Le démineur

Wissame Mekhilef

6 mai 2013

Résumé

Cet article rend compte du déroulement du projet en Informatique et Sciences du Numérique. Il est ici question de développer le code du célèbre jeu qu'est le démineur. Ce projet a débuté en fin d'année 2012 pour se terminer le 6 mai 2013. Il repose sur une modélisation UML [1] du jeu et d'une programmation avec Python 3 [2].
Je vous souhaite une bonne lecture de ce compte-rendu.

1. UML : Unified Modeling language

2. Python est un langage de programmation objet, interprété, multi-paradigme, et multi-plateformes. Il favorise la programmation impérative structurée et orientée objet. Il est doté d'un typage dynamique fort, d'une gestion automatique de la mémoire par ramasse-miettes et d'un système de gestion d'exceptions ; il est ainsi similaire à Perl, Ruby, Scheme, Smalltalk et Tcl. Source : http ://fr.wikipedia.org/wiki/Python_langage

1 Introduction

1.1 Le jeu du démineur : concept

Le jeu du démineur est un jeu inventé par Microsoft [3] sur son système d'exploitation Windows [4]. Microsoft a inventé ce jeu en même temps que le concept. Il a été largement repris depuis sur de nombreuses platformes telles que mac et linux mais aussi sur des plateformes mobiles. Le concept de ce jeu est de découvrir des mines placées sur une grille (qui peut être en 2D ou en 3D). Les informations utilisées sont relatives au voisinage des cellules de la grille selectionnées. Soient elles sont minées, sinon, une information sur son vosinage immédiat est donnée.

1.2 Le jeu du démineur : règles

Les règles de ce jeu sont relativement simples. Au début du jeu, en cliquant sur une case (au hasard) deux possibilités se présentent : soit la cellule contient une mine auquel cas le jeu est fini, sinon une information est donnée sur le voisinage. Cette information prend la forme d'un nombre. Si ce nombre est x celà signifie qu'au voisinage de cette case, il existe x mines. En utilisant ces informations, le joueur doit déduire d'une façon logique les cases ou cellules "libres". le jeu consiste donc à libérer les zones non minées et pacifier ainsi le "terrain".

1.3 Le cahier des charges

Le cahier des charges de ce projet est constitué de plusieurs items :

1. Pouvoir choisir le nombre de mines totales
2. Pouvoir choisir le nombre de cases du tableau
3. Connaître le nombre de mines qui touchent la case delectionnée (ou touchée)
4. Faire un "balayage" jusqu'à trouver une mine autour
5. Connaître le nombre de mines restantes
6. Connaître la durée de la partie (Choronomètre)
7. Tableau des scores avec nom (Liste)
8. Pouvoir marquer les mines d'un drapeau
9. Conditions d'arrêt :
   - Toutes les mines sont marquées
   - Une mine est touchée
   - Chronomètre inférieur à une durée définie

2 La gestion de projet

Ce projet a été organisé grâce à l'utilisation d'outils logiciels tels que Dropbox mais aussi grâce à des méthodes de gestion de projets professionnelles que sont le diagramme de GANTT ou encore le diagramme de PERT.

3. Microsoft est une marque déposée de Microsoft Corp. © 2012 Microsoft Corporation. Tous droits réservés.

4. Windows est une marque déposée de Microsoft Corp. © 2012 Microsoft Corporation. Tous droits réservés.

2.1 Le diagramme de Gantt

Le diagramme de Gantt m'a été utile pour avoir une vue globale du projet. Il m'a aussi permis d'avoir une appréciation plus réelle concernant la durée de développement et l'enchainement des tâches. En retour, il permet de contrôler en cours de processus et en fin de projet les temps de développement nécessaires pour pouvoir corriger et se concentrer sur les réponses à donner au cahier des charges.

	DEMINEUR	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	Tâches	S-1	S-2	S-3	S-4	S-5	S-6	S-7	S-8	S-9	S-10	S-11	S-12	S-13	S-14	S-15	S-16	S-17	S-18	S-19	S-20
1	Recherches préliminaires																				
	T-1.1 Comprendre les règles du démineur																				
	T-1.2 Apprendre quelques base de Python																				
	T-1.3 Recherche d'une base de travail																				
2	Algorithmique																				
	T-2.1 Développement du premier algorithme																				
	T-2.2 Comprendre la Programmation orientée Objet																				
	T-2.3 Comprendre la Récurcivité																				
	T-2.4 Comprendre L'utilisation des interfaces graphiques																				
3	Programmation																				
	T-3.1 Développement du code																				
	T-3.2 Création d'une bibliothèque de test																				
	T-3.3 Ajout de fonctionnalitées																				
4	Rapport																				
	T-4.1 Comprendre LaTex																				
	T-4.2 Ecriture du rapport																				
	T-4.3 Comprendre comment diviser un code																				
	T-4.4 Ajout du code source au document LaTex																				
	Recherche bibliographique																				
	Période d'activité régulière																				
	Période d'activité intensive																				

FIGURE 1 – Le diagramme de GANTT

2.2 Le diagramme PERT

Ce diagramme va de paire avec le premier et donne une vue plus globale sur les liens qu'ont les différentes tâches dans le projet.

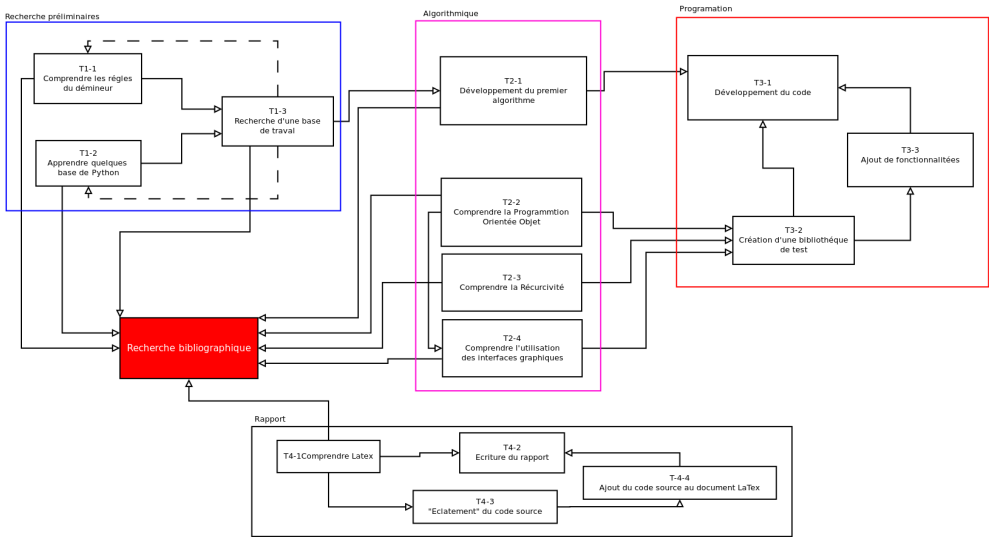

FIGURE 2 – Le digramme PERT

2.3 Récapitulatif des tâches

2.3.1 Recherches préliminaires

Comprendre les règles du démineur : Cette tâche a été la première, elle comprend aussi bien la recherche des règles côté utilisateur comme côté algortihme. En effet il fallait identifier qu'elles sont les règles que le programme doit utiliser pour le déroulement du jeu ainsi que la gestion des évènements.

Apprendre quelques bases de Python : Il m'a ensuite fallut apprendre des bases supplémentaires aux bases travaillées en cours pour pouvoir traiter entre autre de la programmation objet.

Recherche d'une base de travail : Il m'est venu à l'idée de rechercher une base de travail lorsque je me suis rendu compte que l'avancement du projet stagnait, et que je ne voyais pas par où commencer.

2.3.2 Algorithmique

Dévelopemment du premier algorithme : Le premier algorithme a donc était tiré d'une base du jeu du Ping.

Comprendre la Programmation Orientée Objet : Cette étape c'est révélée nécessaire quand j'ai observé pendant les recherches que ce type de programmation était plus simple à mettre en place et se tourne vers l'avenir.

