'on fait un clic droit sur l'objet que l'on veut faire tourner, cependant, il n'y a aucun moyen (à notre connaissance) de récupérer le nom de l'objet sur lequel on clique. Il faut donc que l'on passe en argument à la fonction rotation ce nom d'objet curseur pour pouvoir lui imposer une rotation. Vous devez donc penser que nous nous torturons pour rien puisque l'on a déjà fait cela avec une fonction lambda ! Le problème est qu'une fonction lambda ne reçoit pas d'argument, or la méthode onclick() des curseurs qui permet de détecter un clic sur ces derniers, passe à la commande qu'il lance (ici rotation) les coordonnées x,y du clique, lambda est donc inutilisable. Nous avons donc décidé que pour chaque curseur, il y aurait un binding d'événement qui lancerait la fonction rotation qui lui convient. Ces fonctions sont assez simples, à chaque lancement de l'une d'elle, le contenu de la case du tableau comp qui correspond au curseur à tourner est incrémenté et l'on utilise la méthode seth() du curseur pour l'aligner au nouvel angle (nous avons décidé arbitrairement qu'une rotation ferait 15 degrés). Il faut encore souligner que les fonctions rotations mettent aussi à jour comp

comme on l'a vu mais pour que détection marche, il faut que lorsque le curseur a fait un tour complet, sa case dans comp repasse à zéro. Par exemple, pour le carré, dés que comp vaut 6, sa position est la même que lors du démarrage de l'application (bien qu'il n'y ait pas eu un tour complet) on a donc tout de même comp[5] qui tombe à zéro, cela permet que chaque rotation est une unique valeur dans comp. Sans cela, la fonction detection() ne pourrait pas marcher.

Nous arrivons à la fonction reverse_para() qui comme son nom l'indique renverse le parallélogramme (en fait son symétrique). Son fonctionnement est simple : on regarde la forme courante de C_para grâce à la méthode shape(), si c'est la forme para1 on lui assigne la forme para2 (C_para.shape("para2")) sinon on fait l'inverse. Il y a par contre une chose dont on a pas encore parlée dans cette fonction, en effet on lui passe comme argument le mot clé "event". Celui-ci correspond en fait à l'objet passé par la méthode bind() d'un canvas Tkinter. Nous avons décidé d'utiliser celle-ci tout simplement car nous n'avons pas trouvé comment lancer une fonction avec onclick à la suite d'un double clic ce qui est facilement réalisable avec bind grâce au mot clé: "<Double-Button-1>". Il faut aussi souligner que cet objet event est très pratique puisque l'on peut récupérer des informations sur l'évènement comme les coordonnées x,y du clic qui sont les variables event.x et event.y.

Une autre fonction utilise ce type de binding, c'est la fonction where(), elle est appelée à chaque relâché de clic et a pour but de mettre à jour tabl. Ainsi, tabl possède toujours des positions misent à jour. De plus, where() appelle détection() pour vérifier après chaque mise à jour si le jeu est oui ou non terminé.

Le jeu est à présent parfaitement opérationnel. Cependant, il manque de contenu, nous allons donc voir comment marche la fonction level() qui permet de changer de niveau à partir du menu. En effet, en cliquant sur le bouton niveau du menu, vous voyez les quelques niveaux disponibles (comme canard et lapin). La fonction level() ne fait qu'affecter à tabl_end, comp_end et reverse_end les valeurs finales du niveau en plus de changer le fond d'écran du jeu pour que le joueur ait les contours de la figure à réaliser. Toutes les données nécessaires sont contenues dans des tableaux définis dans le bloc menu principal et qui dépendent de la résolution. Il y a donc quatre arguments possibles pour level : canard, prosterne, lapin et figure. Le premier élément de ce tableau correspond à tabl_end, le deuxième à comp_end, le troisième à reverse_end et enfin le

quatrième au chemin menant au fond correspondant au niveau choisi. À la fin de la fonction, restart() est appelée pour tout remettre en place avant le début du nouveau niveau.

Ce chapitre touche à sa fin, la dernière fonction que nous allons décrire est see(), elle permet d'aider le joueur s'il le demande. La fonction peut être appelée à tout moment à partir du menu aide mais elle ne peut l'être que deux fois par partie, elle sera donc disponible à nouveau après avoir fait fichier recommencer ou après avoir changé de niveau. Cela est permis grâce à la variable helped qui est réinitialisée à deux lors de l'exécution de restart(). Si helped est supérieur à 0 on tire un curseur au hasard, pour cela on les stocke dans un tableau et on tire un indice entre 0 et 6 pour pouvoir agir sur un curseur. Ensuite il faut déterminer dans quel niveau on est, pour cela on va utiliser la valeur courante du fond d'écran que l'on peut récupérer grâce à la méthode bgpic(). Il nous suffit maintenant de déplacer le curseur à son emplacement final en utilisant directement la position contenue dans les tableaux qui contiennent toutes les données des niveaux. De même on lui imposera sa rotation finale et si le curseur tiré est le parallélogramme on contrôlera s'il doit être ou non retourné et si c'est le cas, on appelle reverse_para qui se chargera de le faire.

Ce chapitre implémentation est donc fini, le code final est disponible en annexe et est commenté pour éclaircir certains points qui pourraient ne pas être clairs.

V/ Bonus

V.A. Le Taquin

Le jeu du Taquin est un puzzle, une image a été découpée en plusieurs carrés (ici 16) et un carré à été enlevé. Pour reconstituer l'image il faut donc déplacer les carrés en les échangeant avec la case vide.

Le fonctionnement de ce premier programme Bonus est très simple, à l'aide du module Tkinter nous avons créé 16 boutons qui représentent chacun une image. Lorsque l'on va cliquer sur un bouton, si celui-ci est à côté de la case vide on échange les positions avec la méthode place() du bouton.

Nous avons donc la technique mais il va nous falloir trouver un moyen de savoir où est la case vide et si le bouton sur lequel on clique est échangeable à celle-ci.

De plus un dernier problème va se poser, tous les taquins ne sont pas solubles ! Il nous faudra donc mélanger le Taquin de façon à ce qu'il y ait toujours une solution au puzzle.

Pour ce qui est de la disposition des cases, on utilise deux listes symétriques : position et bouton. Il est très important de toujours bien se représenter ces deux listes ! En effet l'une va contenir la case courante et donc son bouton associé (bouton) tandis que l'autre va contenir les coordonnées des cases (position). Pour chaque case d'indice i dans bouton ses coordonnées sont dans position[i]. Cela résou donc le problème de toujours savoir ou est la case vide. Il nous suffira de récupérer les coordonnées : position[16]. On voit aussi s'éclaircir le prototype de la fonction de déplacement que nous nommerons depl, on lui passera 3 arguments : le bouton sur lequel on a cliqué ainsi que les deux tableaux symétriques. Ainsi lorsque l'on clique sur un bouton, on récupère la position de ce dernier et de la case vide et si le déplacement est possible on l'effectue avec la méthode place() des boutons (cela permet donc d'utiliser le tableau position) sans oublier de mettre la table des positions à jour.

Nous pouvons donc à présent déplacer nos morceaux d'images. Pour détecter, que le jeu est terminé il nous suffira de comparer position et tab_end (qui contient les coordonnées du Taquin terminé) à chaque lâché de souris (avec la méthode bind()).

