**DECORONAVISEUR**

Guide technique

Avril 2020

Version 1.0

G7

Stacey Cnudde

### Table des matières

[Table des matières 2](#_Toc38593051)

[1. Introduction 3](#_Toc38593052)

[2. Concept 3](#_Toc38593053)

[3. Fonctionnalités 3](#_Toc38593054)

[3.1. Inscription 3](#_Toc38593055)

[3.2. Difficulté 3](#_Toc38593056)

[3.3. Options 3](#_Toc38593057)

[3.4. Scoring. 3](#_Toc38593058)

[3.5. Modes de jeux 4](#_Toc38593059)

[Mode Normale 4](#_Toc38593060)

[Mode Propagation 4](#_Toc38593061)

[Mode Apocalypse 4](#_Toc38593062)

[4. Développement 5](#_Toc38593063)

[4.1. Diagramme de classe 5](#_Toc38593064)

[4.2. Classes et méthodes 5](#_Toc38593065)

[4.3. Etude algorithmique 5](#_Toc38593066)

[4.4. Use case 7](#_Toc38593067)

[5. Ressources 8](#_Toc38593068)

[5.1. Logiciels 8](#_Toc38593069)

[5.2. Graphisme 8](#_Toc38593070)

[Vue ingame 8](#_Toc38593071)

[Tableau d’Icones du jeu 8](#_Toc38593072)

[Affiches 9](#_Toc38593073)

[6. Test 10](#_Toc38593074)

[6.1. Politique de test 10](#_Toc38593075)

[6.2. Test unitaire 11](#_Toc38593076)

[6.3. Test d’intégration 11](#_Toc38593077)

[6.4. Test utilisateur 12](#_Toc38593078)

[7. Annexe 13](#_Toc38593079)

[7.1. Codes 13](#_Toc38593080)

[7.2. Tests 22](#_Toc38593081)

# Introduction

Ce guide reprend les spécificités techniques de l’application Decoronaviseur. Vous trouverez ici les informations nécessaires au bon déroulement d’une reprise de projet

Projet commandé par

# Concept

Decoronavirus est un jeu revisitant le principe du démineur. S’inspirant de la pandémie qui a touché le monde en 2020 le projet s’est orienté vers le VIRUS.

C’est donc autour de ce mot que se sont construits tous les piliers du développement.

Les mécaniques de jeu originales et la charte graphique agissent de concert afin d’assurer une vision cohérente du concept

# Fonctionnalités

## Inscription

Le joueur doit impérativement entrer son pseudo avant de lancer une partie. Il y est invité par un popup

## Difficulté

**Niveau Débutant :** - Lignes : 9 - Colonne : 9 - Mines : 10

**Niveau Intermédiaire :** - Lignes : 16 - Colonne : 16 - Mines : 40

**Niveau Expert :** - Lignes : 16 - Colonne : 30 - Mines : 99

## Options

**Nouvelle partie :** crée une grille en fonction des paramètres entrés par le joueur

**Règle du jeu :** Affiche les règles du jeu

**Mode de jeu :** Permet de choisir le mode de jeu (Normal / propagation / Apocalypse)

**Quitter :** Quitter le jeu

## Scoring.

Il existe différents types de compteurs (nombre de virus restants / timer incrémental / nombre de coups joués). Ceux-ci servent de base pour le calcul du score final.

**Calcul du score :** [insérer ici l’algorithme de votre choix]

Ce score est enregistré dans un fichier texte avec d’autres informations.

Exemple :

|| Pseudo || Mode || score ||

-----------------------------------------------------------------------------------------

||DesmetTePete || Apocalypse || 99999 ||

||Stasilicone85B || Normal || 2 ||

||MalaiseBaleze || Propagation || 777 ||

## Modes de jeux

### Mode Normale

----

-mécanique principal :

Une grille de cases est générée.

- condition de victoire :

Même règles qu’au démineur : toutes les cases hormis les cases virus doivent êtres découvertes