Comprendre la Récurcivité : La récurcivité est utilisée dans le balayage de la surface de jeu. En effet la fonction "liberer" s'appelle elle-même pour pouvoir continuer le balayage après avoir traité les 8 cases adjacentes

Comprendre l'utilisation des interfaces graphiques : Contrairement à d'autre projets qui ne nécessitent pas une interface graphique le démineur se doit d'en avoir une.

2.3.3 Programmation

Dévelopement du code : Le dévelopement du code n'a pas débuté de "zéro" car j'avais à disposition le jeu du ping, qu'il a fallut tout d'abord comprendre avant de le modifier. Une fois cette étape effectuée j'ai commencé à modifier le code, à créer de nouvelles méthodes. Une fois encore l'affichage graphique étant à ma disposition, sous la forme d'exemple, il était plus simple de tester les nouvelles méthodes.

Création d'une bibliothèque de test : Une fois le corps du programme construit d'un point de vue algorithmique, il m'a fallut tester de nouvelles méthodes et pour celà créer des petits programmes. A titre d'exemple : le lancement d'une musique d'ambiance durant le jeu. Il fallait tester la musique indépendemment pour savoir si la partie de code fonctionne avant de l'intégrer au programme.

Ajout de fonctionnalités : L'ajout de fonctionnalités c'est fait en toute fin de développement, je voulais agrémenter le jeu de petites fonctions comme sauvegarder une partie et la reprendre plus tard. Mais aussi remplacer les ronds qui matérialisent les mines et les polygonnes qui représentent les drapeaux par des images à la manière du démineur de Microsoft.

2.3.4 Rapport

Comprendre LaTeX : Je me suis dirigé vers LaTeXpour écrire le rapport car il permet un affichage plus professionnel du texte et me permettait surtout grâce à l'inclusion de fichiers de pouvoir modifier mes codes sources jusqu'au dernier moment et d'avoir toujours la dernière version de mon code source dans mon rapport.

L'inclusion d'un fichier python tel que vous la voyez en lisant ce document m'a demandé beaucoup de temps, c'etait la première fois que j'incluais un code source python dans un fichier LaTeX, que j'avais auparavant utilisé pour l'écriture du rapport du TPE de Première S.

Ecriture du rapport : L'ecriture du rapport à demandé un temps de réflexion pour trouver un enchainement logique des titres.

Comprendre comment diviser un code : Comprendre comment diviser un code a été nécessaire ; en effet le parcellement en plusieurs sous parties du programme a permis un affichage plus aisé du code.

Ajout du code source au document LaTeX : L'ajout du code source au document c'est donc fait par inclusion grâce au package `listing`.

3 Concepts et fonctions informatiques nécessaires

3.1 Interface graphique

L'interface graphique est à la base du jeu démineur. Il m'a fallu d'abord trouver une base graphique de travail relativement semblable à celle du démineur, en effet je ne savais pas par où commencer. Et une fois cette base trouvée qui était celle du ping que j'ai pu trouver dans le livre de *Swinnen*, j'ai commencé à travailler sur le code pour le comprendre et ensuite le modeler pour le besoin du jeu du démineur.
L'interface du ping est très semblable au démineur, car les deux se jouent sur une grille c'est ce qui m'a permis de choisir ce programme plutôt qu'un autre pour démarrer. L'avantage que m'a fourni cette interface graphique et qui m'a beaucoup aidé été le fait de pouvoir tester immédiatement mes fonctions qui, n'avaient aucun rapport avec le jeu du "ping".
Dans le langage de programmation qu'est Pyhton les interfaces graphiques peuvent être gérées à l'aide du module `tkinter`. Ce module fonctionne parfaitement bien avec la programmation orientée objet. C'est ce qui m'a amené à travailler avec ce type de programmation. Celà a donc impliqué un apprentissage de la modéllisation UML et donc la création de classe.

3.2 La programmation objet

La programmation orientée objet est un des concepts clef de ce projet. Le principe de cette programation est simple il consiste à traiter le **tout** en tant qu'**objet**.
Un objet appartient à une classe d'objet qui peut elle aussi appartenir à une classe d'objet encore plus large. Pour résumer ceci, c'est comme si on disait que `ma chaise de bureau` appartenait à la classe `chaise, classe qui regroupe tous les types de chaises qui puissent exister, et que cette classe chaise appartenait elle aussi à un ensemble plus large qui serait l'ensemble de classs mobilier et qui regrouperait aussi la classe table.`

Dans le langage de programmation Python pour créer une classe il suffit de faire comme dans le code suivant.

defclass.py

```
1  class NomObjet(natureObjet ou objetParent):
2   "Ecrire␣ici␣une␣description␣de␣l'objet"
3   # Ici on va definir grace à la méthode constructeur qui va
4   # permettre un paramètrage automatique de l'objet
5   def __init__(self, les parametre necessaire):
6    ici initialiser des variables
7    self.nomVariable= XXX
8   # Ici on va definir des méthodes pour l'objet comme on
9   # le ferrais pour definir une fonction basique
10  def nomMéthodes(self, paramètre nécessaire):
11   ici procéder a l'ecriture␣de␣la␣fonction
12 ␣␣
```

Il faut noter que le paramètre self que l'on peut voir lors de la méthode constructeur et dans la définition des méthodes de la classe mais aussi dans les attributs de la classe permet de signaler au programme que ce qui suit self appartient à l'objet.
Maintenant que nous avons définit ce qu'était une classe, on peut créer un objet de cette classe : Instanciation. Comme on peut le voir en ?? dans l'objet Demineur, pendant la méthode constructeur on crée grâce à :
`self.mbar = MenuBar(self)`
une instance mbar de l'objet MenuBar et on fait de même pour les objets Jeu et Affichage.
Cela peut paraître un peu à part mais au contraire le fait de pouvoir créer des objets comme on le souhaite, de leur attribuer des méthodes rend plus facile la création d'une fenêtre de jeu complexe, c'est-à-dire comportant plusieurs cadres indépendants. Cela permet de pouvoir gérer de manière indépendante, l'affichage des objets sur le cadre, leur place, et leurs fonctions ainsi que leurs méthodes.
Pour représenter cette programmation il y a les schémas UML. Pour ce projet j'ai utilisé le logiciel DIA[5] qui est libre. Cette étape m'a permis de réfléchir sur la modélisation du jeu.

3.3 La récurcivité

Le principe de recurcivité est assez simple, c'est une fonction qui s'appelle elle-même. Mais ce principe de récurcivité n'est pas accepté par tous les langages de programmation.
La récurcivité a été necessaire et utilisée ici dans la fonction qui permet de libérer les cases autour de la case traitée quand celle-ci n'a pas de mines voisines.

5. DIA est un logiciel du monde libre www.dia.org

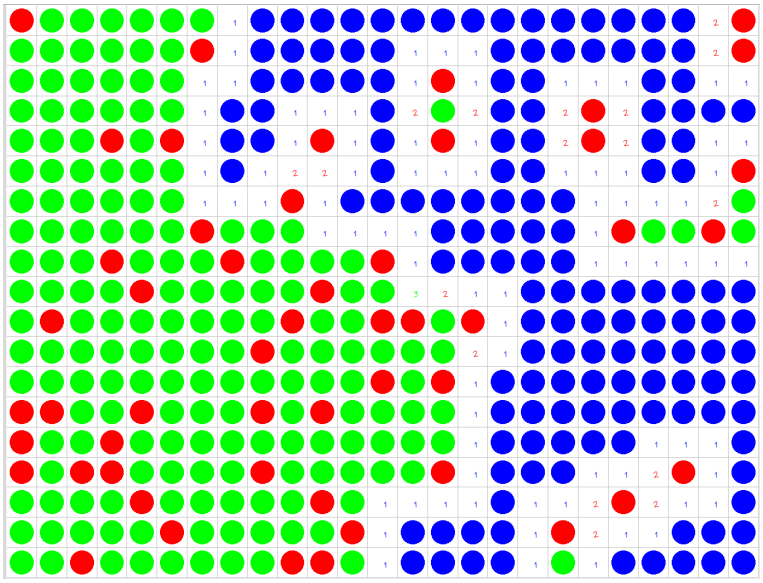

FIGURE 3 – Ce que fais la fonction liberer.py

3.4 La fonction gagne

Cette fonction est essentielle au déroulement du jeu elle n'est pas complexe mais est appellée après chaque ‘‘clic’’ sur une case par le joueur.
Cette fonction utilise le codage des cases pour vérifier leur état.