Le denier problème à résoudre est celui du mélange du Taquin. En effet, si l'on réalise un mélange aléatoire de la liste position le Taquin n'est pas forcément soluble. Nous avons donc dû chercher une solution, la plus simple à notre goût est la suivante : pour être sûr de pouvoir résoudre le Taquin, il faut le mélanger comme si l'on jouait. Pour mieux dire, avant d'afficher nos cases nous allons les mélanger en tirant au hasard l'une des cases échangeable avec la case vide et en permutant les deux positions. Nous allons donc créer un tableau qui contient les quatre cases qui entourent la vide (perm). On procède ainsi : on tire un élément au hasard dans perm, on regarde si ces coordonnées sont dans position (pour ne pas sortir du cadre du taquin) si elles sont bien présentent, on permute vide et cette case. Cependant, cette méthode ne marchait pas, en effet le hasard informatique n'étant pas parfait c'est toujours la même case qui était déplacée. La solution a était de simplement agrandir perm en dupliquant plusieurs fois chacun des éléments. On peut à présent mélanger le Taquin tout en étant certain de pouvoir le résoudre !

V.B. Les Allumettes

Dans ce jeu, nous avons utilisé le module Turtle. Tkinter n'est là que pour les 3 boutons.

Tout d'abord, nos allumettes sont toutes des curseurs turtle. Si le joueur clique sur une allumette, ça l'efface.

La principale difficulté était d'interdire la suppression des allumettes d'une certaine ligne quand celles d'une autre ligne avait précédemment été effacées lors d'un même tour (conformément au règles du jeu).

Pour ce faire, nous avons créé une variable ligne_select, qui indique quelle ligne est sélectionnée, et n'autorise la suppression d'une allumette que si elle est sur la même ligne que la première allumette effacée.

Ensuite, nous avons utilisé une astuce nous simplifiant grandement la gestion de nos curseurs, c'est le fait de les mettre tous dans une liste, et de parcourir la liste en permanence pour savoir si l'utilisateur a cliqué sur une allumette (fonction efface). La fonction pos() nous permettant d'avoir la position du centre de chaque curseur.

Puis à chaque fois que l'utilisateur efface une allumette, on vérifie la fonction gagner, qui s'exécute si la variable état, représentant le nombre d'allumette restante est égale a 0.

Enfin notre dernière fonction "new_part", permet de réinitialiser toute les variables et de replacer tous les curseurs afin de faire une nouvelle partie.

VI/ Conclusion

Pour conclure, ce projet nous aura été bénéfique en plusieurs points :

- ✗ nous aurons appris à lire une documentation
- ✗ à contourner les problèmes
- ✗ à persévérer
- ✗ à passer plusieurs heures sur une fonction puis à la remplacer par une tout autre beaucoup plus simple

Cependant, nous aurions pu améliorer certaines choses, par exemple dans la fonction see(), il est possible que l'aide s'effectue deux fois pour la même forme, ce qui implique qu'on a un seul coup d'aide au lieu de deux. Nous aurions aussi dû approfondir nos recherches dans la solutions pour les multiples fonctions rotation.

Il est important de préciser que l'étape la plus importante dans ce projet fut celle de la préparation avant l'implémentation, c'est à dire qu'on a avant toute chose noté sur papier notre algorithme, nos fonctions et surtout les intéractions (avec des petits schémas).

Finalement, notre solution n'est peut-être pas la meilleure, mais cela fonctionne très bien sans lags pour l'utilisateur.

VII/ Annexes

VII.A. Le Tangram : main.py :

```
#! /usr/bin/python
# -*- coding: utf-8 -*-
from turtle import *
from turtle import TK
from Tkinter import *
from random import randrange
import os
import pickle


#Fonctions



def distance(x,y):
    '''Cette fonction calcule la distance entre deux points x,y assimilables à des tuples'''
    dist=sqrt(((x[0]-y[0])**2) + ((x[1]-y[1])**2))
    return dist


def detection():
    '''Cette fonction détecte si le niveau est terminé ou non'''
    test = True
    #on initialise le le test comme étant vrai, si l'une des conditions du niveau n'est pas remplit il deviendra faux.
```

```
for i,pt in enumerate(tabl_end) :


    #premier test : les positions.
    if i == 0 :
        if distance(tabl[0],pt)<=5 or distance(tabl[1],pt)<=5:
            d1=distance(tabl[0],pt)
            d2=distance(tabl[1],pt)
            if d1<d2:
                C_tri1.goto(tabl_end[0])
            elif d1-d2 < 5:
                C_tri1.goto(tabl_end[0])
                C_tri2.goto(tabl_end[0])
            else:
                C_tri2.goto(tabl_end[0])
        else:
            test = False


    if i == 1 :
        if distance(tabl[1],pt)<=5 or distance(tabl[0],pt)<=5:
            d1=distance(tabl[0],pt)
            d2=distance(tabl[1],pt)
            if d1<d2:
                C_tri1.goto(tabl_end[1])
            elif d1-d2 < 5:
                C_tri1.goto(tabl_end[1])
```

```
            C_tri2.goto(tabl_end[1])
        else:
            C_tri2.goto(tabl_end[1])
    else:
        test = False
    #les deux grands triangles

if i == 3 :
    if distance(tabl[3],pt)<=5 or distance(tabl[4],pt)<=5:
        d1=distance(tabl[3],pt)
        d2=distance(tabl[4],pt)
        if d1<d2:
            C_tripe1.goto(tabl_end[3])
        elif d1-d2 < 5:
            C_tripe1.goto(tabl_end[3])
            C_tripe2.goto(tabl_end[3])
        else:
            C_tripe2.goto(tabl_end[3])
    else:
        test = False

if i == 4 :
    if distance(tabl[4],pt)<=5 or distance(tabl[3],pt)<=5:
        d1=distance(tabl[3],pt)
        d2=distance(tabl[4],pt)
```

```
            if d1<d2:
                C_tripe1.goto(tabl_end[4])
            elif d1-d2 < 5:
                C_tripe1.goto(tabl_end[4])
                C_tripe2.goto(tabl_end[4])
            else:
                C_tripe2.goto(tabl_end[4])
        else:
            test = False
        #les deux petits triangles
```