-condition de défaite

### Mode Propagation

---

-mécanique principal

- condition de victoire

-condition de défaite

### Mode Apocalypse

-----

\*\*expliquer le fonctionnement du mode apocalypse

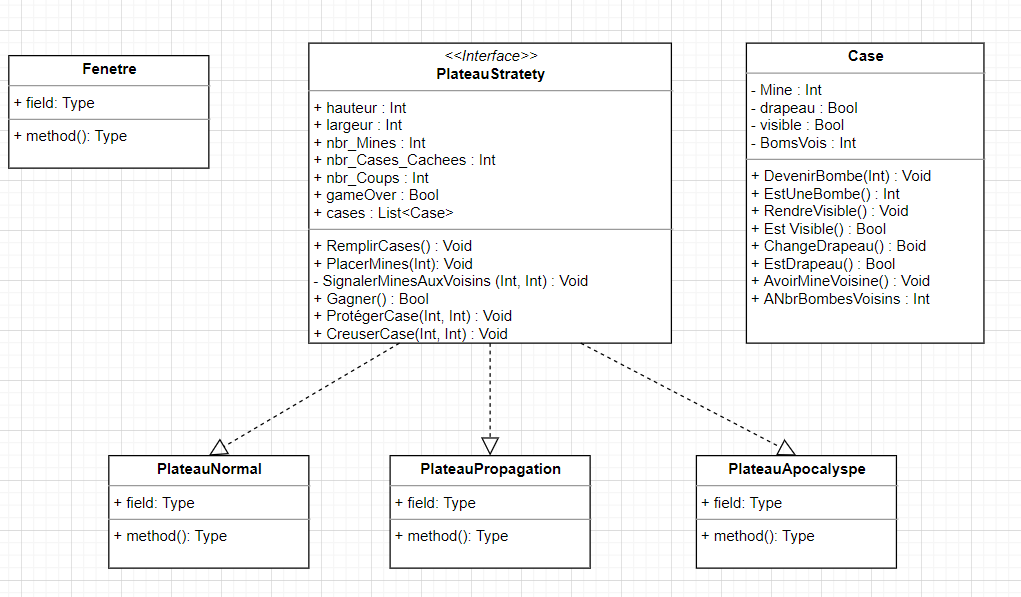
-mécanique principal

- condition de victoire

-condition de défaite

# Développement

## Diagramme de classe



## Classes et méthodes

\*\* j’ai peur que ce soit redondant avec le diagramme de classe si on liste toutes les classe mais

Certain codes méritent surement de figurer dedans (mise en évidence )

\*\*pour le reste voir annexe

[voir absolument avec DEV]

## Etude algorithmique

Etude Algorithmique du démineur

Lancement du jeu :

1. Invitation au joueur à se connecter
2. Demander à l’utilisateur de choisir son mode de jeu et de sa difficulté
3. Génération de la grille
   1. Création du tableau avec les cases
      1. Toutes les cases sont cachées
      2. Toutes les cases n’ont pas de mines
      3. Le nombre de voisins qui ont une mine est à 0
   2. Placement des mines
      1. Choix aléatoire d’une case
      2. Tant que la case choisie est minée, on rechoisit la case
      3. On mine la case choisie
      4. On signale aux voisins que la case est minée
      5. On boucle tant que le nombre de mines n’est pas atteint
      6. Si c’est une partie apocalypse
         1. 20% de Mines Propagation
         2. 20% de Mines Timer+
         3. 20% de Mines Coup+
         4. Le reste de Mines Léthale
4. Affichage de la grille

Partie normale :

1. On demande à l’utilisateur de choisir une case et s’il veut la creuser ou la marquer.
   1. S’il la marque
      1. On place un drapeau s’il n’y en a pas
      2. On retire de drapeau s’il y en a un
   2. S’il la creuse
      1. Si elle est visible ou qu’il y a un drapeau, on ne fait rien
      2. Sinon, on la creuse
         1. On la rend visible
         2. Si elle est minée, la partie est perdue
         3. Sinon, on affiche le nombre de voisins qui ont une mine
         4. Si aucun voisin n’a de mines, on creuse les cases voisines.
      3. On regarde si le nombre de case cachées restantes égale celui des mines
         1. Si oui, la partie est gagnée
         2. Sinon, la partie continue