4 Algorithme

4.1 Algorithme schématisé

Ci-dessous vous pouvez voir l'algorithme représenté en UML avec le logiciel DIA. Il n'y a pas de grande boucle de jeu comme il pourrait y avoir dans une programmation qui n'est pas orientée objet et une programmation qui ne prend pas en compte la récurcivité.

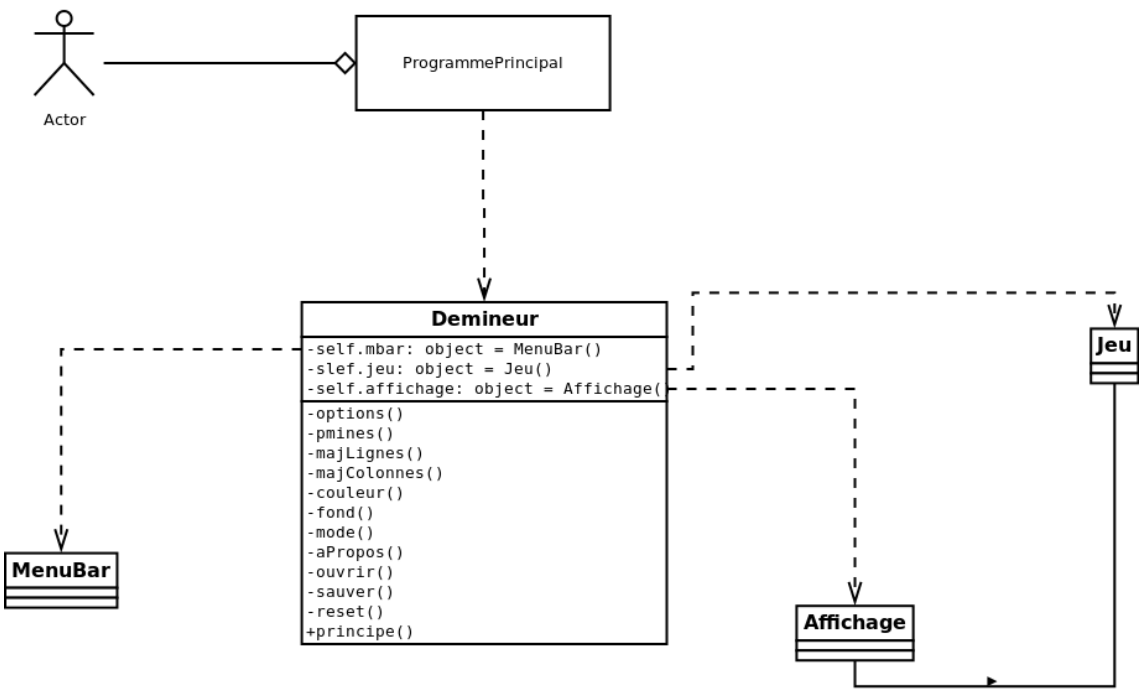

FIGURE 4 – Schéma de l'algorithme

4.2 Algorithme codé en Python 3

Le code source du projet figure en 5.

4.2.1 Outils pour comprendre le code

Tout d'abord je tiens à signaler qu'il faut pour que le jeu commence dans de bonnes conditions utiliser le bouton restart qui va initialiser le codage des mines. Sans cette manipulation le jeu ne tourne pas comme il le devrait.
Pour ce projet j'ai choisis de mettre en place un codage pour mes cases celà m'a permis de pouvoir les traiter et celà de plusieurs manières.
Le premier codage correspond à l'état des cases comme le montre le tableau ci-dessous.

État	Signification
0	Absence de mines
1	Mines
2	Libérée par un clic
3	Voisin ayant au moins une mine
4	Minée sur laquelle on a cliqué
5	Libérée par l'algorithme
6	Avec drapeau

FIGURE 5 – Tableau récapitulatif du codage de l'état des cases

Le deuxième permet d'afficher le nombre de mines adjacentes mais est aussi utile lors de l'utilisation de la fonction gagne qui a besoin de savoir entre autre si la case posséde un voisin miné ou non.

Code	Signification
0	La case n'a aucun voisin minée
1; 2; 3; 4; 5; 6; 7; 8	Indique le nombre de mine(s) qui touche(nt) la case
9	La case est miné

FIGURE 6 – Tableau récapitulatif du code des cases

Pour le codage de cet algorithme j'ai choisi de séparer toutes les parties qui composent le projet, en effet cela à permis de pouvoir ajouter facilement des méthodes (fonctions) ou objets au projet, mais aussi de permettre lors de l'écriture du compte-rendu un affichage plus aisé.

5 Conclusions

Pour finir j'aimerais aborder un aspect de la mise en relation des différentes fonctions. Elle a posé un seul problème majeur qui était due à la programmation orientée objet. Car il fallait porter une grande attention à ce qui était un attribut de l'objet de ce qui n'était pas un attribut de l'objet concerné par la méthode.
Avec la gestion du codage en tant que tel n'a pas posé de grandes difficultés.
En revanche il manque un arrêt du jeu après que la partie soit perdue ou gagnée, la gestion du clic sur la grille de jeu continue. Et le lien entre le chronomètre et la liste des joueurs n'est pas encore établi au moment de l'écriture de ce rapport.
Mais ce projet m'a beaucoup inspiré et donc demandé un certain temps pour que toutes mes fonctions fontionnent comme elle le font. Il est vrai qu'il a fallut de la patience mais le résultat n'est pas déplaisant.

Resources utilisées

{ python avec éditeur idle
{ geany
{ Dia
{ LaTeX(sous kile)
{ Dropbox
{ Libre Office

Remerciments

Je remercie pour m'avoir assité dans ce projet, Monsieur CHEVALIER[6], Monsieur PETIT[7] et Monsieur MEKHILEF[8]. J'aimerais ajouter, pour rester dans l'esprit du libre, et pour remercier la vaste communauté Python[9] et LaTeX, que les codes sources du projet et du rapport son disponibles, sur demande, au travers de Dropbox[10].

Références

[1] **www.developpez.com**.

[2] **Gérard Swinenn**. *Apprendre à programmer avec Python*

6. Professeur de Mathématiques et d'Informatique et Sciences du Numérique au Lycée Alain Fournier à Bourges.

7. Professeur de Mathématiques et d'Informatique et Sciences du Numérique au Lycée Alain Fournier à Bourges.

8. Professeur à l'Université d'Orléans.

9. Python est une marque déposée Copyright © 1990-2013, Python Software Foundation Legal Statements.

10. Dropbox est une marque déposée de Dropbox Inc. Copyright Agent Dropbox Inc. 185 Berry Street, Suite 400 San Francisco, CA 94107 copyright@dropbox.com

Annexe 1: Le programme en Python

parametre.py

```
#! /usr/bin/env python3
# -*- coding:Utf-8 -*-
from tkinter import *
from random import *
import pickle
import platform
import time
from datetime import datetime
```

ProgrammePrincipal.py

```
# Ici on importe la totalité avec * du code parametre.py.
from parametre import *
# ------------------------------------------------------------
# Ici on va importer tout les objets du programme pour faire
# tourner le jeu
from MenuBar import MenuBar
from Jeu import Jeu
from Affichage import Affichage
from Demineur import Demineur
# ------------------------------------------------------------
# Ici on céer un instance sans nom de l'objet Demineur



if __name__ == '__main__':
    Demineur().mainloop()
```