#Ces 4 premiers tests sont plus complexes car on ne sait pas quel triangle le joueur va choisir (2 emplacements pour chaque triangle).

```
    if i == 2:
        if distance(tabl[i],pt)<=5 :
            C_trimoy.goto(pt)
        else:
            test = False
        #le triangle moyen

    if i == 5:
        if distance(tabl[i],pt)<=5 :
            C_carre.goto(pt)
        else:
```

```
            test = False
        #le carré

    if i == 6:
        if distance(tabl[i],pt)<=5 :
            C_para.goto(pt)
        else:
            test = False
        #le parallélogramme

    #Cette première partie fait donc en sorte qu'un élément a une distance > à 5 de sa position finale se met à celle-ci

  for i,cpt in enumerate(comp):
    if i == 0 :
        if cpt != comp_end[0] and cpt != comp_end[1]:
            test = False
    if i == 1 :
        if cpt != comp_end[0] and cpt != comp_end[1]:
            test = False
    if i == 3:
        if cpt != comp_end[3] and cpt != comp_end[4]:
            test = False
    if i == 4:
        if cpt != comp_end[3] and cpt != comp_end[4]:
```

```
            test = False
    if i == 2:
        if cpt != comp_end[i] :
            test = False
    if i == 5:
        if cpt != comp_end[i] :
            test = False
    if i == 6:
        if cpt != comp_end[i] :
            test = False
#deuxième test qui regarde si la rotation des éléments est la bonne

if reverse != reverse_end :
    test = False
#Dernier test vérifiant si le parallélogramme doit ou non être inversé

s1.update()

if test == True:
    #le test est vrai, annonçons le au joueur
    fen3 = Toplevel()
    fen3.title("Bravo !")

    fond = Label (fen3,image = image6)
    #Initialisation d'une fenêtre qui a un fond que l'on affiche grâce à un label ( qui est ici
```

```
une image )
        res = Button (fond,image = image7,command = lambda : [restart(),
                                                fen3.destroy()
                                                ]
                    )
        by = Button (fond,image = image8,command = fen.quit)

        #création d'un bouton recommencer et d'un bouton quitter ( qui quitte le jeu
entièrement)

        fond.pack()
        res.place(x=xres,y=yres)
        by.place(x=xby,y=yby)
        posx = fen3.winfo_screenwidth()
        posy = fen3.winfo_screenheight()
        diffx=T_fen_end
        diffy=T_fen_end2
        x = (posx/2) - (diffx/2)
        y = (posy/2) - (diffy/2)
        fen3.geometry('%dx%d+%d+%d'%(diffx,diffy,x,y))
        #affichage et centrage de la fenêtre.

def restart():
    '''Cette fonction permet de recommencer un niveau'''
```

```
    global comp,tabl,reverse,helped
    #récupération des variables à modifier en tant que variable globale

    tabl = list(carre[0])
    comp = list(carre[1])
    reverse = carre[2]
    #modification (ou création) du tableau de position et du tableau de rotation courant ainsi que de la variable indiquant si le parallélogramme est inversé
    helped = 2
    #la variable helped est remise à deux pour pouvoir réutiliser l'aide ( voir fonction see())
    init()
    #appel de la fonction init pour que les changements prennent effets

def init():
    '''Cette fonction initialise les éléments avec leur position et rotation respective qui sont dans comp et tabl'''

    C_tri1.up()
    C_tri2.up()
    C_tripe1.up()
    C_tripe2.up()
    C_trimoy.up()
    C_carre.up()
    C_para.up()

    C_tri1.shape("tri1")
```

```
    C_tri2.shape("tri2")
    C_tripe1.shape("tripe1")
    C_tripe2.shape("tripe2")
    C_trimoy.shape("trimoy")
    C_carre.shape("carre")
    C_para.shape("para1")

    C_tri1.seth(comp[0]*15)
    C_tri1.goto(tabl[0])
    C_tri2.seth(comp[1]*15)
    C_tri2.goto(tabl[1])
    C_trimoy.seth(comp[2]*15)
    C_trimoy.goto(tabl[2])
    C_tripe1.seth(comp[3]*15)
    C_tripe1.goto(tabl[3])
    C_tripe2.seth(comp[4]*15)
    C_tripe2.goto(tabl[4])
    C_carre.seth(comp[5]*15)
    C_carre.goto(tabl[5])
    C_para.seth(comp[6]*15)
    C_para.goto(tabl[6])
    color_init()
    #appel de la fonction color_init pour récupérer et afficher les couleurs des éléments

def color_init():
```

```
    '''Cette fonction met à jour la couleur des éléments du tangram (permet au joueur de personnaliser son jeu'''
    C_para.color(C_color7.get())
    C_carre.color(C_color6.get())
    C_tripe2.color(C_color5.get())
    C_tripe1.color(C_color4.get())
    C_trimoy.color(C_color3.get())
    C_tri2.color(C_color2.get())
    C_tri1.color(C_color1.get())

    s1.update()

def sav_color():
    '''Cette fonction récupère et sauvegarde dans un fichier les couleurs choisies par le joueur pour son tangram'''
    with open("sav_color.txt","wb") as sauvegarde_couleur:
        sav = pickle.Pickler(sauvegarde_couleur)
        sav.dump(C_color1.get())
        sav.dump(C_color2.get())
        sav.dump(C_color3.get())
        sav.dump(C_color4.get())
        sav.dump(C_color5.get())
        sav.dump(C_color6.get())
        sav.dump(C_color7.get())

def sav_res(res):
```

```
    '''Cette fonction change la résolution par défaut par celle sélectionnée par le joueur et la place dans le fichier qui est lu au démarage du jeu
pour initialiser les fenêtres'''
    with open("sav_res.txt","wb") as sauvegarde_resolution:
        sav = pickle.Pickler(sauvegarde_resolution)
        sav.dump(res)

    advise = Toplevel()
    redem = Label(advise,text = "Vous devez relancer l'application pour que les changements prennent effet !")
    btn_quit = Button(advise,text = "Fermer l'application",command = fen.quit)
    redem.pack()
    btn_quit.pack()

    posx = advise.winfo_screenwidth()
    posy = advise.winfo_screenheight()
    diffx=500
    diffy=50
    x = (posx/2) - (diffx/2)
    y = (posy/2) - (diffy/2)
    advise.geometry('%dx%d+%d+%d'%(diffx,diffy,x,y))
    #création et centrage d'une fenêtre demandant au joueur de redémarrer l'application pour que les changements prennent effet


def rotation0(x,y):
```

```
    '''fonction de rotation pour C_tri1'''
    comp[0]+=1
    C_tri1.seth(comp[0]*15)
    if comp[0]==24 :
        comp[0]=0
    s1.update()

def rotation1(x,y):
    '''fonction de rotation pour C_tri2'''
    comp[1]+=1
    C_tri2.seth(comp[1]*15)
    if comp[1]==24 :
        comp[1]=0
    s1.update()

def rotation2(x,y):
    '''fonction de rotation pour C_trimoy'''
    comp[2]+=1
    C_trimoy.seth(comp[2]*15)
    if comp[2]==24 :
        comp[2]=0
    s1.update()

def rotation3(x,y):
    '''fonction de rotation pour C_tripe1'''
```

```
    comp[3]+=1
    C_tripe1.seth(comp[3]*15)
    if comp[3]==24 :
        comp[3]=0
    s1.update()

def rotation4(x,y):
    '''fonction de rotation pour C_tripe2'''
    comp[4]+=1
    C_tripe2.seth(comp[4]*15)
    if comp[4]==24 :
        comp[4]=0
    s1.update()

def rotation5(x,y):
    '''fonction de rotation pour C_carre'''
    comp[5]+=1
    C_carre.seth(comp[5]*15)
    if comp[5]==6 :
        comp[5]=0
    s1.update()

def rotation6(x,y):
    '''fonction de rotation pour C_para'''
    comp[6]+=1
```

```
    C_para.seth(comp[6]*15)
    if comp[6]==12 :
            comp[6]=0
    s1.update()


'''les 6 fonctions rotations sont necessaires car on ne peut pas récupérer l'objet curseur actif au moment du clic de plus,
une fonction lambda ne peut pas recevoir d'argument ce qui rend son utilisation avec onclick impossible
Ces dernières mettent aussi à jour la table des rotation (comp)'''


def reverse_para(event):
    '''Cette fonction récupere la forme de l'objet C_para et lui assigne sa forme symétrique'''
    global reverse


    forme=C_para.shape()
    if forme=="para1":
        reverse=True
        C_para.shape("para2")
    if forme=="para2":
        reverse=False
        C_para.shape("para1")
    s1.update()


def where(event):
    '''Cette fonction met a jour la table des positions (tabl)'''
```

```
    tabl[0]=C_tri1.position()
    tabl[1]=C_tri2.position()
    tabl[2]=C_trimoy.position()
    tabl[3]=C_tripe1.position()
    tabl[4]=C_tripe2.position()
    tabl[5]=C_carre.position()
    tabl[6]=C_para.position()

    detection()
    #lance la fonction détection pour voir si le tangram est terminé


def level(lvl):
    '''Cette fonction configure le jeu pour afficher le niveau choisi par l'utilisateur'''
    global tabl_end,comp_end,reverse_end

    tabl_end = lvl[0]
    comp_end = lvl[1]
    reverse_end= lvl[2]
    bgc = lvl[3]
    s1.bgpic(bgc)
    #mise de tabl_end comp_end et reverse_end qui sont utilisées dans detection et mise à jour du fond d'écran
```

```
def see():
    '''Cette fonction permet au joueur d'obtenir de l'aide si le joueur en demande'''
    global helped

    helped -= 1
    #on décrémente helped (appelé en global) car l'aide est limitée à deux par jeu

    if helped >= 0:
        #si le joueur à droit à un peu d'aide, on l'aide
        curseur = [C_tri1,C_tri2,C_trimoy,C_tripe1,C_tripe2,C_carre,C_para]
        fond = s1.bgpic()
        #récupération du fond d'écran (permet de savoir à quel niveau on est
        cpt = len(curseur)
        j = randrange(0,cpt)
        #on tire un curseur au hasard
        if fond == "HT/1000x700/canard.gif" or fond == "HT/500x350/canard.gif":
                curseur[j].goto(canard[0][j])
                curseur[j].seth(canard[1][j]*15)
                tabl[j] = canard[0][j]
                comp[j] = canard[1][j]
                if reverse != canard[3] and (curseur[j].shape()=="para1" or
curseur[j].shape()=="para2"):
                    reverse_para(False)
        if fond == "HT/1000x700/lapin.gif" or fond == "HT/500x350/lapin.gif":
                curseur[j].goto(lapin[0][j])
```

```
            curseur[j].seth(lapin[1][j]*15)
            tabl[j] = lapin[0][j]
            comp[j] = lapin[1][j]
            if reverse != lapin[3] and (curseur[j].shape()=="para1" or
curseur[j].shape()=="para2"):
                reverse_para(False)
        if fond == "HT/1000x700/prosterne.gif" or fond == "HT/500x350/prosterne.gif":
            curseur[j].goto(prosterne[0][j])
            curseur[j].seth(prosterne[1][j]*15)
            tabl[j] = prosterne[0][j]
            comp[j] = prosterne[1][j]
            if reverse != prosterne[3] and (curseur[j].shape()=="para1" or
curseur[j].shape()=="para2"):
                reverse_para(False)
        if fond == "HT/1000x700/figure.gif" or fond == "HT/500x350/figure.gif":
            curseur[j].goto(figure[0][j])
            curseur[j].seth(figure[1][j]*15)
            tabl[j] = figure[0][j]
            comp[j] = figure[1][j]
            if reverse != figure[3] and (curseur[j].shape()=="para1" or
curseur[j].shape()=="para2"):
                reverse_para(False)
        #pour le niveau courant, on met le curseur tiré au hasard à sa place on le tourne
correctement et on met à jour la table des positions et des rotations
        #si le curseur tiré est C_para, on regarde en plus s'il doit être retourné et on lance la
fonction reverse_para qui se charge de mettre à jour reverse
```

```
    else:
        fen4=Toplevel()
        attention = Label(fen4,text = "Attention vous avez utilisé l'aide trop de fois ! ")
        btnok = Button(fen4,text = "OK", command = fen4.destroy)
        attention.pack()
        btnok.pack()

        posx = fen4.winfo_screenwidth()
        posy = fen4.winfo_screenheight()
        diffx=300
        diffy=50
        x = (posx/2) - (diffx/2)
        y = (posy/2) - (diffy/2)
        fen4.geometry('%dx%d+%d+%d'%(diffx,diffy,x,y))

        #si le joueur a utilisé deux fois l'aide, on lui signale qu'il ne peut plus utiliser cette
fonction pour le moment à l'aide d'une fenetre

    s1.update()



def Tangram():
    '''Cette fonction est le coeur du programme elle gère le tangram même'''
    #Init
    global
s1,C_tri1,C_tri2,C_trimoy,C_tripe1,C_tripe2,C_carre,C_para,C_color1,C_color2,C_color3,
```

