MULLER
Romain

Louis Marchal
LYCÉE D'ENSEIGNEMENT GÉNÉRAL - TECHNOLOGIQUE - PROFESSIONNEL - INDUSTRIEL

Projet ISN : Paintux

SOMMAIRE :

Introduction

I. Genèse du projet

II. Explication de Paintux

1/ Fonctionnement générale
2/ Fonctionnement d'une partie du code

III. Organisation et réalisation

1/ Répartition des tâches
2/ Problème et résolution
3/ Idées d'améliorations

Conclusion

Annexe : le code du projet

Introduction

Durant cette année de Terminale Scientifique en Informatique en Science du Numérique (ISN), nous avons dû développer un projet en utilisant le langage Python. Le but est d'avoir quelque chose de fonctionnel que l'on puisse présenter le jour du baccalauréat. Notre projet est un logiciel de dessin, Paintux.

I. Genèse du projet

Le projet a commencé en décembre. On devait former des binômes, j'ai donc choisi de me mettre avec Yann car on avait un niveau équivalent en programmation. Il avait à ce moment déjà commencé le projet. L'idée m'a paru intéressante car cela sortait des projets habituelles de jeux vidéos et l'idée de faire une interface graphique avec GTK m'a plu. Ce qui était aussi intéressant, c'était de travailler en groupe, ce qui était une nouveauté pour moi puisque je n'avais jamais travaillé en groupe dans le cadre d'un projet informatique.

Paintux est un logiciel de dessin basique. Le nom « Paintux » fait référence à la mascotte de GNU/Linux, Tux, car le projet a été développé sur GNU/Linux, mais reste compatible Windows grâce à la grande compatibilité de Python. Nous utilisons pour la partie graphique PyGame qui est une interface en python de la bibliothèque SDL (Simple DirectMedia Layer) écrit en C qui permet de gérer la partie graphique d'un programme. Pour ce qui est de l'interface Homme-Machine nous utilisons PyGTK qui est aussi une interface Python de la bibliothèque GTK souvent utilisée dans les environnements GNU/Linux pour faire des interfaces Homme-Machine. Vu que PyGTK n'est pas compatible avec la version 3.x de Python nous avons utilisé la version 2.7 de Python.

II. Explication de Paintux

1/ Fonctionnement générale

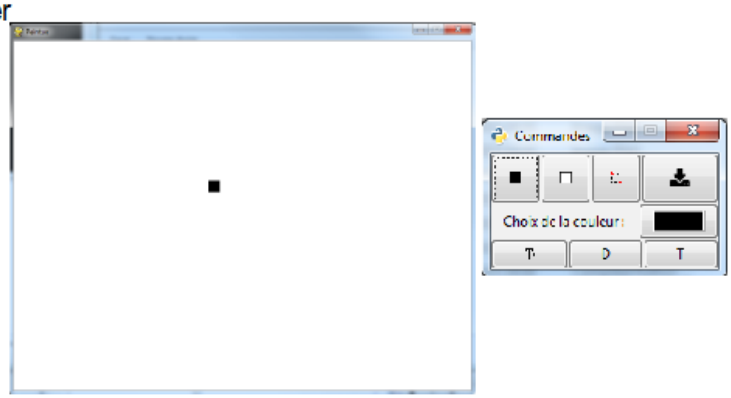
Notre projet est divisé en deux modules : « paint » et « interfacee ». Le module « paint » utilise PyGame et sert à gérer la partie graphique c'est-à-dire la fenêtre de dessein. C'est dans cette fenêtre que l'on va pouvoir dessiner, gommer et modifier les images importées, en somme c'est dans cette partie que l'on utilise les outils sélectionnés. Le module « interfacee » utilise PyGTK pour l'interface Homme-Machine et permet de sélectionner les outils que l'on souhaite utiliser comme la gomme, l'importation d'une image, le carré de dessin, changer la couleur, sauvegarder l'image, changer la taille du carré et les dimensions de la fenêtre.

Les deux parties sont indépendantes et peuvent être lancées sans l'autre. Il faut alors trouver un moyen de faire communiquer ces deux programmes, sachant que ni PyGTK, ni PyGame propose d'outil pour cela. Nous avons alors décidé de tirer profit de la bibliothèque standard de Python qui propose le module *pickle* qui permet d'enregistrer des objets Python dans des fichiers. Ainsi l'interface écrit les outils sélectionnés dans le fichier *tools*, la couleur dans le fichier *color*, les dimensions dans le fichier *dimension*, la taille du carré est écrit dans *taille* et l'image choisie pour l'import est dans le fichier *image_choisie*. De son coté « paint » va lire pour chaque tour de boucle les différents fichiers et réagit en conséquence.