Objet : Demineur

Demineur.py

```
# On importe les paramètres
from parametre import *
# Mais on aussi besoin de tous les objets pour pouvoir
# les instanciers
from MenuBar import MenuBar
from Jeu import Jeu
from Affichage import Affichage
# On définie la classe Demineur
class Demineur(Frame):
 "corps principal du programme"
#
    ------------------------------------------------------------
```

```
12 # Ici on détermine les attributs de l'objet Demineur
13  def __init__(self):
14   # On définie la fenetre de jeu
15   Frame.__init__(self)
16   # Et on lui donne les dimensions initiale
17   self.master.geometry("900x600")
18   # On donne ensuite un titre à cette fenêtre
19   self.master.title("␣Jeu␣de␣Demineur")
20
21   # On créer une instance mbar de MenuBar
22   # C'est la barre qui se situe en haut de la fenêtre et
23   # qui permet de paramètrer le jeu.
24   self.mbar = MenuBar(self)
25   # On place ceci en indiquant quelques paramètres.
26   self.mbar.pack(side =TOP, expand =NO, fill =X)
27
28   # De même on créer une instance jeu de Jeu.
29   # C'est le cadre de jeu , celui sur lequel la grille
30   # est dessinée.
31   self.jeu =Jeu(self)
32   # On place ceci en indiquant quelques paramètres
33   # notemment celui expand qui permet au cadre de prendre
34   # le maximum de place
35   self.jeu.pack(side= LEFT, expand =YES, fill=BOTH,)
36
37   # De même on créer une instance affichage d'Afficahge
38   # C'est le cadre situé à droite de la fenêtre de jeu
39   # il permet un affichage de l'évolution de la partie.
40   self.affichage = Affichage(self)
41   # On place ceci en indiquant quelques paramètres
42   self.affichage.pack(side=RIGHT, expand = NO, fill=BOTH)
43
44   # Ici on utilise la fonction pack du module tkinter.
45   self.pack()
46
47   # Pour lancer la musique pendant le jeu on a besoin de
48   # lancer une fonction particulière en fonction du
        système
49   # d'exploitation sur lequel le programme tourne et donc
50   # d'enregistrer le nom du système dans une variable
        global
51   # Ici on va determiner sur quelle systeme le jeu tourne
52   global operatingSystem
53   # On initialise une variable qui va enregistrer le nom
        du
54   # systeme
```

```
55    operatingSystem = 'none'
56    # On attribut a cette variable le nom du system grace a
          la
57    # librairie platform
58    operatingSystem = platform.system()
59    print('Demineur : Le jeu tourne actuellement sous :',
          operatingSystem)
60
61  # Ici on importe les méthodes de l'objet Demineur
62   from options import options
63   from pmines import pmines
64   from majColonnes import majColonnes
65   from majLignes import majLignes
66   from reset import reset
67   from couleur import couleur
68   from fond import fond
69   from principe import principe
70   from aPropos import aPropos
71   from mode import mode
72   from sauver import sauver
73   from ouvrir import ouvrir
```

options.py

```
1  from parametre import *
2  # On définie la fonction options
3  def options(self):
4          "Choix du nombre de lignes et de colonnes pour la
               grille"
5          # on indique où placer ceci
6          opt =Toplevel(self)
7
8          # On définie un curseur long de 200 ayant pour
               titre
9          # "Nombre de lignes" et qui va permettre de
               choisir le
10         # nombre de ligne. On fais varier ce curseur de 1
               à 30
11         # en lui faisant prendre des valeurs entière. On
               associe
12         # à ce curseur la commande majLignes de l'objet
               Demineur
13         # On précise aussi que le curseur est horizontal
14         curL =Scale(opt, length =200, label ="Nombre de
               lignes :",
15               orient =HORIZONTAL, from_ =1, to =30, command =
                     self.majLignes)
```

```
16          # On détermine la position initiale du curseur, en
                lui
17          # affectant la valeur de la variable correspondant
                dans
18          # l'objet Jeu : nlig
19          curL.set(self.jeu.nlig)
20          curL.pack()
21
22          # On définie un curseur long de 200 ayant pour
                titre
23          # "Nombre de colonnes" et qui va permettre de
                choisir le
24          # nombre de ligne. On fais varier ce curseur de 1
                à 30
25          # en lui faisant prendre des valeurs entière. On
                associe
26          # à ce curseur la commande majColonnes de l'objet
                Demineur.
27          # On précise aussi que le curseur est horizontal
28          curH =Scale(opt, length =200, label ="Nombre de
                colonnes :",
29              orient =HORIZONTAL, from_ =1, to =30, command =
                    self.majColonnes)
30          # On détermine la position initiale du curseur, en
                lui
31          # affectant la valeur de la variable correspondant
                dans
32          # l'objet Jeu : ncol
33          curH.set(self.jeu.ncol)
34          curH.pack()
35
36          # On définie un curseur long de 200 ayant pour
                titre
37          # "Probabilitee de mines" et qui va permettre de
                choisir
38          # le nombre de ligne. On fais varier ce curseur de
                1 à 9
39          # en lui faisant prendre des valeurs entière. On
                associe
40          # à ce curseur la commande pmines de l'objet
                Demineur
41          # On précise aussi que le curseur est horizontal
42          curM =Scale(opt, length =200, label ="Probabilitee
                de mines :",
43              orient =HORIZONTAL, from_ =1, to =9, command =
                    self.pmines)
```

```
44          # On détermine la position initiale du curseur, en
                lui
45          # affectant la valeur de la variable correspondant
                dans
46          # l'objet Jeu : pmines
47          curM.set(self.jeu.pmines)
48          curM.pack()
```

pmines.py

```
1  from parametre import *
2  # On définie ici la fonction pmines qui va mettre a jour
3  # la variable pmines de l'objet Jeu
4  def pmines(self,n):
5   # On affecte à la variable pmines de l'objet Jeu
6   # la valeur choisie par le joueur par le biais du
7   # curseur en la divisant par 10.
8   # On obtient donc une probabilité comprise entre
9   # 0.1 et 0.9
10  self.jeu.pmines = (int (n))/10
11  # On affiche sur la fenêtre de commande la nouvelle
12  # valeur de la probabiltée de mines
13  print("pmines: pmines=",self.jeu.pmines)
14  # On relance une initialisation de la grille suivie
15  # de son affichage.
16  self.jeu.traceGrille()
```

majColonnes.py

```
1  from parametre import *
2  # Ici on va definir la méthode qui va permettre
3  # de mettre a jour le nombre de colonne pour la
4  # grille de jeu.
5  def majColonnes(self, n):
6   # On affecte donc à la variable définie dans
7   # l'objet Jeu prennant en compte le nombre de
8   # colonne la valeur choisie avec le curseur
9   self.jeu.ncol = int(n)
10  # Ici on va procéder à l'initialisation d'une
11  # noivelle grille de jeu puis à son affichage.
12  self.jeu.traceGrille()
```

majLignes.py

```
1  from parametre import *
2  # Ici on va définir la méthode qui va mettre à jour le
       nombre
```

```
# de ligne de la grille de jeu
def majLignes(self, n):
 # On affecte donc à la variable nlig de l'instance jeu la
 # valeur prise par le curseur.
 self.jeu.nlig = int(n)
 # On va donc comme pour la mise à jour du nombre de
    colonne
 # initiaiser une nouvelle grille et procéder à son
    affichage.
 self.jeu.traceGrille()
```

couleur.py

```
from parametre import *
# Ici on va définir la méthode qui va permettre de changer
# le mode de jeu, entre développeur et joueur.
def couleur(self):
 # On indique où placer le texte du curseur.
 msg =Toplevel(self)
 # On met un curseur qui va permettre de choisir entre la
 # valeur 1 ou 0, ce curseur renvoie à la methode mode de
 # Demineur.
 curM =Scale(msg, length =250, label ="Choisir developpeur
    ou joueur",
 orient =HORIZONTAL, from_ =0, to =1, command =self.mode)
 # On met comme valeur initiale pour le curseur la valeur
 # qui est inscrite dans la variable mode de Jeu
 curM.set(self.jeu.mode)
 curM.pack()
```

fond.py

```
from parametre import *
# Cette méthode est resté au stade de test elle n'est pas
# utilisable en l'état
def fond(self):
 # On indique où placer le message
 msg =Toplevel(self)
 # On choisit içi les paramètre du message : taille,
 # aspect, position et le texte à afficher.
 Message(msg, width =200, aspect =100, justify =CENTER,
 text ="Choisir la couleur du fond du jeux\n").pack(padx
    =10, pady =10)
```