C_color4,C_color5,C_color6,C_color7,helped

```
    fen2=Toplevel()
    #Création d'une nouvelle fenètre
    cv1 = Canvas(fen2, width=T_fen, height=T_fen2)
    s1 = TurtleScreen(cv1)
    p = RawTurtle(s1)
    #Création d'un canvas Tkinter dans lequel on crée un écran turtle et un premier curseur nommé p.

    C_tri1 = RawTurtle(s1)
    C_tri2= RawTurtle(s1)
    C_trimoy= RawTurtle(s1)
    C_tripe1= RawTurtle(s1)
    C_tripe2= RawTurtle(s1)
    C_carre= RawTurtle(s1)
    C_para= RawTurtle(s1)
    #création des curseurs qui seront les éléments du Tangram

    cv1.pack()
    #affichage du canvas(et donc de l'écran turtle)

    C_color1 = StringVar()
    C_color2 = StringVar()
    C_color3 = StringVar()
```

```
    C_color4 = StringVar()
    C_color5 = StringVar()
    C_color6 = StringVar()
    C_color7 = StringVar()
    #création des objets qui contiendront les couleurs des curseurs, on utilise ici des objets pour permettre la mise à jour simple des couleurs grâce à des radiobutton

    with open("sav_color.txt","rb") as sauvegarde_couleur:
        color_val = pickle.Unpickler(sauvegarde_couleur)
        colorset[0] = color_val.load()
        colorset[1] = color_val.load()
        colorset[2] = color_val.load()
        colorset[3] = color_val.load()
        colorset[4] = color_val.load()
        colorset[5] = color_val.load()
        colorset[6] = color_val.load()
    #lecture des couleurs sauvegardées par le joueur dans le fichier sav_color.txt

    C_color1.set(colorset[0])
    C_color2.set(colorset[1])
    C_color3.set(colorset[2])
    C_color4.set(colorset[3])
    C_color5.set(colorset[4])
    C_color6.set(colorset[5])
    C_color7.set(colorset[6])
```

```
    #assignation des chaînes contenant le nom des couleurs aux objets créés plus tôt


    #Nous allons à présent créer le menu qui est situé en haut de la fenêtre de jeu
    menu1 = Menu(fen2)


    fichier = Menu(menu1,tearoff=0)
    menu1.add_cascade(label="Fichier",menu=fichier)
    fichier.add_command(label="Recommencer",command=restart)
    #la commande Fichier/recommencer permet au joueur de remettre son niveau à 0 grâce à la fonction restart
    options = Menu(fichier,tearoff=0)
    fichier.add_cascade(label="Options",menu = options)
    color = Menu(options,tearoff=1)
    options.add_cascade(label ="Couleur",menu = color)
    triangle1 = Menu(options,tearoff=0)
    triangle2 = Menu(options,tearoff=0)
    trianglemoy = Menu(options,tearoff=0)
    trianglepe1 = Menu(options,tearoff=0)
    trianglepe2 = Menu(options,tearoff=0)
    carree = Menu(options,tearoff=0)
    paral = Menu(options,tearoff=0)
    #ici nous allons définir un menu pour chaque élément du tangram qui permetra au joueur de personnaliser les couleurs de son jeu qui sont automatiquement mis à jour grâce à la fonction color_init()


    liste_init = [["Triangle 1",triangle1,C_color1],["Triangle 2",triangle2,C_color2],["Triangle
```

```
Moyen",trianglemoy,C_color3],["Petit Triangle 1",trianglepe1,C_color4],["Petit Triangle
2",trianglepe2,C_color5],["Carre",carree,C_color6],["Parallélogramme",paral,C_color7]]

    liste_init_color = [["Rouge","red"],["Bleu","blue"],["Vert","green"],["Jaune","yellow"],
["Rose","pink"],["Violet","purple"],["Noir","black"],["Blanc","white"],["Orange","orange"],
["Bleu Ciel","skyblue"],["Or","gold"],["Marron","brown"]]


    for i in liste_init:
        color.add_cascade(label = i[0],menu = i[1])
        for j in liste_init_color:
            i[1].add_radiobutton(label = j[0],value = j[1],variable = i[2],command = color_init)


    color.add_separator()
    color.add_command(label = "Défaut",command = lambda :
[ C_color1.set(color_default[0]),
                                        C_color2.set(color_default[1]),
                                        C_color3.set(color_default[2]),
                                        C_color4.set(color_default[3]),
                                        C_color5.set(color_default[4]),
                                        C_color6.set(color_default[5]),
                                        C_color7.set(color_default[6]),
                                        color_init()
                                        ])
    #permet au joueur de remettre les couleurs par défaut du jeu
    color.add_separator()
    color.add_command(label = "Sauvegarder",command = sav_color)
    #sauvegarde la configuration des couleurs grâce a la fonction
```

```
    resolution = Menu(options,tearoff=0)
    options.add_cascade(label ="Résolution",menu = resolution)
    resolution.add_command(label="1000x700",command = lambda : sav_res("1000x700"))
    resolution.add_command(label="500x350",command = lambda : sav_res("500x350"))
    #permet de modifier la résolution courante
    fichier.add_command(label="Quitter",command=fen.quit)
    #permet au joueur de quitter complètement le programme

    niveaux = Menu(menu1,tearoff=0)
    menu1.add_cascade(label="Niveau",menu=niveaux)

    niveaux.add_command(label="Canard",command = lambda : level(canard))
    niveaux.add_command(label="Lapin",command = lambda : level(lapin))
    niveaux.add_command(label="Homme prosterné",command = lambda :
level(prosterne))
    niveaux.add_command(label="Homme Rigolant",command = lambda : level(figure))
    #permet de choisir le niveau

    aide = Menu(menu1,tearoff=0)
    menu1.add_cascade(label="Aide",menu=aide)

    aide.add_command(label = "Coups de pouce : 2 par partie",command = see)