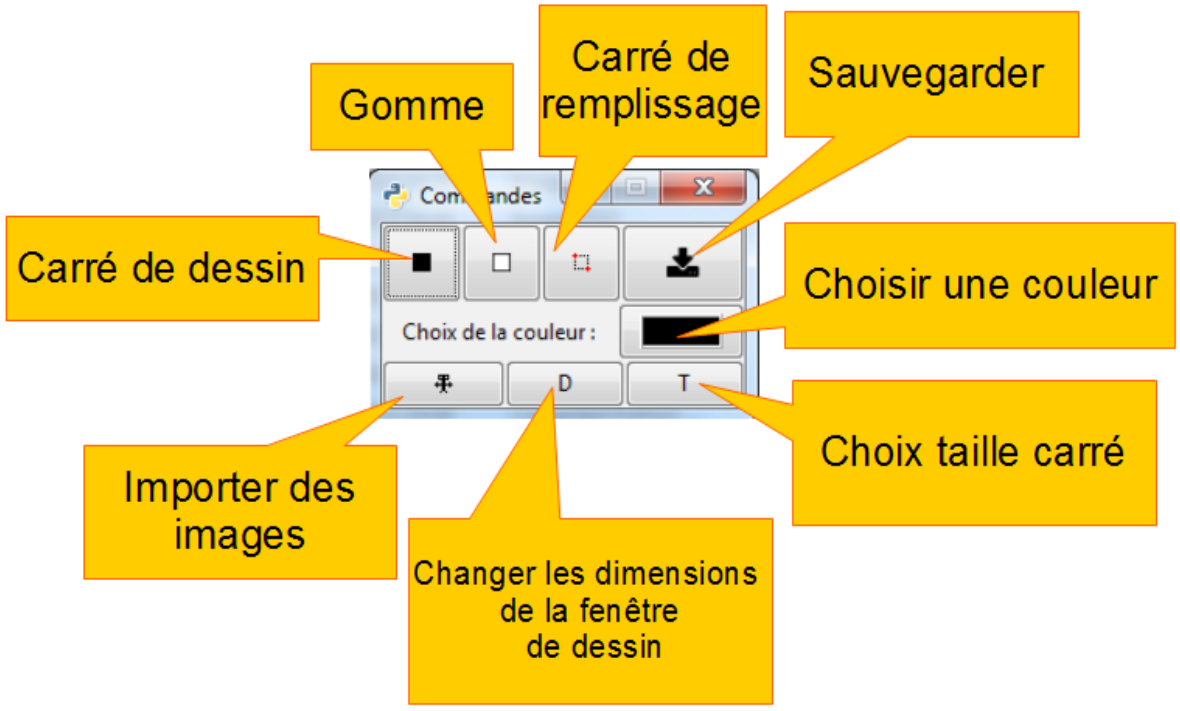
Fonctionnement de Paintux

Lancement du programme :

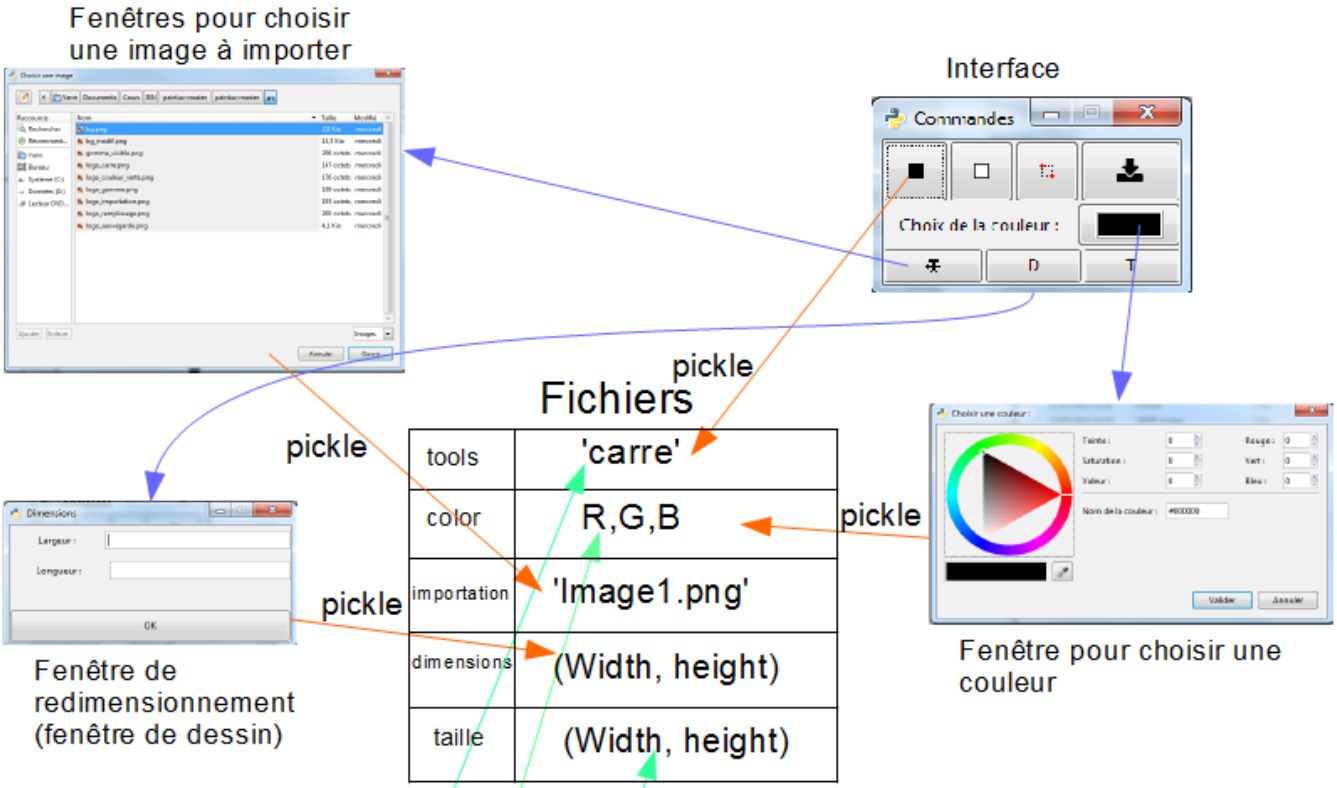
Le programme se lance avec le fichier paint.py, la fenêtre de dessin et l'interface s'ouvrent alors.



Interface de commande :



Communication entre l'interface et la fenêtre de dessin :



Fonctionnement simplifié de la boucle principale de paint.py :
(Exemple quand l'utilisateur utilise le carré de dessin)

Liste des modules importés dans paint.py (chaque module utilisé correspond à une couleur):

- sys
- subprocess
- pygame
- objet
- fonctions

Boucle de paint.py (extrait)

```
Début boucle :
    choix = LectureFichier()
    Si choix == "carre" :
        mouse_x, mouse_y = PositionMouse(carre1)#retourne la position du carré de tel sorte que la souris
        soit au milieu du carré
        #couleur
        couleur = LectureCouleur()
        carre1.ChangementCouleur(couleur)
        #taille et position du carré
        carre_w,carre_h = LectureTaille()
        carre1.ChangementTaille(carre_w ,carre_h)
        carre1.Position(mouse_x,mouse_y)
```

2/ Fonctionnement d'une partie du code

Nous allons plus particulièrement nous intéresser à la fonction de retour en arrière. On utilise une liste qui enregistre l'image à chaque fois qu'on la modifie ainsi que d'une variable *MAX_BACK* qui définit la taille de la liste et ainsi le nombre de retour en arrière possible afin d'éviter de surcharger la mémoire.

```
MAX_BACK = 30
count_back = 0
retour_arriere = [bg]
retour_arriere_tmp = []
```