mode.py

```
from parametre import *
```

```
2  # Ici définit la méthode qui va permettre la mise
3  # à jour du mode de visualisation de la grille
4  def mode(self,n):
5   # On affecte à la variable mode de l'objet jeu
6   # la valeur choisie avec le curseur
7   self.jeu.mode = int(n)
8   # Ici on a jsute besoin de reafficher la grille avec la
9   # nouvelle vue. Il ne faut pas initialiser un nouveau jeu
10  # au risque de ne pas pouvoir suivre la même partie
11  self.jeu.traceMaGrille()
```

sauver.py

```
1  from parametre import *
2  def sauver(self):
3   f=open('./etat', 'wb')
4   pickle.dump(self.jeu.etat, f)
5   f.close()
6
7   g=open('./code', 'wb')
8   pickle.dump(self.jeu.code, g)
9   g.close()
```

ouvrir.py

```
1  from parametre import *
2  def ouvrir(self):
3   f=open('./etat', 'rb')
4   self.jeu.etat = pickle.load(f)
5   f.close()
6
7   g=open('./code', 'rb')
8   self.jeu.code = pickle.load(g)
9   g.close()
10
11  self.jeu.traceMaGrille()
```

reset.py

```
1  from parametre import *
2  def reset(self):
3   self.jeu.traceGrille()
```

principe.py

```
1  from parametre import *
2  def principe(self):
```

```
3   "Fenetre-message contenant la description sommaire du
        principe du jeu"
4   # Ici on indique ou placer le texte
5   msg =Toplevel(self)
6   Message(msg, bg ="navy", fg ="ivory", width =400,
7    font ="Helvetica 10 bold",
8    # On insere dans la variable texte le texte que l'on
         souhaite afficher
9    text =""""Chaque case de la grille peut soit cacher une
         mine, soit être vide. Le but du jeu est de découvrir
         toutes les cases libres sans faire exploser les mines
         ,
10     c'est-à-dire sans cliquer sur les cases qui les
         dissimulent. Lorsque le joueur clique sur une case
         libre et que toutes les cases adjacentes le sont
         également,
11     une case vide est affichée. Si en revanche au moins l'
         une des cases avoisinantes contient une mine, un
         chiffre apparaît, indiquant le nombre de cases
         adjacentes
12     contenant une mine.
13     En comparant les différentes informations récoltées,
         vous avez ainsi la possibilité de progresser dans le
          déminage du terrain.
14     Si vous vous trompez et que vous cliquiez sur une mine,
          vous avez perdu.""").pack(padx =10, pady =10)
```

aPropos.py

```
1  from parametre import *
2  def aPropos(self):
3   "Fenetre-message indiquant l'auteur et le type de licence
        "
4   msg =Toplevel(self)
5   Message(msg, width =200, aspect =100, justify =CENTER,
6   text ="Jeu de Demineur\n\n(C) MEKHILEF Wissame, Fevrier
        2013.\n").pack(padx =10, pady =10)
```

Objet : MenuBar

MenuBar.py

```
1  from parametre import *
2  class MenuBar(Frame):
3      """Barre de menus déroulants"""
4      def __init__(self, boss =None):
5          Frame.__init__(self, borderwidth =2, relief =
               GROOVE)
```

```
6           ##### Menu <Fichier> #####
7           fileMenu = Menubutton(self, text ='Fichier')
8           fileMenu.pack(side =LEFT, padx =5)
9           me1 = Menu(fileMenu)
10          me1.add_command(label ='Options',
11               underline =0,command = boss.options)
12          me1.add_command(label ='Restart',
13               underline =0,command = boss.reset)
14          me1.add_command(label ='Terminer',
15               underline =0,command = boss.quit)
16          fileMenu.configure(menu = me1)
17
18          ##### Menu <gestion> #####
19          JeuMenu = Menubutton(self, text ='Gestion')
20          JeuMenu.pack(side =LEFT, padx =5)
21          me1 = Menu(JeuMenu)
22          me1.add_command(label ='Sauver', underline =0,
23               command = boss.sauver)
24          me1.add_command(label ='Ouvrir', underline =0,
25               command = boss.ouvrir)
26          JeuMenu.configure(menu = me1)
27
28          ##### Menu <Aide> #####
29          helpMenu = Menubutton(self, text ='Aide')
30          helpMenu.pack(side =LEFT, padx =5)
31          me1 = Menu(helpMenu)
32          me1.add_command(label ='Principe du jeu',
               underline =0,
33               command = boss.principe)
34          me1.add_command(label ='A propos ...', underline
               =0,
35               command = boss.aPropos)
36          helpMenu.configure(menu = me1)
37
38          ##### Menu <Configuration> #####
39          confMenu = Menubutton(self, text ='Configuration')
40          confMenu.pack(side =LEFT, padx =5)
41          me1 = Menu(confMenu)
42          me1.add_command(label ='Couleurs', underline =0,
43               command = boss.couleur)
44          me1.add_command(label ='Image de fond', underline
               =0,
45               command = boss.fond)
46          confMenu.configure(menu = me1)
```

Objet : Affichage

Affichage.py

```
1  from parametre import *
2  class Affichage(Frame):
3
4  # Ici on détermine les attributs de l'objet Affcihage
5   def __init__(self, parent):
6    Frame.__init__(self, parent)
7    self.parent = parent
8    self.initUI()
9   # Ici on détermine la méthode qui va créer le canevas
10  # et le remplir
11  def initUI(self):
12   self.pack()
13   canvas = Canvas(self)
14   canvas.create_text(0,20, anchor=W, font="Purisa",text="
       Le jeu tourne actuellement sous :")
15   canvas.create_text(0,60, anchor=W, font="Purisa",text="
       Nombre de case a liberer")
16 ##          canvas.create_text(0,60, anchor=W, font="Purisa
     ",
17 ##          text=jeu.NbCasesaLiberer)
18 ##          print("Affichage: ",NbCasesaLiberer)
19   canvas.create_text(0,100, anchor=W, font="Purisa",text="
       Nombre de mines restante(s)")
20   canvas.pack(fill=BOTH, expand=1)
```

Objet : Jeu

Jeu.py

```
1  from parametre import *
2  class Jeu(Frame):
3      """Jeu de jeu (grille de n x m cases)"""
4  #
       ------------------------------------------------------------ ----

5  # Ici on détermine les attributs de l'objet Jeu
6      def __init__(self, boss =None):
7          # Ce Jeu de jeu est constitué d'un cadre
               redimensionnable
8          # contenant lui-même un canevas. A chaque
               redimensionnement du
9          # cadre, on calcule la plus grande taille possible
                pour les
10         # cases (carrées) de la grille, et on adapte les
               dimensions du
11         # canevas en conséquence.
```

```
12          Frame.__init__(self)
13
14          self.nlig, self.ncol = 10, 10
15          # Grille initiale = 10 x 10
16          # Liaison de l'événement <resize> à un
                gestionnaire approprié :
17          self.bind("<Configure>", self.redim)
18          # Canevas :
19          self.can =Canvas(self, bg ="white", borderwidth
                =0,highlightthickness =1,
20      highlightbackground ="white")
21          # Liaison de l'événement <clic de souris> à son
                gestionnaire :
22          self.can.bind("<Button-1>", self.clicGauche)
23          self.can.bind("<Button-3>",self.clicDroit)
24          self.can.pack()
25          # Initialistaion de la probabilité de base
26          self.pmines = 0.2
27          # Initialisation d'une variable gardant le nombres
                de mines
28          self.nmines = 0
29          # Dimensionnement des tableaux aux valeurs max
30          self.code =[]
31          for i in range(30):
32              self.code.append([0]*30)
33          self.etat =[]
34          for i in range(30):
35              self.etat.append([0]*30)
36          self.NbCasesaLiberer=0
37          # Variable prennant en compte le mode de vue
38          # Programmeur ou Joueur
39          self.mode=1
40          # variable pour le nom du joueur
41          self.gagnant=""
42          # On met en place des variables qui vont
43          # enregistrer l'heure de début et l'heure de fin
44          self.t0=0
45          self.t1=0
46
47  #
        ------------------------------------------------------------
48  # Ici on importe toutes les méthodes de l'objet Jeu
49      from redim import redim
50      from initGrille import initGrille
51      from traceMaGrille import traceMaGrille
```

```
52      from liberer import liberer
53      from clicGauche import clicGauche
54      from clicDroit import clicDroit
55      from gagne import gagne
56      from traceGrille import traceGrille
```