    #lance la fonction see pour placer un des éléments du Tangram
    aide.add_command(label = "Comment Jouer ?",command = lambda : os.system("gedit
comment_jouer.txt"))
    #affiche dans gedit le fichier commen_jouer.txt
```

```
    bonus = Menu(menu1,tearoff=0)
    menu1.add_cascade(label="Bonus",menu=bonus)
    bonus.add_command(label = "Taquin",command = lambda : os.system("python Taquin/taquin.py"))
    bonus.add_command(label = "Allumettes",command = lambda : os.system("python Allumettes/allumettes.py"))
    #permet de lancer les scripts contenant les jeux bonus : le jeu de Allumette et le Taquin

    fen2.config(menu=menu1)
    #affectation du menu à notre fenêtre de jeu

    posx = fen2.winfo_screenwidth()
    posy = fen2.winfo_screenheight()
    diffx=T_fen
    diffy=T_fen2
    x = (posx/2) - (diffx/2)
    y = (posy/2) - (diffy/2)
    fen2.geometry('%dx%d+%d+%d'%(diffx,diffy,x,y))
    #centrage de la fenêtre de jeu

    p.tracer(8,25)
    #configure la vitesse de tracer du curseur p de façon à ce que l'on ne remarque pas l'initialisation des curseurs

    #Création des curseurs
```

```
p.ht()
p.up()

p.begin_poly()
p.fd(cote/4)
p.right(135)
p.fd((cote*sqrt(2))/4)
p.right(90)
p.fd((cote*sqrt(2))/4)
p.seth(0)
p.fd(cote/4)
p.end_poly()
tripe1 = p.get_poly()
s1.register_shape("tripe1", tripe1)
s1.register_shape("tripe2", tripe1)

p.begin_poly()
p.fd(cote/2)
p.right(135)
p.fd((cote*sqrt(2))/2)
p.right(90)
p.fd((cote*sqrt(2))/2)
p.seth(0)
p.fd(cote/2)
```

```
p.end_poly()
tri1 = p.get_poly()
s1.register_shape("tri1", tri1)
s1.register_shape("tri2", tri1)

p.begin_poly()
p.left(90)
p.fd(sqrt((cote/2)**2 + (cote/2)**2)/4)
p.right(90)
p.fd(cote/8)
p.right(90)
p.fd(cote/4)
p.seth(225)
p.fd(sqrt((cote/2)**2 + (cote/2)**2)/2)
p.seth(90)
p.fd(cote/2)
p.seth(45)
p.fd(sqrt((cote/2)**2 + (cote/2)**2)/2)
p.seth(270)
p.fd(cote/4)
p.right(90)
p.fd(cote/8)
p.goto(0,0)
p.seth(0)
p.end_poly()
```

```
para1 = p.get_poly()
s1.register_shape("para1", para1)


p.begin_poly()
p.right(90)
p.fd(sqrt((cote/2)**2 + (cote/2)**2)/4)
p.left(90)
p.fd(cote/8)
p.right(90)
p.fd(cote/4)
p.seth(135)
p.fd(sqrt((cote/2)**2 + (cote/2)**2)/2)
p.seth(90)
p.fd(cote/2)
p.seth(315)
p.fd(sqrt((cote/2)**2 + (cote/2)**2)/2)
p.seth(0)
p.right(90)
p.fd(cote/4)
p.right(90)
p.fd(cote/8)
p.goto(0,0)
p.end_poly()
para2 = p.get_poly()
s1.register_shape("para2", para2)
```

```
p.begin_poly()
p.left(90)
p.fd(sqrt((cote/2)**2 + (cote/2)**2)/4)
p.right(90)
p.fd(sqrt((cote/2)**2 + (cote/2)**2)/4)
p.right(90)
p.fd(sqrt((cote/2)**2 + (cote/2)**2)/2)
p.right(90)
p.fd(sqrt((cote/2)**2 + (cote/2)**2)/2)
p.right(90)
p.fd(sqrt((cote/2)**2 + (cote/2)**2)/2)
p.right(90)
p.fd(sqrt((cote/2)**2 + (cote/2)**2)/4)
p.right(90)
p.fd(sqrt((cote/2)**2 + (cote/2)**2)/4)
p.seth(0)
p.end_poly()
carre = p.get_poly()
s1.register_shape("carre", carre)

p.begin_poly()
p.left(45)
p.fd(sqrt((cote/2)**2 + (cote/2)**2)/2)
p.right(135)
```

```
    p.fd(cote/2)
    p.right(90)
    p.fd(cote/2)
    p.goto(0,0)
    p.end_poly()
    trimoy=p.get_poly()
    s1.register_shape("trimoy", trimoy)

    #pour créer les curseurs on enregistre une forme que l'on dessine avec le curseur p puis on l'enregistre en temps que shape pour notre écran turtle

    #Corps

    level(canard)
    #on choisi le niveau par défaut

    #le jeu est prêt on fait donc un restart pour que l'initialisation se termine et que le joueur puisse commencer la partie
    restart()

    C_tri1.onclick(rotation0,btn=3)
    C_tri2.onclick(rotation1,btn=3)
    C_trimoy.onclick(rotation2,btn=3)
    C_tripe1.onclick(rotation3,btn=3)
    C_tripe2.onclick(rotation4,btn=3)
    C_carre.onclick(rotation5,btn=3)
```

```
    C_para.onclick(rotation6,btn=3)


    C_tri1.ondrag(C_tri1.goto)
    C_tri2.ondrag(C_tri2.goto)
    C_tripe1.ondrag(C_tripe1.goto)
    C_tripe2.ondrag(C_tripe2.goto)
    C_trimoy.ondrag(C_trimoy.goto)
    C_carre.ondrag(C_carre.goto)
    C_para.ondrag(C_para.goto)
    #initialisation des bindings pour que le joueur puisse faire tourner l'élément et le déplacer


    cv1.bind("<ButtonRelease-1>", where)
    cv1.bind("<Double-Button-1>",reverse_para)
    # 2 derniers binds pour metre à jour tabl et pour permettre au joueur de faire le symétrique du parallélogramme


    s1.listen()
    #il ne reste plus qu'à attendre un évenement


#Création des première variable : les couleurs, les couleurs par défaut et le nombre d'aides disponibles


colorset = ["brown","purple","pink","yellow","blue","red","green"]


color_default = ["brown","purple","pink","yellow","blue","red","green"]
```

```
helped = 2

#Première fenêtre

fen=TK.Tk()
fen.title("Tangram Project")

#Initialisation de la résolution

with open("sav_res.txt","rb") as sauvegarde_resolution:
        res_val = pickle.Unpickler(sauvegarde_resolution)
        res = res_val.load()
#lecture dans sav_res.txt de la résolution du programme ( par défaut 1000x700 )

if res == "1000x700":
    image1 = PhotoImage(file="HT/1000x700/Tangram.gif")
    image2 = PhotoImage(file="HT/1000x700/main.gif")
    image3 = PhotoImage(file="HT/1000x700/Jouer.gif")
    image4 = PhotoImage(file="HT/1000x700/Crédits.gif")
    image5 = PhotoImage(file="HT/1000x700/Quitter.gif")
    image6 = PhotoImage(file="HT/1000x700/end.gif")
    image7 = PhotoImage(file="HT/1000x700/reco.gif")
    image8 = PhotoImage(file="HT/1000x700/quitter_end.gif")
    #importation des images qui dépendent de la résolution (le fond des fenètres)
```