A chaque fois que l'on modifie l'image, on l'ajoute à la fin de la liste, on vérifie ensuite que la taille est pas trop importante. Le cas échéant on enlève au début de la liste les images en trop, soit les éléments les plus anciens.

```
retour_arriere.append(bg_modif)
if len(retour_arriere) > MAX_BACK:
    retour_arriere = retour_arriere[1:]
count_back = len(retour_arriere) - 2
```

Si on utilise le retour en arrière on utilise la variable *count_back* qui sert à compter le nombre de fois que l'on veut revenir en arrière et sert donc index pour la liste. On récupère alors l'image de la liste que l'on définit alors comme image de PyGame, et on rajoute la modification dans une liste temporaire afin que le retour en arrière soit fluide.

```
bg_modif = retour_arriere[count_back] # On reprend l'image sauvegardée
retour_arriere_tmp.append(bg_modif)
pygame.image.save(bg_modif, "bg_modif.png")
# La dernière image ajoutée a le plus grand index , du coup on commence au max
pour aller à zéro
count_back -= 1
```

Ensuite on vérifie si cette fonctionnalité a été utilisée et on rajoute les éléments de la liste temporaire à la liste principale.

```
if count_back != (len(retour_arriere) - 2) or count_back != len(retour_arriere):
    retour_arriere += retour_arriere_tmp[::-1] # On récupère le dernier élément
     # On vérifie qu'il n'y a pas d'éléments en trop
     difference = len(retour_arriere) - MAX_BACK
     if difference > 0:
        retour_arriere = retour_arriere[difference:]
     retour_arriere_tmp = []
```

III. Organisation et réalisation

1/ Répartition des tâches

Pour la répartition des tâches et l'organisation du travail cela s'est fait assez naturellement. Chacun prenait les fonctionnalités qui l'intéressait. Ainsi pendant les cours de ISN on décidait des améliorations à faire et des fonctionnalités à rajouter. On les développait ensuite chez nous pendant la semaine ou sur plusieurs semaines en fonctions de ce qu il y avait à implémenter. Par exemple Yann a ajouté la possibilité d'importer une image. Alors que j'ai ajouté le retour en arrière afin d'effacer une erreur faite pendant que l'on dessine.

Yann étant déjà sur le projet quand je suis arrivé il a écrit le noyau principal, c'est-à-dire la boucle d'affichage et le début de l'interface. Ainsi c'est lui qui a développé les principaux outils comme le carré de dessin, la gomme ou le choix de la couleur. Pour ma part j'ai ajouté le retour en arrière ainsi que la possibilité de sauvegarder l'image. J'ai aussi aidé à optimiser le noyau et ajouter des améliorations générales comme l'ouverture et la fermeture de l'interface en même temps que la fenêtre de dessin qui est la partie principale du programme.

2/ Problème et résolution

On va revenir sur la fonctionnalité de retour en arrière qui m'a posé plusieurs problèmes. Tout d'abord on a remarqué qu'il fallait appuyer deux fois sur « z » au lieu d'une pour faire le premier retour en arrière. Cela est dû au fait que l'on rajoute chaque modification à la fin de la liste, ainsi quand on faisait un retour en arrière on prenait le dernier élément de la liste qui était donc identique à ce qu'on avait déjà l'écran. On a donc décalé l'index pour la première utilisation de la fonctionnalité.

Ensuite quand on utilise cette fonctionnalité, pour que l'ordre soit respecté, il fallait ajouter l'image à la liste, or si nous faisions cela on tournait en boucle. Donc on utilise une deuxième liste temporaire qui est additionnée à la première quand on n'utilise plus la fonction de retour en arrière.

3/ Idées d'améliorations

Il existe de nombreux points sur lequel notre projet peut s'améliorer. Tout d'abord l'organisation du code ne respecte aucune architecture connue et s'en trouverait fortement améliorée si on utilisait un patron de design (*design pattern* en anglais) comme le patron MVC (pour Modèle, Vue, Contrôleur). Ensuite la communication entre les deux programmes « paint » et « interfacee » consomme beaucoup d'accès au disque dur. Une communication sur le modèle serveur-client comme pour internet pourrait être une solution.

Et de manière générale l'ajout de fonctionnalités supplémentaires comme pouvoir dessiner des formes plus complexes comme des polygones, pouvoir utiliser un cercle comme crayon au lieu du carré, pouvoir flouter, etc.

Conclusion

Ainsi tout au long de ce projet j'ai pu enrichir mes connaissances dans divers domaines comme le travail en groupe qui était une nouveauté pour moi. J'ai ainsi du apprendre la manière de gérer un projet à plusieurs, savoir se répartir les tâches, comment gérer les différents versions du projet qui s'accumulent vite et dans lesquelles on s'y perd rapidement.

J'ai aussi appris à utiliser les bibliothèques PyGTK et PyGame et donc plus généralement appris à créer une interface Homme-Machine ainsi que de faire du graphisme. L'utilisation de ces bibliothèques m'a permis d'avoir une autre approche des problèmes car les problèmes sont différents de ce que l'on peut rencontrer couramment.

Annexe : le code du projet

Le code du projet est aussi disponible sur github (téléchargement ici).