[Jeu]

redim.py

```
1  from parametre import *
2  def redim(self, event):
3      "Opérations effectuées à chaque redimensionnement"
4      # Les propriétés associées à l'événement de
           reconfiguration
5      # contiennent les nouvelles dimensions du cadre :
6      self.width, self.height = event.width -4, event.height
           -4
7      # La différence de 4 pixels sert à compenser l'
           épaisseur
8      # de la 'highlightbordure" entourant le canevas
9      self.traceMaGrille()
```

[redim]

initGrille.py

```
1  from parametre import *
2
3  def initGrille(self):
4   # remise a zero du compteur de mines
5   self.nmines=0
6
7   # On remplis ici le tableau de mines
8   # On recupere self.nlig qui est le nombre de ligne
9   for n in range(self.nlig):
10   # et self.ncol, le nombres de colonne
11   for m in range(self.ncol):
12    difficulte=self.pmines
13    p=random()
14    if  p<difficulte :
15     # il y a une mine
16     self.etat[n][m]=1
17     # on incremente la variable nmines
18     self.nmines = self.nmines+1
19    else:
20     # il n'y a pas de mine
21     self.etat[n][m]=0
```

```
22  print("initGrille : Le jeu comporte :",self.nmines," mines")
23  self.NbCasesaLiberer= (self.ncol*self.nlig)- self.nmines
24  print("initGrille : Nombre de cases à libérer : ",self.
      NbCasesaLiberer)
25  #### CODAGE DES CASES DE LA GRILLE ####
26  # Pour chaque grille proposée au joueur on va coder
27  # chacune des cases de la grille pour permettre un
28  # "éclatement" plus simple et une recupération
29  # d'un grille aussi plus simple.
30  ### Création d'un tableau pour enregistrer le codage ###
31  self.code =[]
32  for i in range(30):
33   self.code.append([0]*30)
34
35  # PARCOURS DE LA GRILLE POUR CODER LES CASES #
36  # On parcourt toute la grille pour
37  # traiter toute les cases
38  for i in range(self.nlig):
39   for j in range(self.ncol):
40    # Si la case traitée ne comporte pas de mine
41    if self.etat[i][j]==0:
42     # On determine limin la borne inferieure en terme de
         ligne
43     limin=max(0,i-1)
44     # On determine limax la borne superieure en terme de
         ligne
45     limax=min(self.nlig-1,i+1)
46     # On determine comin la borne inferieure en terme de
         colonne
47     comin=max(0,j-1)
48     # On determine comax la borne superieure en terme de
         colonne
49     comax=min(self.ncol-1,j+1)
50     # On initialise une variable qui va être le nombre de
         mines adjacente a la case traité
51     nbm=0        #Compteur local à la case (i,j)
52     # Traitement des 8 cases adjacentes pour determiner
         combien de mines touche la case
53     for i1 in range (limin,limax+1):
54      # On fais varier i1 de la borne inf à la borne sup
55      for j1 in range (comin,comax+1):
56       # On fais varier j1 de la borne inf à la borne sup
57       if self.etat[i1][j1]!=self.etat[i][j]:
58        # Si la case traitée est différente de la case
            centrale
```

```
59          if self.etat[i1][j1]==1:
60           # Et Si la case traitée est minée
61           # Alors le compteur de mine est incrementé de 1
62           nbm=nbm+1
63       # Quand on a traité les 8 cases adjacentes, on met la
             valeur du compteur dans le tableau le nombre de
             mine qui touche la case
64       self.code[i][j]= nbm
65      else:
66       # Sinon celà veut donc dire que la case traiter est
             miné :  On la code donc en tant que tel
67       self.code[i][j]=9
68   self.t0=datetime.now()
69   print("initGrille : ", self.t0 ," heure de début du jeu"
       )
```

traceMaGrille.py

```
1  from parametre import *
2  def traceMaGrille(self):
3          "Dessin de la grille en fonction des options &
               dimensions"
4          # On doit determiner la taille maximal que pour
               nos cases
5          # et celà en fonction de la taille de la fenêtre
               et du
6          # nombre de colonnes ainsi que le nombre de lignes
               .
7          # lmax=largeur et hmax=hauteur maximales possibles
                pour
8          # les cases :
9          lmax = self.width/self.ncol
10         hmax = self.height/self.nlig
11         # Pour que la case reste un carré on donne comme
               coté à
12         # à la case le plus petit coté entre lmax et hmax.
13         self.cote = min(lmax, hmax)
14         # Maintenant il est nécessaire de redessiner le
               canevas
15         # pour l'adapter aux nombres de cases et à la
               taille
16         # de la fenêtre de jeu
17         # établissement de nouvelles dimensions pour le
               canevas :
18         larg, haut = self.cote*self.ncol, self.cote*self.
               nlig
```

```
19          # On reconfigure le canevas avec les nouvelles
                dimensions
20          self.can.configure(width =larg, height =haut)
21          #### Tracé de la grille :Jeu de Démineur ####
22          # Effacement dessins antérieurs
23          self.can.delete(ALL)
24          ### Determinantion du mode de jeu ###
25
26          ### Tracé des lignes pour le cadrillage ###
27          # On initialise une variable
28          s =self.cote
29          # On commence par les lignes horizontales.
30          # On fais donc varier l en fonction  du nombre de
                lignes
31          # pour pouvoir tracer autant de trait que l'
                utilisateur
32          # a souahité
33          for l in range(self.nlig):
34      # A chaque ligne on utilise la fonction create_line
35      # du module tkinter pour tracer la ligne.
36      # On donne comme argument a fill black pour permettre
37      # de tracer les traits en noir
38              self.can.create_line(0, s, larg, s, fill="
                    black")
39              # Ici on ajoute self.cote pour pouvoir passer
                    a la
40              # ligne suivante
41              s +=self.cote
42          # On réinitialise cette variable pour pouvoir la
43          # réutiliser.
44          s =self.cote
45          # On recommence comme pour la boucle précédentes
                sauf
46          # qu'ici on va traiter les lignes verticales
47          for c in range(self.ncol):
48              self.can.create_line(s, 0, s, haut, fill ="
                    black")
49              s +=self.cote
50          ### Dessin des ronds ###
51          # Ici on va procéder au tracé des ronds qui
                matérialise
52          # les mines.
53          # On créer une boucle qui va traiter toute les
                cases du
54          # tableau de jeu.
55          # On fais varier n jusqu'au nombre de ligne max
```

```
56          for n in range(self.nlig):
57      # On fais varier m jusqu'au nombre de colonne max
58              for m in range(self.ncol):
59       # On établit des variables
60                      x1 = m *self.cote +5          #
61                      x2 = (m +1)*self.cote -5      #
62                      y1 = n *self.cote +5          #
63                      y2 = (n +1)*self.cote -5      #
64                      # Maintenant si la case voisine n'est pas
                            miné et
65                      # qu'elle n'est pas non plus mise en
                            drapeu
66                      if (self.etat[n][m]!=3) & (self.etat[n][m
                            ]!=6):
67        # Ici on va profiter de la boucle pour gagner
68        # tu temps de calcul.
69        # Si l'etat de la variable est a 0 c'est à
70        # dire que l'utilisateur est en mode
71        # développeur
72                          if self.mode==0:
73                              # On attribut à la variable coul
                                    une
74                              # matrice qui permet de distinguer
75                              # les cases minées
76                              coul=["green","red","blue","yellow
                                    ","black","cyan","purple"][self
                                    .etat[n][m]]
77                          # Sinon alors l'utilisateur est en
                                mode jeu
78                          else :
79                              # Alors on attribut à la variable
                                    coul
80                              # une matrice ne permettant pas de
81                              # distinguer miné ou libre
82                              coul =["white","white","blue","
                                    yellow","black","cyan","purple"
                                    ][self.etat[n][m]]
83                          # Connaissant le mode de jeu on peut
                                tracer
84                          # les cercles en conséquences
85                          # Et donc si la case voisines n'est
                                pas minée
86                          # ou mise en drapeau
87                          self.can.create_oval(x1, y1, x2, y2,
                                outline ="white",width =1, fill =
                                coul)
```

```
88                      elif self.etat[n][m]==3:
89                          coul =["white","blue","red","green","
                                orange","purple","brown","black","
                                magenta"][self.code[n][m]]
90                          self.can.create_text(x1+self.cote/2,
                                y1+self.cote/2, font="Purisa",fill
                                = coul, text=self.code[n][m])
91                      # Sinon si la case est en état 6
92                      elif self.etat[n][m]==6:
93                          d=7
94                          e=2*d
95                          # On créer une matrice de point qui
                                vont
96                          # servir pour construire le drapeau
97                          points = [x1+d,y1+self.cote-e,x1+d+(
                                self.cote-e)/4,y1+d,x1+d+2*(self.
                                cote-e)/3,y1+d,x1+self.cote-e,y1+d
                                +(2*d),x1+self.cote-e,y1+d+(2*d)+
                                self.cote/3,x1+d+2*(self.cote-e)/3,
                                y1+d+self.cote/3,x1+3.225*d,y1+d+
                                self.cote/3]
98                          # On se sert de la fonction
                                creat_polygon pour
99                          # dessiner le drapeau
100                         self.can.create_polygon(points,
                                outline='black', fill='purple',
                                width=2)
```