T_fen = 1000

T_fen2 = 700

T_fen_end = 300

T_fen_end2 = 210

#configuration de la taille des fenêtres principales et de la fenêtre de fin de jeu

cote = 200

#taille du coté principal des éléments du tangram

xbtn = 410

ybtn = 252

xbtn1 = 400

ybtn1 = 375

xbtn2 = 400

ybtn2 = 490

xres = 5

yres = 140

xby = 180

yby = 137

#configuration des emplacements des diffèrents bouton qui dépendent de la résolution contenu dans les fenetre

carre = [[(409.00,2.00), (308.00,-99.00), (257.00,52.00), (358.00,103.00), (258.00,1.00), (308.00,53.00), (232.00,-34.00)],[0, 18, 18, 6, 12, 3, 6],False]

canard = [[(-161.00,-21.00), (-160.00,-21.00), (-262.00,-91.00), (-384.00,10.00), (-60.00,30.00), (-334.00,61.00), (-309.00,-25.00)],[0, 12, 15, 12, 6, 3, 6],False,"HT/1000x700/canard.gif"]

prosterne = [[(-230.00,26.00), (-154.00,18.00), (-292.00,-21.00), (-304.00,28.00), (-86.00,-107.00), (-39.00,-25.00), (-319.00,-55.00)],[16, 13, 13, 19, 16, 2, 1],False,"HT/1000x700/prosterne.gif"]

lapin = [[(-281.00,-115.00), (-251.00,-44.00), (-253.00,128.00), (-200.00,-81.00), (-250.00,-30.00), (-216.00,59.00), (-198.00,137.00)],[3, 0, 21, 0, 12, 0, 4],False,"HT/1000x700/lapin.gif"]

figure = [[(-223.00,16.00), (-240.00,-111.00), (-228.00,138.00), (-285.00,26.00), (-322.00,33.00), (-219.00,-39.00), (-184.00,131.00)],[15, 9, 0, 12, 9, 3, 9],True,"HT/1000x700/figure.gif"]

#tableaux contenant toutes les informations pour la réalisation des niveaux (1 : emplacement,2 : rotation, 3: valeur de reverse, 4: emplacement de l'image de fond associées)

else:

image1 = PhotoImage(file="HT/500x350/Tangram.gif")

image2 = PhotoImage(file="HT/500x350/main.gif")

image3 = PhotoImage(file="HT/500x350/Jouer.gif")

image4 = PhotoImage(file="HT/500x350/Crédits.gif")

image5 = PhotoImage(file="HT/500x350/Quitter.gif")

image6 = PhotoImage(file="HT/500x350/end.gif")

image7 = PhotoImage(file="HT/500x350/reco.gif")

image8 = PhotoImage(file="HT/500x350/quitter_end.gif")

#importation des images qui dépendent des de la résolution (le fond des fenètres)

T_fen = 500

T_fen2 = 350

T_fen_end = 150

T_fen_end2 = 105

#configuration de la taille des fenètres principales et de la fenetre de fin de jeu

cote = 100

#taille du coté principal des éléments du tangram

```
    xbtn = 205

    ybtn = 126

    xbtn1 = 200

    ybtn1 = 188

    xbtn2 = 200

    ybtn2 = 245

    xres = 3

    yres = 70

    xby = 90

    yby = 69

    #configuration des emplacements des différents boutons qui dépendent de la résolution contenue dans les fenêtres

    carre =  [[(204.5, 1.0), (154.0, -49.5), (128.5, 26.0), (179.0, 51.5), (129.0, 0.5), (154.0, 26.5), (116.0, -17.0)],[0, 18, 18, 6, 12, 3, 6],False]

    canard = [[(-80.5, -10.5), (-80.0, -10.5), (-131.0, -45.5), (-192.0, 5.0), (-30.0, 15.0), (-167.0, 30.5), (-154.5, -12.5)],[0, 12, 15, 12, 6, 3, 6],False,"HT/500x350/canard.gif"]

    prosterne =  [[(-115.0, 13.0), (-77.0, 9.0), (-146.0, -10.5), (-152.0, 14.0), (-43.0, -53.5), (-19.5, -12.5), (-159.5, -27.5)],[16, 13, 13, 19, 16, 2, 1],False,"HT/500x350/prosterne.gif"]

    lapin = [[(-140.5, -57.5), (-125.5, -22.0), (-126.5, 64.0), (-100.0, -40.5), (-125.0, -15.0), (-108.0, 29.5), (-99.0, 68.5)],[3, 0, 21, 0, 12, 0, 4],False,"HT/500x350/lapin.gif"]

    figure = [[(-111.5, 8.0), (-120.0, -55.5), (-114.0, 69.0), (-142.5, 13.0), (-161.0, 16.5), (-109.5, -19.5), (-92.0, 65.5)],[15, 9, 0, 12, 9, 3, 9],True,"HT/500x350/figure.gif"]

    #tableaux contenant toutes les informations pour la réalisation des niveaux ( 1 : emplacement,2 : rotation, 3: valeur de reverse, 4: emplacement de l'image de fond associée)
```

```
#Fin de l'initialisation de la fenêtre


lab0 = Label(fen,image=image1 )

#Ce label contient l'image de fond ( la première de l'animation)

btn = Button(lab0,text="Jouer",command = Tangram,image = image3)

btn1 = Button(lab0,text="Crédits",command = lambda : os.system("gedit
Crédits.txt"),image = image4)

btn2 = Button(lab0,text="Quitter",command=fen.quit,image = image5)

#Création de 3 boutons permettant de jouer au tangram, de quitter le jeu ou d'afficher
dans gedit le fichier crédits.txt


posx = fen.winfo_screenwidth()

posy = fen.winfo_screenheight()

#récupération de la position de la fenêtre ( ne marche que sous linux )

diffx=T_fen

diffy=T_fen2

x = (posx/2) - (diffx/2)

y = (posy/2) - (diffy/2)

fen.geometry('%dx%d+%d+%d'%(diffx,diffy,x,y))

#centrage de la fenêtre


lab0.pack(side='top', fill='both', expand='yes')

#affichage du fond


fen.after(2500, lambda : [lab0.configure(image = image2),

                    btn.place(x=xbtn,y=ybtn),
```

```
                btn1.place(x=xbtn1,y=ybtn1),
                btn2.place(x=xbtn2,y=ybtn2)]
        )
#après 2,5 seconde, on change l'image de fond et on affiche les boutons configurés plus haut, cette ligne permet de créer l'animation du début

fen.mainloop()
fen.destroy()
#on entre dans la boucle principale et on détruit la fenêtre principale si l'on en sort.
```