paint.py

```
#coding: utf-8
import pygame
import sys
import subprocess
import atexit

from pygame.locals import *
from objet import *
from fonctions import *

# initialisation des parametres de base de pygame
pygame.init()
size = width, height = 800, 600
screen = pygame.display.set_mode(size)
clock = pygame.time.Clock()
window_width, window_height = size
pygame.display.set_caption("Paintux")

# couleurs de base utilisees dans la suite du programme
white = 255, 255, 255
black = 0, 0, 0

# partie gerant la sourie
pygame.mouse.set_visible(1)
mouse_x = 0
mouse_y = 0

# creation d'une instance de la classe Carre
carre1 = Carre()
carres = pygame.sprite.Group()
carres.add(carre1)

# creation d'une instance de la classe Gomme
gomme1 = Gomme()
gommes = pygame.sprite.Group()
gommes.add(gomme1)


# creation d'une instance de la classe Remplissage
remplissage1 = Remplissage()
remplissages = pygame.sprite.Group()
remplissages.add(remplissage1)
nb_point = 0
liste_point = [] #liste qui va stocker la liste des points choisis par l'utilisateur
ok = 0 #variable qui verifie la validite des points 'cliques'

# creation d'une instance de la classe Imagess
importation1 = Imagess()
```

```
importations = pygame.sprite.Group()
importations.add(importation1)
changement_image = 1 # variable qui permet de savoir si l'utilisateur a change d'image ou
non

# background de depart
bg = pygame.image.load("bg.png")
bg_depart = 1 # variable qui va servir a savoir si le background a ete modifie ou non

# repetition des touches
pygame.key.set_repeat(1, 1)

# ecriture du choix de depart qui sera par default toujours le meme
ecriture = "carre"
EcritureFichier(ecriture) # ecriture de carre dans le fichier 'tools'

# ecriture de la couleur de depart pour le carre de dessin
with open("color", "wb") as file:
        data = pickle.Pickler(file)
        data.dump(black)
# ecriture de la taille de depart du carre de dessin
EcritureTaille(20,20)

# lance un nouveau processus : l'interface PyGTK
interfaceProcess = subprocess.Popen(['python', 'interfacee.py'])

# Ce lance automatiquement à la fermeture de la VM
@atexit.register
def goodbye(): # Ferme le processus
    interfaceProcess.kill()

# nombre de retour en arriere possible, pour eviter de surcharger la mémoire
MAX_BACK = 30
count_back = 0
retour_arriere = [bg]
retour_arriere_tmp = []

# redimension de la fenetre
non_modif_zone = 1; #variable qui va permettre de savoir si la fenetre de dessin a ete
modifiee ou non
# pour adapter la redimension en consequence

while 1:
    clock.tick(2000)
    choix = LectureFichier() # lecture de l'outil choisi par l'utilisateur

    # si la background n'a pas ete modifie on colle le background de depart
    # sinon on colle le background modifie
    if bg_depart == 1:
        screen.blit(bg, (0,0))
    elif bg_depart == 2:
        screen.blit(bg_modif, (0,0))

    if choix == "carre":
        # position du carre
        mouse_x, mouse_y = PositionMouse(carre1)
        # couleur
        couleur = LectureCouleur() # va lire la couleur choisie par l'utilisateur
        carre1.ChangementCouleur(couleur)
        carre_w,carre_h = LectureTaille()
```

```
            carre1.ChangementTaille(carre_w,carre_h)
            carre1.Position(mouse_x, mouse_y) # pour eviter le retour a zero de la position en cas de redimension
        elif choix == "gomme":
            # position de la gomme
            mouse_x, mouse_y = PositionMouse(gomme1)
        elif choix == "remplissage":
            # position de la souris
            mouse_x, mouse_y = pygame.mouse.get_pos()
            # couleur
            couleur = LectureCouleur()
        elif choix == "importation":
            if changement_image == 1:
                importation1.ChangementImage(window_height,window_width) # va aller lire l'image choisie par l'utilisateur
                # dans le fichier image et adapter les dimensions de l'image pour quelle ne depasse pas les dimmensions de la fenetre
                changement_image = 0
            #position de la souris
            mouse_x, mouse_y = PositionMouse(importation1)
        elif choix == "redimension":
            window_width, window_height = LectureDimensions() # va aller lire les dimensions de la fenetre entrees par l'utilisateur
            size,surface = RedimensionEcran(window_width,window_height) # effectue des verifications sur ces dimemsions
            if surface == 0:
                screen = pygame.display.set_mode(size)
            else:
                if non_modif_zone:
                    pygame.image.save(screen, "bg_modif.png")
                    bg_modif = pygame.image.load("bg_modif.png")
                    non_modif_zone = 0
                    bg_depart = 2
                screen = pygame.display.set_mode(size) # on redimensionne la fenetre
                screen.blit(surface, (0,0))
                screen.blit(bg_modif, (0,0))
                pygame.image.save(screen, "bg_modif.png")
                bg_modif = pygame.image.load("bg_modif.png")
            EcritureFichier("carre")

        # lecture des differents evenements
        for event in pygame.event.get():
            if event.type == QUIT:
                sys.exit()
            elif event.type == KEYDOWN:
                if event.key == K_ESCAPE:
                    sys.exit()
            elif event.type == KEYUP:
                if event.key == K_z:
                    if count_back >= 0:
                        bg_depart = 2
                        bg_modif = retour_arriere[count_back] # On reprend l'image sauvegardé
                        retour_arriere_tmp.append(bg_modif)
                        pygame.image.save(bg_modif, "bg_modif.png")
                        # La derinère image ajoutée a le plus grand index , du coup on commence au max pour aller à zéro
                        count_back -= 1
                    elif event.key == K_UP:
                        if choix == "importation":
                            importation1.RedimensionnementImage(1) # on agrandit l'image
```

```
            elif event.key == K_DOWN:
                if choix == "importation":
                    importation1.RedimensionnementImage(-1) # on diminue la taille de l'image
            elif event.key == K_RIGHT:
                if choix == "importation":
                    importation1.RotationImage(-1) # on fait tourner l'image vers la droite
            elif event.key == K_LEFT:
                if choix == "importation":
                    importation1.RotationImage(1) # on fait tourner l'image vers la gauche
        elif event.type == MOUSEBUTTONDOWN:
            if event.button == 1:
                if choix == "remplissage":
                    if nb_point < 2: # tant que le nombre de points est inferieur a 2 on ne dessine pas
la carre de remplissage
                        point_x,point_y = pygame.mouse.get_pos()
                        liste_point.append((point_x,point_y))
                        if nb_point == 1:
                            nb_point = 1
                            ok = 1
                        else:
                            nb_point += 1
                    if ok: # si on a deux points, on stocke les coordonnees de ces deux points
                        x_remplissage,y_remplissage = liste_point[0]
                        x_remplissage_d,y_remplissage_d = liste_point[1]

remplissage1.PointduCarre(x_remplissage,y_remplissage,x_remplissage_d,y_remplissage_d,co
uleur) # va creer un surface
                        # avec les coordonnees donnees en parametre
                        remplissage1.Position(x_remplissage,y_remplissage) # va mettre cette nouvelle
surface a la position du premier point choisi
                        remplissages.draw(screen) # puis va afficher la carre de remplissage a l'ecran
                        pygame.display.update()
                        pygame.image.save(screen, "bg_modif.png")
                        bg_modif = pygame.image.load("bg_modif.png")
                        bg_depart = 2
                        remplissage1.Resset()
                        nb_point = 0
                        liste_point = []
                        ok = 0
                elif choix == "carre":
                    bg_depart = 0 # empeche que le fond se recole et permet donc de declancher le
'dessin'
                elif choix == "gomme":
                    bg_depart = 0  # empeche que le fond se recole et permet donc de declancher le
'dessin'
                    gommes.empty()
                    gomme1.Gommer(1)
                    gommes.add(gomme1)
        elif event.type == MOUSEBUTTONUP:
            if event.button == 1:
                if choix == "carre":
                    # on affiche le carre a l'ecran puis on sauvegarde le nouveau background
                    # on affiche une deuxieme fois le carre car on a colle le fond au debut de la boucle
                    # si on ne reaffiche pas le carre il ne sera pas visible
                    carres.draw(screen)
                    pygame.image.save(screen, "bg_modif.png")
                    bg_modif = pygame.image.load("bg_modif.png")
                    bg_depart = 2
                elif choix == "gomme":
                    gommes.draw(screen)
```

```
                pygame.image.save(screen, "bg_modif.png")
                bg_modif = pygame.image.load("bg_modif.png")
                bg_depart = 2
                gommes.empty()
                gomme1.Gommer(0)
                gommes.add(gomme1)
            elif choix == "importation":
                importation1.Position(mouse_x, mouse_y)
                importations.draw(screen)
                pygame.image.save(screen, "bg_modif.png")
                bg_modif = pygame.image.load("bg_modif.png")
                bg_depart = 2
                changement_image = 1 # car on vient de coller l'image

            # Si on a utiliser le retour en arrière
            if count_back != (len(retour_arriere) - 2) or count_back != len(retour_arriere):
                retour_arriere += retour_arriere_tmp[::-1] # On récupère le dernier élément
                # On vérifie qu'il n'y a pas d'éléments en trop
                difference = len(retour_arriere) - MAX_BACK
                if difference > 0:
                    retour_arriere = retour_arriere[difference:]
                retour_arriere_tmp = []


            retour_arriere.append(bg_modif)
            if len(retour_arriere) > MAX_BACK:
                retour_arriere = retour_arriere[1:]

            count_back = len(retour_arriere) - 2

    # on affiche l'outil choisi
    if choix == "carre":
        carre1.Position(mouse_x, mouse_y)
        carres.draw(screen)
    elif choix == "gomme":
        gomme1.Position(mouse_x, mouse_y)
        gommes.draw(screen)
    elif choix == "importation":
        importation1.Position(mouse_x, mouse_y)
        importations.draw(screen)

    pygame.display.update()
```