clicDroit.py

```
1  from parametre import *
2  def clicDroit(self,event):
3   "Gestion du clic droit de la souris"
4   # On commence par determiner la ligne et la colonne :
5   lig, col = int(event.y/self.cote), int(event.x/self.cote)
6   ### On change l'etat de la case en case avec drapeau ##
7   # On verifie que la case est soit libre soit miné
8   if (self.etat[lig][col]==0) | (self.etat[lig][col]==1):
9    # Si c'est le cas alors on change l'état de la case
10   # pour quelle apparaisse avec drapeau lors du
11   # rafraichissement
12    self.etat[lig][col]=6
13  # On lance la fonction gagne pour tester si le jeu est
14  # gagné ou pas
15  self.gagne()
16  # Et on lance la fonction traceMaGrille pour rafraichir
17  # la grille de jeu
```

```
18  self.traceMaGrille()
```

clicGauche.py

```
1  from parametre import *
2  def clicGauche(self, event):
3   "Gestion du clic de souris gauche"
4   # On commence par déterminer la ligne et la colonne :
5   lig, col = int(event.y/self.cote), int(event.x/self.cote)
6   # On vérifie sur quel type de case on a cliqué
7   # Si on a cliqué sur une case miné
8   if self.code[lig][col]==9:
9    # Alors on va parcourir la grille pour coder toutes
10   # les cases en état 4
11    for m in range(0,self.ncol):
12     for n in range(0,self.nlig):
13      # Ici code pour chaque case l'état 4
14      self.etat[n][m]=4
15  # Sinon si on clique sur une case codé 0
16   elif self.code[lig][col]==0:
17    # Alors on libére la case et on appel la fonction
18    # libérer pour procéder à la libération des cases
        voisines
19    self.etat[lig][col]=2
20    # Ici on stipule que l'on souahite libérer
21    # uniquement la case codé 0 sur laquel on a cliqué
22    self.liberer(lig,col)
23  # Sinon celà veut dire que la case a un ou des
24  # voisins possédant des mines
25   else :
26    # Dans ce cas on passe dans la case dans l'état 3
27    # qui va permettre lors du passage de la fonction
28    # traceMaGrille de voir combien de mine touchent
29    # cette case
30    self.etat[lig][col]=3
31  # Ici comme pour le clic Droit on teste si le jeu est
32  # gagné
33   self.gagne
34  # et on lance cette fonction pour rafraichir la grille
35   self.traceMaGrille()
```

liberer.py

```
1  from parametre import *
2  def liberer(self,li,co):
3   # Cette fonction est appelée seulement pour liberer les
      cases
```

```
4   # adjacentes dans le cas d'un espace
5   limin=max(0,li-1)
6   limax=min(self.nlig-1,li+1)
7   comin=max(0,co-1)
8   comax=min(self.ncol-1,co+1)
9   self.etat[li][co]=2
10  for i in range(limin,limax+1):
11   for j in range(comin,comax+1):
12    if (i!=li)|(j!=co):
13     if (self.code[i][j]!=0) & (self.code[i][j]!=9):
14      self.etat[i][j]=3
15     elif (self.code[i][j]==0) & (self.etat[i][j]!=2):
16      self.etat[i][j]=5
17  self.traceMaGrille()
18
19
20  for i in range(limin,limax+1):
21   for j in range(comin,comax+1):
22    if ((i!=li)|(j!=co))&(self.etat[i][j]==5)&(self.etat[i
        ][j]!=2):
23     self.etat[i][j]=2
24     self.liberer(i,j)
25  self.traceMaGrille()
```

gagne.py

```
1  from parametre import *
2
3  def saisir_nom(self,event):                                    #
     event précise que la fonction sera utilisée après une
     action de l'utilisateur
4
5      gagnant=entr1.get()                                        #affecte
         la saisie à la variable gagnant
6      fen1.destroy()
7      self.affichageNom(gagnant)
8
9  def affichageNom(self):
10         fen1 = Tk()                                            #
             crée une fenetre graphique
11         fen1.title('Vous_entrez_dans_le_top_10')               #
             donne un nom à la fenetre
12         txt1 = Label(fen1, text = 'Entrez_votre_nom_:')        #
             crée un texte dans la fenetre
13         entr1 = Entry(fen1)                                    #
             crée une zone dans laquelle l'utilisateur
             pourra saisir un nom
```

```
14          txt1.grid(row=0,)                                   #
               place la zone de texte
15          entr1.grid(row=0,column=1)                          #
               place la zone de saisie
16          entr1.bind("<Return>",saisir_nom)                   #
               relie la commande "Entrer" à la fonction "
               saisir_nom"
17
18
19          fen1.mainloop()                                     #
               lance la boucle d'attente d'événement
20          obFichier = open('noms_des_joueurs','a')
21          obFichier.write(gagnant)
22          obFichier.write('\n')
23          obFichier.close()
24          of = open('noms_des_joueurs','r')
25          t = of.read()
26          print(t)
27
28  def gagne(self):
29   # Cette fonction sert a tester le jeu pour savoir si il
        est
30   # gagné ou pas et dans le cas contraire d'afficher des
31   # information concernant le déroulement de la partie
32   ## On initialise quelques variables ##
33   # La première compte le nombre de case qui sont dans l'
        état 4
34   compteurp=0
35   # La deuxième compte le nombre de case qui sont libérer,
36   # c-à-d soit dans l'état 2 , 3 ou 5
37   NbLibres=0
38   # On créer une boucle qui parcourt la grille
39   for n in range(self.nlig):
40    for m in range(self.ncol):
41     # Si la case rencontrée est dans l'état 4
42     if self.etat[n][m]==4:
43      # Alors on incrémente le compteurp de 1
44      # Etape longue certe car on sais déjà que si la
45      # première case rencontrée est dans l'état 4 alors
46      # toutes les autres case seront dans cette état
47      compteurp=compteurp+1
48     # Mais si la case a était libérée, état 2 , 3 ou 5
49     if ( (self.etat[n][m]==2) | (self.etat[n][m]==3) | (
          self.etat[n][m]==5)):
50      # Alors on Incrémente le compteur de case libre
51      # de 1
```

```
52      NbLibres=NbLibres+1
53  # Ici on imprime sur la fenêtre de commande le nombre de
        case
54  # restantes à libérer
55  print("gagne : Nombre de cases restantes à libérer : ",
        self.NbCasesaLiberer-NbLibres,NbLibres)
56  # Si le nombre de cases libérées est égal au nombre de
        case
57  # à libérer alors le jeu est gagné
58  if(NbLibres==self.NbCasesaLiberer):
59   # On imprime sur la fenêtre de commande l'heure de fin
60   print("gagne : heure de fin", self.t1)
61   # On enregistre l'heure de fin de la partie
62   self.t1=datetime.now()
63   # On imprime sur la fenêtre de commande que la partie
         est
64   # gagné et on indique aussi en combien de temps
65   print("gagne : tu as gagné en ",self.t1-self.t0,"
         secondes")
66   #self.saisir_nom()
67  # Sinon si le compeurp est égale au nombre de cases à
        libérer
68  # ajouté du nombre de mines que comporte le jeu. Alors
        celà
69  # veut dire que la partie est perdu
70  elif compteurp==self.NbCasesaLiberer+self.nmines:
71                  print("gagne : tu as perdu")
```

traceGrille.py

```
1  from parametre import *
2  def traceGrille(self):
3   "Fonction qui permet l'appel de deux fonctions"
4   self.initGrille()
5   self.traceMaGrille()
```