VII.B. Taquin.py

```
#! /usr/bin/python
# -*- coding: utf-8 -*-
from Tkinter import *
from random import *

def init(position,bouton):
    '''Cette fonction mélange le taquin'''
    for i in range(1000):

        vide = position[15]
        #on récupère la position de la case vide
        perm = [(vide[0]+200,vide[1]),(vide[0],vide[1]-200),(vide[0]-200,vide[1]),
(vide[0],vide[1]+200),(vide[0]+200,vide[1]),(vide[0],vide[1]-200),(vide[0]-200,vide[1]),
(vide[0],vide[1]+200),(vide[0]+200,vide[1]),(vide[0],vide[1]-200),(vide[0]-200,vide[1]),
```

```
(vide[0],vide[1]+200)]
    #perm contient les positions où la case vide va pouvoir aller chaque positions y est plusieurs fois pour augmenter l'impression de hasard de randrange
    count = len(perm)
    j = randrange(0,count-1)

    while perm[j] not in position:
        j = randrange(0,count-1)
        #on tire une position au hasard jusqu'à en avoir une où on peut mettre la case vide (il ne faut que cela soit en dehors de l'écran)

    current = perm[j]
    #on récupere la future position de la case vide
    btn = bouton[position.index(current)]
    #on regarde quel morceau de l'image est à cette position
    btn.place(x=vide[0],y=vide[1])
    #on met cette image à la place de la case vide
    position[15]=current
    position[bouton.index(btn)]=vide
    #on met à jour la table des positions

def depl(btn,position,bouton):
    '''Cette fonction déplace une image'''
    vide = position[15]
    #on récupère la position de la case vide
    current = position[bouton.index(btn)]
```

```
    #on récupère la position de la case sur laquelle on a cliqué
    if (abs(current[0] - vide[0]) <= 5 and abs(current[1] - vide[1]) <= 210) or (abs(current[1] - vide[1]) <= 5 and abs(current[0] - vide[0]) <= 210):
        #si les deux cases sont voisines :
        btn.place(x=vide[0],y=vide[1])
        #on met la case à la place de la case vide
        position[15]=current
        position[bouton.index(btn)]=vide
         #on met à jour la table des positions


def detec(event):
    '''Cette fonction détecte si le joueur a gagné'''
    global position,tab_end,fenetre,bouton

    if position == tab_end:
        #si les positions courantes sont les mêmes que les positions du Taquin fini :
        fenetre_fin=Toplevel(fenetre)
        gagne = Label(fenetre_fin,text = "Félicitations vous avez réussi !")
        btngagne = Button(fenetre_fin,text = "OK", command = fenetre_fin.destroy)
        bouton[15].place(x=0,y=0)
        #on crée une fenêtre pour notifier le joueur et on affiche l'image non découpée
        posx = fenetre_fin.winfo_screenwidth()
        posy = fenetre_fin.winfo_screenheight()
        diffx=200
        diffy=50
```

```
        x = (posx/2) - (diffx/2)
        y = (posy/2) - (diffy/2)
        fenetre_fin.geometry('%dx%d+%d+%d'%(diffx,diffy,x,y))
        #on centre le fenêtre

        gagne.pack()
        btngagne.pack()
        #on affiche la fenêtre de notification et son contenu

#fenêtre principale
fenetre=Tk()
fenetre.title("Taquin Project")
tab_end = [(0.00,0.00), (200.00,0.00), (400.00,0.00), (600.00,0.00), (0.00,200.00),
(200.00,200.00),(400.00,200.00),(600.00,200.00),(0.00,400.00),(200.00,400.00),
(400.00,400.00),(600.00,400.00),(0.00,600.00),(200.00,600.00),(400.00,600.00),
(600.00,600.00)]
#Création de la table de position fini

posx = fenetre.winfo_screenwidth()
posy = fenetre.winfo_screenheight()
diffx=800
diffy=800
x = (posx/2) - (diffx/2)
y = (posy/2) - (diffy/2)
fenetre.geometry('%dx%d+%d+%d'%(diffx,diffy,x,y))
#on centre le fenêtre
```

```
case1 = PhotoImage(file="Taquin/1.gif")
case2 = PhotoImage(file="Taquin/2.gif")
case3 = PhotoImage(file="Taquin/3.gif")
case4 = PhotoImage(file="Taquin/4.gif")
case5 = PhotoImage(file="Taquin/5.gif")
case6 = PhotoImage(file="Taquin/6.gif")
case7 = PhotoImage(file="Taquin/7.gif")
case8 = PhotoImage(file="Taquin/8.gif")
case9 = PhotoImage(file="Taquin/9.gif")
case10 = PhotoImage(file="Taquin/10.gif")
case11 = PhotoImage(file="Taquin/11.gif")
case12 = PhotoImage(file="Taquin/12.gif")
case13 = PhotoImage(file="Taquin/13.gif")
case14 = PhotoImage(file="Taquin/14.gif")
case15 = PhotoImage(file="Taquin/15.gif")
taquin = PhotoImage(file="Taquin/Taquin.gif")
#on importe les images du Taquin(le jeu devant être lancé depuis le Tangram on lui indique le dossier à partir du dossier contenant main.py

img0 = Button(text = "0",command = lambda : depl(img0,position,bouton),image = case1)
img1 = Button(text = "1",command = lambda : depl(img1,position,bouton),image = case2)
img2 = Button(text = "2",command = lambda : depl(img2,position,bouton),image = case3)
img3 = Button(text = "3",command = lambda : depl(img3,position,bouton),image = case4)
img4 = Button(text = "4",command = lambda : depl(img4,position,bouton),image = case5)
```

```
img5 = Button(text = "5",command = lambda : depl(img5,position,bouton),image = case6)

img6 = Button(text = "6",command = lambda : depl(img6,position,bouton),image = case7)

img7 = Button(text = "7",command = lambda : depl(img7,position,bouton),image = case8)

img8 = Button(text = "8",command = lambda : depl(img8,position,bouton),image = case9)

img9 = Button(text = "9",command = lambda : depl(img9,position,bouton),image = case10)

img10 = Button(text = "10",command = lambda : depl(img10,position,bouton),image =
case11)

img11 = Button(text = "11",command = lambda : depl(img11,position,bouton),image =
case12)

img12 = Button(text = "12",command = lambda : depl(img12,position,bouton),image =
case13)

img13 = Button(text = "13",command = lambda : depl(img13,position,bouton),image =
case14)

img14 = Button(text = "14",command = lambda : depl(img14,position,bouton),image =
case15)

img15 = Label(fenetre,image = taquin)

#on crée des boutons contenant les images et qui lancent la fonction depl sur eux même
( grâce à lambda ) lorsque l'on clique dessus

#img15 représente la case vide et aussi l'image complète que l'on affiche si le jeu est fini


bouton =
[img0,img1,img2,img3,img4,img5,img6,img7,img8,img9,img10,img11,img12,img13,img14,i
mg15]

#Ce tableau contient le nom des boutons dans l'ordre

position = [(0.00,0.00), (200.00,0.00), (400.00,0.00), (600.00,0.00), (0.00,200.00),
(200.00,200.00),(400.00,200.00),(600.00,200.00),(0.00,400.00),(200.00,400.00),
(400.00,400.00),(600.00,400.00),(0.00,600.00),(200.00,600.00),(400.00,600.00),
(600.00,600.00)]
```

```
#Ce tableau contient les positions des boutons, sont index correspond a celui du tableau bouton


#on mélange la liste et on affiche les cases
for i,j in enumerate(tab_end):
    if i <15:
        bouton[i].place(x=j[0],y=j[1])
        position[i] = j
        #on fait attention de bien garder la correspondance d'indexage entre position et bouton
    else :
        position[i] = j
        #on ne veut pas afficher img15 qui ne doit l'être que quand le Taquin est fini

init(position,bouton)
#on mélange le Taquin quand tout est pret

fenetre.bind("<ButtonRelease-1>", detec)
#lorsque l'on relache le clic droit, on lance détection.

fenetre.mainloop()
#boucle principale
```