interfacee.py

```
# coding: utf-8
import pygtk
import gtk
import pickle
import shutil
import re
from fonctions import *



class Window():
    """Classe gerant l'affichage du panneau du controle du paint"""

    def __init__(self):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.set_position(gtk.WIN_POS_CENTER)
        self.window.set_size_request(200,100)
        self.window.set_resizable(False)
        self.window.set_title("Commandes")

        self.Buttons = gtk.VBox()
        self.ligne_1 = gtk.HBox()
        self.ligne_2 = gtk.HBox()
        self.ligne_3 = gtk.HBox()

        #bouton carre
        image_carre = gtk.Image()
        image_carre.set_from_file("logo_carre.png")
        self.Button_carre = gtk.Button()
        self.Button_carre.add(image_carre)
        # La methode "connect" sert a associer un evenement (ici "clicked") a
une fonction ou une methode (ici self.sauvegarde)
        self.Button_carre.connect("clicked",lambda
widget,button="carre":self.sauvegarde(widget,button))
        self.Button_carre.set_tooltip_text("Carre de dessin")

        #bouton gomme
        image_gomme = gtk.Image()
        image_gomme.set_from_file("logo_gomme.png")
        self.Button_gomme = gtk.Button()
        self.Button_gomme.add(image_gomme)
        self.Button_gomme.connect("clicked", lambda widget,
button="gomme":self.sauvegarde(widget,button))
        self.Button_gomme.set_tooltip_text("Gommer")

        #bouton remplissage
        image_remplissage = gtk.Image()
        image_remplissage.set_from_file("logo_remplissage.png")
        self.Button_remplissage = gtk.Button()
        self.Button_remplissage.add(image_remplissage)
        self.Button_remplissage.connect("clicked", lambda widget,
button="remplissage":self.sauvegarde(widget,button))
        self.Button_remplissage.set_tooltip_text("Remplir avec un carre")


        #bouton Sauvegarde
        image_sauvegarde = gtk.Image()
        image_sauvegarde.set_from_file("logo_sauvegarde.png")
```

```
        self.Button_sauvegarde = gtk.Button()
        self.Button_sauvegarde.add(image_sauvegarde)
        self.Button_sauvegarde.connect("clicked", self.sauvegarde_image)
        self.Button_sauvegarde.set_tooltip_text("Sauvegarder l'image")


        #bouton importation image
        image_importation = gtk.Image()
        image_importation.set_from_file("logo_importation.png")
        self.Button_importation = gtk.Button()
        self.Button_importation.add(image_importation)
        self.Button_importation.connect("clicked", self.importation_image)
        self.Button_importation.set_tooltip_text("Importer une image")

        #bouton redimensionement
        self.Button_redimension = gtk.Button("D")
        self.Button_redimension.connect("clicked", self.redimension)
        self.Button_redimension.set_tooltip_text("Redimensionner l'image")

        #bouton taille
        self.Button_taille = gtk.Button("T")
        self.Button_taille.connect("clicked", self.taille)
        self.Button_taille.set_tooltip_text("Taille du carre de dessin")

        #choix couleur
        #label couleur
        self.Label_color = gtk.Label('Choix de la couleur :')


        #boutton couleur
        self.Button_color = gtk.ColorButton(gtk.gdk.color_parse('black'))
        self.Button_color.set_title('Choisir une couleur :')
        self.Button_color.connect('color-set', self.color_set)
        self.Button_color.set_tooltip_text("Choisir une couleur")

        self.ligne_1.pack_start(self.Button_carre)
        self.ligne_1.pack_start(self.Button_gomme)
        self.ligne_1.pack_start(self.Button_remplissage)
        self.ligne_1.pack_start(self.Button_sauvegarde)

        self.ligne_2.pack_start(self.Label_color)
        self.ligne_2.pack_start(self.Button_color)

        self.ligne_3.pack_start(self.Button_importation)
        self.ligne_3.pack_start(self.Button_redimension)
        self.ligne_3.pack_start(self.Button_taille)

        self.Buttons.pack_start(self.ligne_1)
        self.Buttons.pack_start(self.ligne_2)
        self.Buttons.pack_start(self.ligne_3)

        self.window.add(self.Buttons)
        self.window.connect("destroy", self.destroy)
        self.window.show_all()

    def sauvegarde(self, widget, button):
        """
        Sauvegarde quel outil (gomme, crayon, etc) est utilise
        """
        EcritureFichier(button)
```

```
    def color_set(self,widget):
        """
        Recupere la couleur depuis le widget et l'ecrit dans le fichier 'color'
        """
        color = self.Button_color.get_color() # recuperation de la couleur mais
pas dans le bon format
        # converion en RGB
        couleur = color.red // 256, color.green // 256, color.blue // 256
        EcritureCouleur(couleur)


    def modiftaille(self):
        """
        Simule un clic sur le bouton qui active le dessin
        """
        self.Button_carre.clicked()

    def taille(self, widget):
        """
        Ouverture de la fenetre permettant de choisir la taille du carre de
dessin
        """
        t = Taille()
        t.main()

    def redimension(self, widget):
        """
        Ouverture de la fenetre permettant de choisir la taille de la fentre de
dessin
        """
        redim = Redi()
        redim.main()

    def importation_image(self,widget):
        """
        Gere l'importation d'une image
        format supporter : (jpg, png)
        """
        importation_dialog = gtk.FileChooserDialog("Choisir une image",
            None, gtk.FILE_CHOOSER_ACTION_OPEN,
            (gtk.STOCK_CANCEL, gtk.RESPONSE_CANCEL, gtk.STOCK_OPEN,
gtk.RESPONSE_OK))

        importation_dialog.set_default_response(gtk.RESPONSE_CANCEL)

        filter = gtk.FileFilter()
        filter.set_name("Images")
        filter.add_mime_type("image/png")
        filter.add_mime_type("image/jpeg")
        filter.add_pattern("*.png")
        filter.add_pattern("*.jpg")
        filter.add_pattern("*.jpeg")

        importation_dialog.add_filter(filter)