Annexe 2: Image du Projet

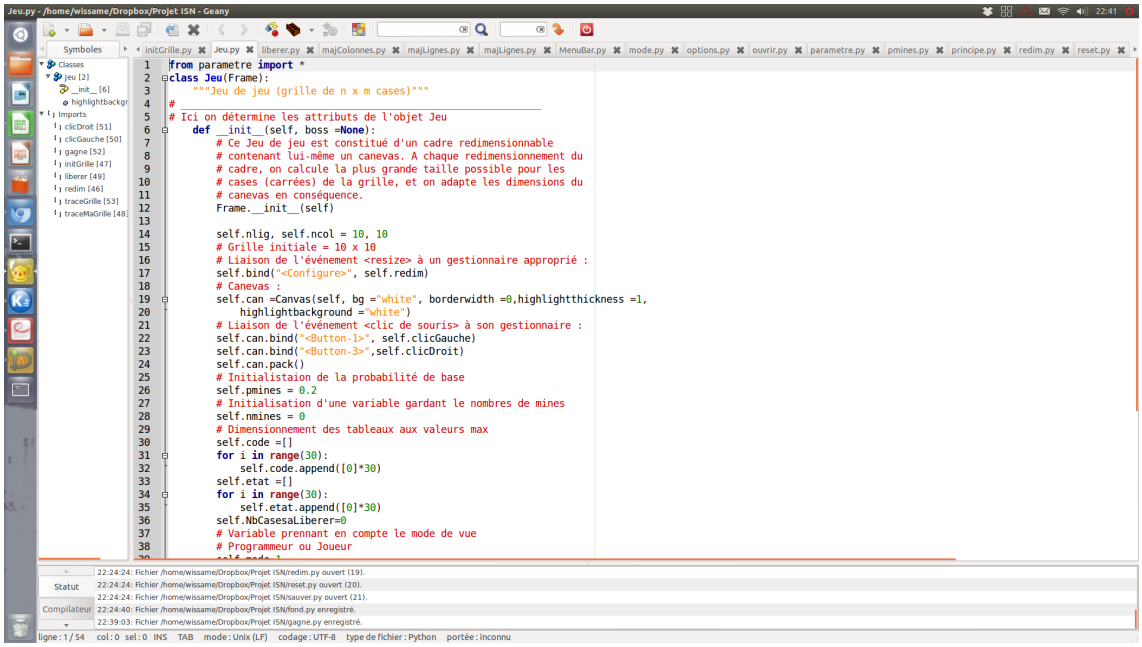

FIGURE 7 – Impression écran de la surface de ttravail du logiciel Geany

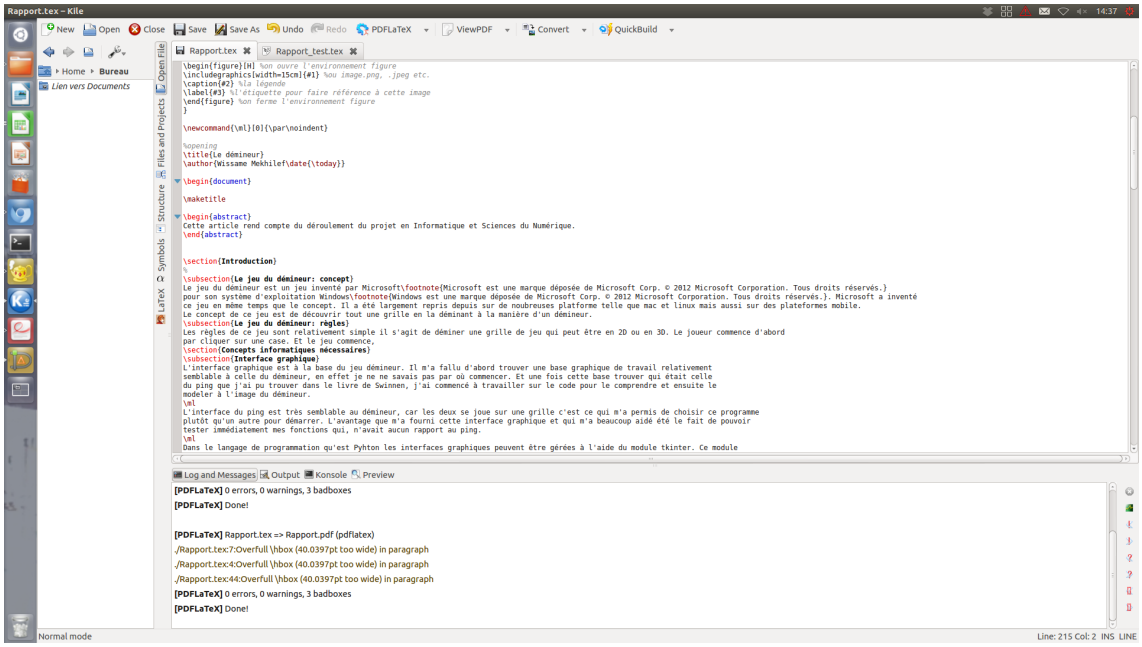

FIGURE 8 – Impression écran de la surface de travail du logiciel Kile

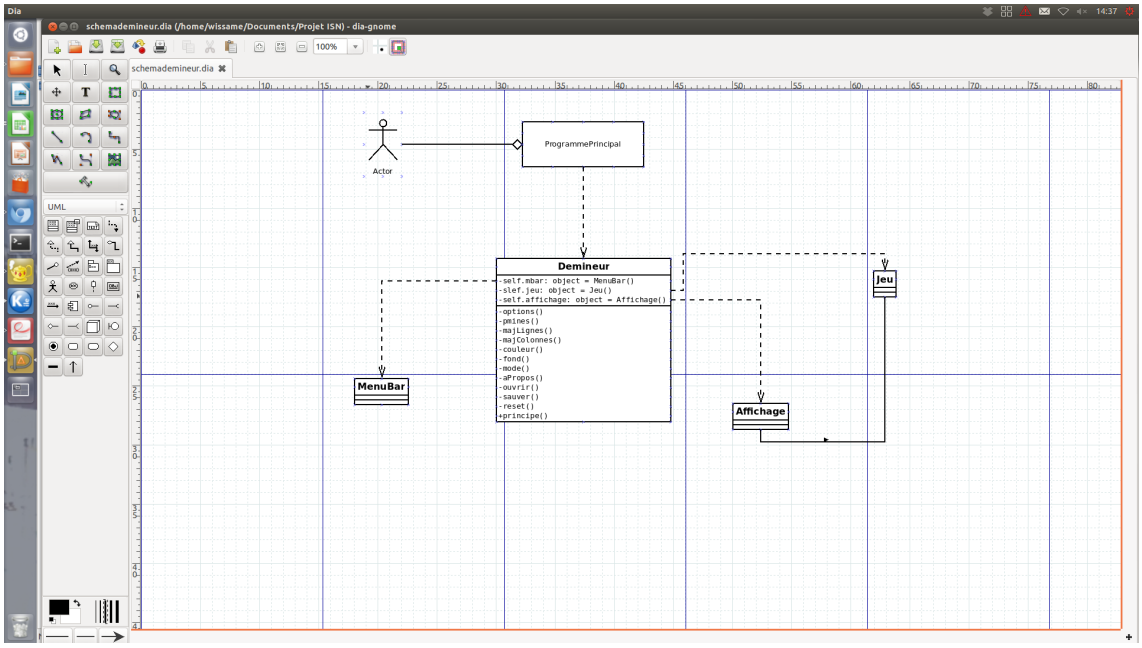

FIGURE 9 – Impression écran de la surface de travail du logiciel Dia

Table des matières

Table des figures