VII.C. Allumettes.py

```
#! /usr/bin/python
# -*- coding: utf-8 -*-
##########Allumettes###########

from turtle import*
from Tkinter import*



#Fonction qui permet d'authoriser l'effaçage d'une allumette en fonction de plusieurs choses, tout d'abord bien sur si le joueur clique sur une allumette, et ensuite si
#il n'a pas déjà effacé pendant son tour une allumette d'un autre étage.


def efface(x,y):
    a = [x,y]
    global ligne_select
    global etat
    for indice,valeur in enumerate(liste_alumette):
        z = valeur.pos()
        if (abs(a[0]-z[0]) < 10 and abs(a[1]-z[1 ]) < 25) and (ligne_select.get() == -1 or z[1] == ligne_select.get()):
            valeur.ht()
            etat = etat - 1
            gagner()
```

```
        if ligne_select.get() == -1:
            ligne_select.set(z[1])

#La variable état représente le nombre d'allumettes restantes en jeux, quand elle est a 0, l'avant dernier joueur est le gagnant.

def gagner():
    if etat == 0:
        fenetre_toto2=Toplevel(fenetre_toto)
        annonce = Label(fenetre_toto2,text = "Félicitations Joueur %d, vous avez gagné !!!" % etat_joueur.get())
        annonce.pack()
        ok_btn = Button(fenetre_toto2,text = "OK",command=fenetre_toto2.destroy)
        ok_btn.pack()

        posx = fenetre_toto2.winfo_screenwidth()
        posy = fenetre_toto2.winfo_screenheight()
        diffx=300
        diffy=50
        x = (posx/2) - (diffx/2)
        y = (posy/2) - (diffy/2)
        fenetre_toto2.geometry('%dx%d+%d+%d'%(diffx,diffy,x,y))

#Fonction qui ré-initialise les variable afin de pouvoir faire une nouvelle partie
def new_part():
```

```
    global etat,etat_joueur,ligne_select,liste_allumette
    etat = 16
    etat_joueur.set(2)
    ligne_select.set(-1)
    for i in liste_alumette:
        i.st()

#Création d'une fenêtre TK pour pouvoir y placer les radioboutton, ainsi que le bouton recommencer

fenetre_toto = Tk()
fenetre_toto.title("Allumettes")


posx = fenetre_toto.winfo_screenwidth()
posy = fenetre_toto.winfo_screenheight()
#Dimension de la fenêtre 800x600
diffx=800
diffy=600
x = (posx/2) - (diffx/2)
y = (posy/2) - (diffy/2)
fenetre_toto.geometry('%dx%d+%d+%d'%(diffx,diffy,x,y))

canvas1 = Canvas(fenetre_toto,width = 800,height = 600)
canvas1.pack()
```

```
screen1 = TurtleScreen(canvas1)

etat_joueur = IntVar()

ligne_select = IntVar()


#Création des 16 curseurs qui auront l'image d'une allumette


a1 = RawTurtle(screen1)

a2 = RawTurtle(screen1)

a3 = RawTurtle(screen1)

a4 = RawTurtle(screen1)

a5 = RawTurtle(screen1)

a6 = RawTurtle(screen1)

a7 = RawTurtle(screen1)

a8 = RawTurtle(screen1)

a9 = RawTurtle(screen1)

a10 = RawTurtle(screen1)

a11 = RawTurtle(screen1)

a12 = RawTurtle(screen1)

a13 = RawTurtle(screen1)

a14 = RawTurtle(screen1)

a15 = RawTurtle(screen1)

a16 = RawTurtle(screen1)


#On les place dans une liste afin de faciliter leur gestions (leur assigner l'image de
l'allumette, les placer au bon endroit etc
```

```
liste_alumette = [a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,a15,a16]

screen1.register_shape("Allumettes/allumettes.gif")
screen1.bgpic("Allumettes/alu2.gif")

for i in liste_alumette :
    i.shape("Allumettes/allumettes.gif")

#Cette liste contient les positions de chaque allumette, afin qu'elles représente un triangle
triangle_toto = [[0,210],[-30,140],[0,140],[30,140],[-60,70],[-30,70],[0,70],[30,70],[60,70],[-90,0],[-60,0],[-30,0],[0,0],[30,0],[60,0],[90,0]]

#On place les allumettes correctement
for indice,valeur in enumerate(triangle_toto):
    liste_alumette[indice].up()
    liste_alumette[indice].goto(valeur)
    liste_alumette[indice].down()


new_part()

for i in liste_alumette:
    i.onclick(efface,btn = 1)

#Création des 2 radiobuttons pour changer de joueur et du bouton recommencer
```

```
Joueur_1 = Radiobutton(text = "Joueur 1",variable = etat_joueur,value = 2,command =
lambda : [ligne_select.set(-1)]);

Joueur_2 = Radiobutton(text = "Joueur 2",variable = etat_joueur,value = 1,command =
lambda : [ligne_select.set(-1)]);

reco = Button(text = "Nouvelle Partie",command = new_part)


#On les place correctement

Joueur_1.place(x=150,y=100)

Joueur_2.place(x=150,y=150)

reco.place(x=150,y=200)



fenetre_toto.mainloop()
```

VIII/ Bibliographie

VIII.A. Contexte

Ce projet a été réalisé dans le cadre de l'UE de programmation impérative de la faculté Jean-François Champollion d'Albi à l'aide de python 2.6 sous Ubuntu 10.10. Les images ont entièrement été réalisée par nos soins sous Photoshop CS5 et Gimp.

VIII.B. Liens Internet et Livres

Voici la liste des sites internet utilisé pour la réalisation de se projet :

http://www.tkdocs.com/tutorial/canvas.html
http://doc.smallbasic.com/default.aspx?id=14&language=fr
http://www.velocityreviews.com/forums/t737888-drawing-with-the-mouse-with-turtle.html
http://doc.smallbasic.com/default.aspx?id=14&language=fr
http://effbot.org/tkinterbook/tkinter-index.htm
http://sebsauvage.net/python/gui/index_fr.html#change_label
http://www.tclscripting.com/articles/jun06/article2.html
http://wiki.python.org/moin/Intro_to_programming_with_Python_and_Tkinter?highlight=%28tkinter%29
http://pyrorobotics.org/usermanual/pyrobot.gui.TK.TKgui-class.html
http://www.jchr.be/python/index.htm
http://www.tcl.tk/man/tcl8.4/TkCmd/colors.htm
http://gnuprog.info/prog/python/tkinter.php
http://www.developpez.net/forums/archive/index.php/f-169.html
http://209.85.135.104/search?q=cache:cQ4_6q5FAB4J:www.nmt.edu/tcc/help/lang/python/tkinter.pdf+tkinter+multi+toplevel&hl=fr&ct=clnk&cd=63&gl=fr#62
http://artyprog.freezope.org/Python/tkinter/index.htm#introduction
http://ferry.eof.eu.org/lesjournaux/pg/public_html/x12080.html
http://fr.wikibooks.org/wiki/Apprendre_%C3%A0_programmer_avec_Python/Et_pour_quelques_widgets_de_plus_...
http://www.java2s.com/Open-Source/Python/3.1.2-Python/Demo/Demo/turtle/turtledemo_two_canvases.py.htm
http://www.pythonware.com/library/tkinter/introduction/index.htm
http://gnuprog.info/prog/python/pwidget.php
http://www.siteduzero.com/tutoriel-3-223267-apprendre-python.html
http://docs.python.org/library/turtle.html

Eyrolles : Programmation Python Édition 2