        response = importation_dialog.run()

        if response == gtk.RESPONSE_OK:
            fichier_choisi = importation_dialog.get_filename()
            # ecriture du chemin de l'image choisie
            with open("image_choisie", "wb") as file:
```

```
                data = pickle.Pickler(file)
                data.dump(fichier_choisi)
            # changement de l'outil dans le paint
            EcritureFichier("importation")
        elif response == gtk.RESPONSE_CANCEL:
            importation_dialog.destroy()

        importation_dialog.destroy()


    def sauvegarde_image(self, widget):
        """
        Gere la sauvegarde de l'image
        format d'export : png
        """
        filechooserdialog = gtk.FileChooserDialog(
            "Sauvegarder l'Image",
            None,
            gtk.FILE_CHOOSER_ACTION_SAVE,
            (gtk.STOCK_CANCEL, gtk.RESPONSE_CANCEL, gtk.STOCK_OK,
gtk.RESPONSE_OK),
            None
        )
        filter = gtk.FileFilter()
        filter.set_name("Images (*.png)")
        filter.add_mime_type("image/png")
        filter.add_pattern("*.png")
        filechooserdialog.add_filter(filter)

        response = filechooserdialog.run()
        if response == gtk.RESPONSE_OK:
            filename = filechooserdialog.get_filename()
            if re.search('\.png$', filename) == None:
                filename += '.png'
                print 'add .png'
            print filename, 'selected'
            shutil.copy('bg_modif.png', filename)
        elif response == gtk.RESPONSE_CANCEL:
            print 'Closed, no files selected'
        filechooserdialog.destroy()


    def main(self):
        gtk.main()

    def destroy(self,data=None):
        gtk.main_quit()




####################
# Classe taille
####################




class Taille():
```

```
    """Classe permettant de gerer la fenetre qui permet de choisir la taille du
carre de dessin"""
    def __init__(self):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.set_position(gtk.WIN_POS_CENTER)
        self.window.set_size_request(350,150)
        self.window.set_resizable(False)
        self.window.set_title("Taille")

        self.lignes = gtk.VBox()
        self.ligne1 = gtk.HBox()
        self.ligne2 = gtk.HBox()
        self.ligne3 = gtk.HBox()

        #Label
        self.label1 = gtk.Label("Taille :")

        self.ligne1.pack_start(self.label1)

        self.labelerror = gtk.Label("")

        self.ligne2.pack_start(self.labelerror)

        #Entry
        self.entry1 = gtk.Entry()

        self.ligne1.pack_start(self.entry1)

        #Button
        self.button1 = gtk.Button("OK")
        self.button1.connect("clicked", self.take_value)

        self.ligne3.pack_start(self.button1)

        self.lignes.pack_start(self.ligne1)
        self.lignes.pack_start(self.ligne2)
        self.lignes.pack_start(self.ligne3)

        self.window.add(self.lignes)
        self.window.connect("destroy", self.destroy)
        self.window.show_all()

    def take_value(self, widget):
        """
        Recuperation des dimensions du carre entrees par l'utilisateur dans la
fenetre puis
        va ecrire ces dimensions dans le fichier taille
        """
        try:
            width = height = int(self.entry1.get_text())
            EcritureTaille(width, height)
            Fenetre.modiftaille()
        except:
            self.labelerror.set_text("Veuillez entrer des dimensions correctes")

    def main(self):
        gtk.main()

    def destroy(self, data=None):
        gtk.main_quit()
```

```
###################
# Classe Redi
###################


class Redi():
    """Classe permettant de gerer la fenetre de redimension"""
    def __init__(self):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.set_position(gtk.WIN_POS_CENTER)
        self.window.set_size_request(350,150)
        self.window.set_resizable(False)
        self.window.set_title("Dimensions")

        self.Buttons = gtk.VBox()
        self.ligne_1 = gtk.HBox()
        self.ligne_2 = gtk.HBox()
        self.ligne_3 = gtk.HBox()
        self.ligne_4 = gtk.HBox()

        #width
        self.Label_width = gtk.Label("Largeur :")
        self.Entry_width = gtk.Entry()

        self.ligne_1.pack_start(self.Label_width)
        self.ligne_1.pack_start(self.Entry_width)

        #height
        self.Label_height = gtk.Label("Longueur :")
        self.Entry_height = gtk.Entry()

        self.ligne_2.pack_start(self.Label_height)
        self.ligne_2.pack_start(self.Entry_height)

        #button
        self.Button_ok = gtk.Button("OK")
        self.Button_ok.connect("clicked", self.take_value)

        self.ligne_3.pack_start(self.Button_ok)

        #Error label
        self.Label_error = gtk.Label("")
        self.Label_error.set_visible(False)

        self.ligne_4.pack_start(self.Label_error)

        self.Buttons.pack_start(self.ligne_1)
        self.Buttons.pack_start(self.ligne_2)
        self.Buttons.pack_start(self.ligne_4)
        self.Buttons.pack_start(self.ligne_3)

        self.window.add(self.Buttons)
        self.window.connect("destroy", self.destroy)
        self.window.show_all()

    def take_value(self, widget):
        """
        Recupere la dimension depuis le widget Entry
        """
```

```
        try:
            width = int(self.Entry_width.get_text())
            height = int(self.Entry_height.get_text())
            dimension = width, height
            with open("dimensions", "wb") as file:
                data = pickle.Pickler(file)
                data.dump(dimension)
            information = "redimension"
            with open("tools", "wb") as file:
                data = pickle.Pickler(file)
                data.dump(information)
        except:
            self.Label_error.set_text("Veuillez entrer des dimensions
correctes")

    def main(self):
        gtk.main()

    def destroy(self, data=None):
        gtk.main_quit()



if __name__ == "__main__":
    Fenetre = Window()
    Fenetre.main()
```

fonctions.py

```
# coding: utf-8
import pickle
import pygame
from objet import *

def PositionMouse(tool):
    """
    Destinee a positionner le carre centre sur la position de la souris
    """
    carre_w, carre_h = tool.PositionMouse()
    mouse_x, mouse_y = pygame.mouse.get_pos()
    return (mouse_x - carre_w), (mouse_y - carre_h)


def EcritureFichier(data_write):
    """
    Va ecrire l'outil utilise (gomme, crayon, etc) dans le fichier 'tools' afin
de le transmettre avec pickle
    """
    data_write = str(data_write)
    with open("tools", "wb") as file:
        data = pickle.Pickler(file)
        data.dump(data_write)

def LectureFichier():
    """
    Va lire l'outil utilise (gomme, crayon, etc) dans le fichier 'tools'
    """
    data_read = ""
    with open("tools", "rb") as file:
        data = pickle.Unpickler(file)
        data_read = data.load()
    return data_read

def EcritureCouleur(couleur):
    """
    Va ecrire la couleur choisie dans la fenetre de choix d'une couleur dans le
fichier 'color'
    """
    with open("color", "wb") as file:
        data = pickle.Pickler(file)
        data.dump(couleur)

def LectureCouleur():
    """
    Va lire la couleur du ficher color pour changer la couleur du carre dessin
    """
    couleur = (0, 0, 0) # Valeur par default
    with open("color", "rb") as file:
        data = pickle.Unpickler(file)
        couleur = data.load()
    return couleur

def LectureTaille():
        """
        Va lire les dimensions entrées par l'utilisateur dans la fenetre de
choix de dimmension du carre de dessin
        """
```

```
        with open("taille", "rb") as file:
                data = pickle.Unpickler(file)
                taille = data.load()
        w,h = taille
        return int(w), int(h)

def EcritureTaille(h, w):
        """
         Va ecrire les dimensions du carre de dessin entrees par l'utilisateur
dans le fichier taille
        """
        taille = h, w
        with open("taille", "wb") as file:
                data = pickle.Pickler(file)
                data.dump(taille)

def LectureDimensions():
    """
    Va lire les dimensions de la fenetre de dessin dans le ficher dimensions
pour permettre de redimensionner
    la fenetre de dessin
    """
    with open("dimensions", "rb") as file:
        data = pickle.Unpickler(file)
        dimension = data.load()
    return dimension

def RedimensionEcran(width, height):
    """
    Va effectuer des verifications sur les dimensions entrees par l'utilisateur.
    Si les dimensions sont inferieures a la taille de la fenetre de depart alors
la fentre est redimensionnee sans modification
    Sinon si la largeur ou la hauteur est superieure aux dimensions de depart on
cree une surface blanche de cette dimension puis
    on recole le background de dessin par dessus cette nouvelle surface blanche.
    On fait cela pour eviter d'avoir des bandes noires sur les cotes quand la
fenetre est redimensionnee avec de plus grandes
    dimensions que celle de depart
    """
    if int(width) <= 800 and int(height) <= 600:
        size = int(width),int(height)
        return size, 0
    else:
        surface = pygame.Surface((int(width), int(height)))
        size = int(width), int(height)
        surface.fill((255, 255, 255))
        return size, surface
```

objet.py

```
# coding: utf-8
import pygame
from math import sqrt
import pickle


class Carre(pygame.sprite.Sprite):
    """Classe gerant le carre de dessin (la pointe du stylo)"""

    def __init__(self, color=(0, 0, 255), width=20, height=20):
        """
        Fonction constructeur de l'objet
        color : couleur de depart du carre (parametre optionel)
        width : largeur du carre a l'instantiation de l'objet (parametre
optionel)
        height : hauteur du carre a l'instantiation de l'objet(parametre
optionel)
        """
        pygame.sprite.Sprite.__init__(self)
        self.color = color
        self.width = width
        self.height = height
        self.image = pygame.Surface((self.width, self.height))
        self.image.fill(self.color)
        self.rect = self.image.get_rect()
        self.rect.x = 0
        self.rect.y = 0

    def Position(self, x, y):
        """
        Fonction permettant de gerer la position du carre
        x : position x ou va aller le carre
        y : position y ou va aller le carre
        """
        self.rect.x = x
        self.rect.y = y

    def PositionMouse(self):
        """
        Fonction qui retourne la moitie de la longueur et de la largeur pour
placer le carre de dessin
        """
        return (self.width/2), (self.height/2)

    def ChangementTaille(self, ch_width, ch_height):
        """
        Fonction permettant de modifier la taille du carre
        ch_width : future largeur du carre
        ch_height : future hauteur du carre
        """
        self.width = ch_width
        self.height = ch_height
        self.image = pygame.Surface((self.width, self.height))
        self.image.fill(self.color)
        self.rect = self.image.get_rect()

    def ChangementCouleur(self, ch_color):
        """
```

```
        Fonction permettant de changer la couleur du carre
        ch_color : couleur que va prendre le carre elle est donnee sous forme
RGB
        """
        self.color= ch_color
        self.image.fill(self.color)


class Gomme(pygame.sprite.Sprite):
    """Classe gerant la gomme"""

    def __init__(self, width=40, height=40):
        """
        Fonction constructeur de l'objet
        width : largeur de depart de la gomme (parametre optionel)
        height : hauteur de depart de la gomme (parametre optionnel)
        """
        pygame.sprite.Sprite.__init__(self)
        self.choix_image = "gomme_visible.png"
        self.image = pygame.image.load(str(self.choix_image))
        self.rect = self.image.get_rect()
        self.width = width
        self.height = height
        self.rect.x = 0
        self.rect.y = 0
        self.color = 255, 255, 255

    def Position(self, x, y):
        """
        Fonction permettant de gerer la position de la gomme
        """
        self.rect.x = x
        self.rect.y = y

    def PositionMouse(self):
        """Meme fonction que dans la classe Carre"""
        return (self.width/2), (self.height/2)


    def Gommer(self, reponse):
        """
        Fonction permettant de gommer
        reponse = 1 : signifie que l'utilisateur est entrain de gommer et qu'il
faut donc coller un carre blanc
        reponse != 1 : signifie que l'utilisateur n'est plus entrain de gommer
et qu'il faut donc afficher le logo de la gomme
        """
        if reponse:
            self.image = pygame.Surface((int(40),int(40)))
            self.image.fill(self.color)
            self.rect = self.image.get_rect()
        else:
            self.image = pygame.image.load(self.choix_image)
            self.rect = self.image.get_rect()


class Remplissage(pygame.sprite.Sprite):
    """Classe gerant le carre de Remplissage"""
    def __init__(self, width=0, height=0, color=(255, 255, 255)):
        pygame.sprite.Sprite.__init__(self)
        self.width = width
```

```
        self.height = height
        self.image = pygame.Surface((self.width,self.height))
        self.rect = self.image.get_rect()
        self.color = color
        self.image.fill(self.color)
        self.rect.x = 0
        self.rect.y = 0

    def Position(self, x, y):
        """Fonction permettant de gerer la position du carre de remplissage"""
        self.rect.x = x
        self.rect.y = y

    def PositionMouse(self):
        """Meme fonction que dans la classe Carre"""
        return (self.width/2), (self.height/2)

    def PointduCarre(self, x, y, x_d, y_d, color):
        """
        Permet de tracer un carre plein en fonction des points passes en
parametre
        En fonction des points passes en parametre on donne une largeur et une
hauteur a ce carre
        """
        self.width = abs(x - x_d)
        self.height = abs(y - y_d)
        self.image = pygame.Surface((self.width,self.height))
        self.color = color
        self.rect = self.image.get_rect()
        self.image.fill(self.color)

    def Resset(self, color=(255, 255, 255), width=0, height=0):
        """
        Fonction qui remet a zero les informations concernant le carre
        """
        self.image = pygame.Surface((0, 0))
        self.rect = self.image.get_rect()
        self.color = color
        self.image.fill(self.color)
        self.width = width
        self.height = height
        self.rect.x = 0
        self.rect.y = 0

class Imagess(pygame.sprite.Sprite):
    """Classe gerant les importations d'images"""
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.Surface((0,0))
        self.rect = self.image.get_rect()
        self.color = 255,255,255
        self.image.fill(self.color)
        self.rect.x = 0
        self.rect.y = 0

    def Position(self, x, y):
        """Fonction permettant de gerer de l'image importee"""
        self.rect.x = x
        self.rect.y = y

    def PositionMouse(self):
```

```
        """Meme fonction que dans la classe Carre"""
        self.width = self.image.get_width()
        self.height = self.image.get_height()
        return (self.width/2), (self.height/2)

    def ChangementImage(self, screenHeight, screenWidth): #ajout de la taille de
l'ecran car appelle de la fonction AjustementImage
        """Fonction permettant de gerer le changement d'image apres le choix par
l'utilisateur"""
        with open("image_choisie", "rb") as file:
            data = pickle.Unpickler(file)
            chemin = data.load()
        self.chemin = str(chemin)
        self.image = pygame.image.load(self.chemin)
        self.rect = self.image.get_rect()
        self.AjustementTaille(screenHeight, screenWidth)

    def AjustementTaille(self, screenHeight, screenWidth):
        """
        Fonction permettant de verfier la taille de l'image apres importation
        On recupere donc les dimensions de l'image
        Si les dimensions de l'image importee sont superieures a la fenetre de
dessin on redimesionne l'image
        aux dimensions de la fenetre
        """
        width = self.image.get_width()
        height = self.image.get_height()
        if width > screenWidth and height < screenHeight:
            self.image = pygame.transform.scale(self.image,
(screenWidth,height))
        elif height > screenHeight and width < screenWidth:
            self.image = pygame.transform.scale(self.image, (width,
screenHeight))
        elif width > screenWidth and height > screenHeight:
            self.image = pygame.transform.scale(self.image,
(screenWidth,screenHeight))
        self.rect = self.image.get_rect()

    def RedimensionnementImage(self, sens):
        """
        Fonction permettant de redimesionner l'image une fois importee
        Si sens > 0 : on agrandit l'image
        Si sens < 0 : on diminue la taille de l'image
        """
        if sens > 0:
            width = self.image.get_width()
            height = self.image.get_height()
            self.image = pygame.transform.scale(self.image, (width + 30, height
+ 30))
            self.rect = self.image.get_rect()
        elif sens < 0:
            width = self.image.get_width()
            height = self.image.get_height()
            self.image = pygame.transform.scale(self.image, (width - 30, height
- 30))
            self.rect = self.image.get_rect()

    def RotationImage(self, sens):
        """
        Fonction permettant de faire tourner l'image
        Si sens > 0 : on fait tourner l'image vers la droite
```

```
Si sens > 0 : on fait tourner l'image vers la gauche
"""
if sens > 0:
    self.image = pygame.transform.rotate(self.image, 30)
    self.rect = self.image.get_rect()
elif sens < 0:
    self.image = pygame.transform.rotate(self.image, -30)
    self.rect = self.image.get_rect()
```