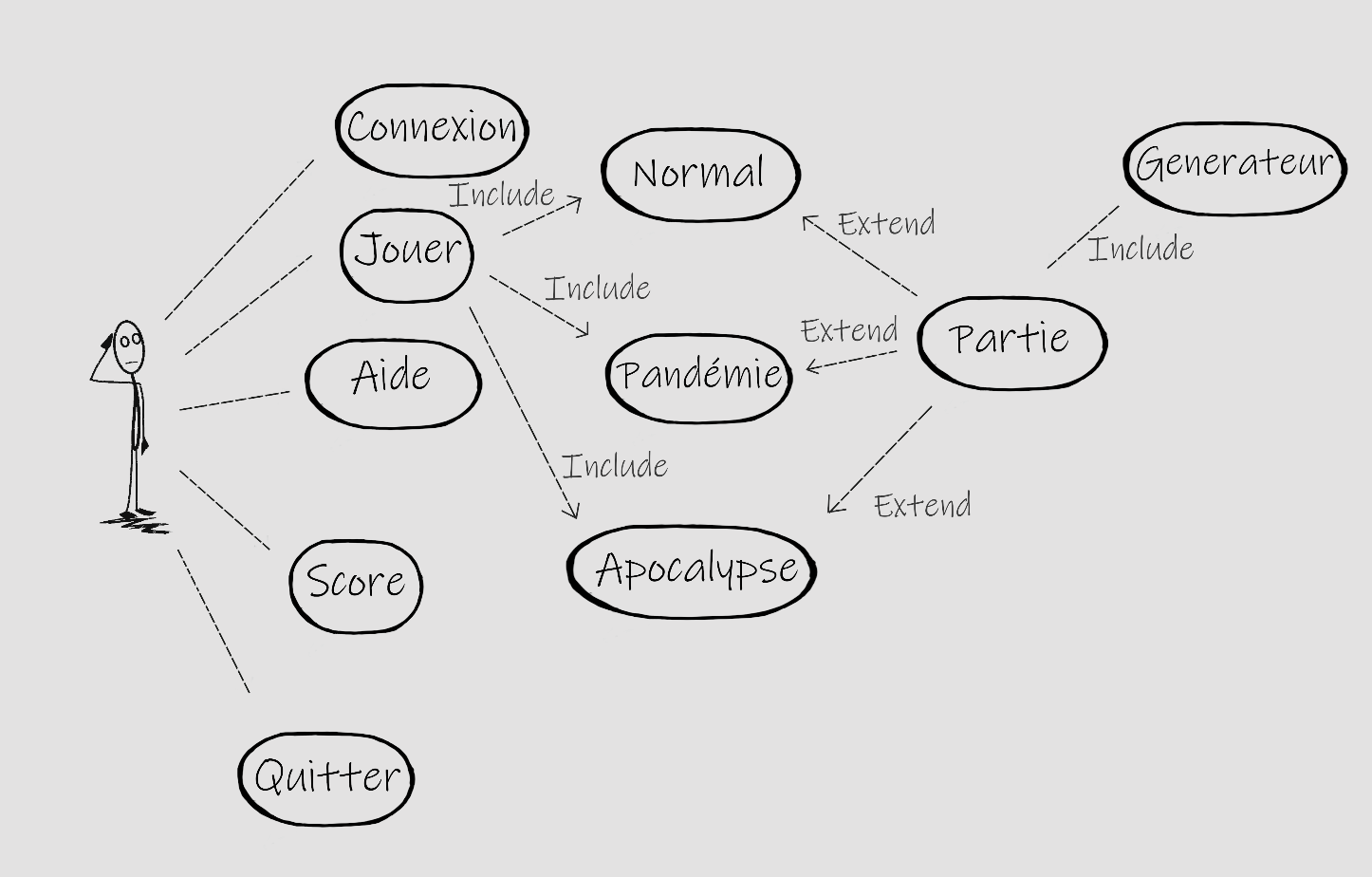
Partie Propagation

1. On demande à l’utilisateur de choisir une case et s’il veut la creuser ou la marquer.
   1. S’il la marque
      1. On place un drapeau s’il n’y en a pas
      2. On retire de drapeau s’il y en a un
   2. S’il la creuse
      1. Si elle est visible ou qu’il y a un drapeau, on ne fait rien
      2. Sinon, on la creuse
         1. On la rend visible
         2. Si elle est minée, la partie est perdue
         3. Sinon, on affiche le nombre de voisins qui ont une mine
         4. Si aucun voisin n’a de mines, on creuse les cases voisines.
      3. On regarde si le nombre de case cachées restantes égale celui des mines
         1. Si oui, la partie est gagnée
         2. Sinon, la partie continue
   3. Tous les 5 coups
      1. On regarde le nombre de cases cachées restantes moins le nombre de mines et on rajoute 1/20 de mines
         1. Choix aléatoire d’une case
         2. Tant que la case choisie est minée ou visible, on rechoisit la case
         3. On mine la case choisie
         4. On signale aux voisins que la case est minée
         5. On boucle tant que le nombre de mines n’est pas atteint

Partie Apocalypse

1. On demande à l’utilisateur de choisir une case et s’il veut la creuser ou la marquer.
   1. S’il la marque
      1. On place un drapeau s’il n’y en a pas
      2. On retire de drapeau s’il y en a un
   2. S’il la creuse
      1. Si elle est visible ou qu’il y a un drapeau, on ne fait rien
      2. Sinon, on la creuse
         1. On la rend visible
         2. Si elle est minée
            1. Si c’est une mine létale, la partie est perdue
            2. Si c’est une mine Timer+, 10 secondes sont ajoutées au compteur
            3. Si c’est une mine Coup+, 5 coups sont ajoutés au compteur coup
            4. Si c’est une mine Propagation, on rajoute 1 mines de chaque type
         3. Sinon, on affiche le nombre de voisins qui ont une mine, de n’importe quel type
         4. Si aucun voisin n’a de mines, on creuse les cases voisines.
      3. On regarde si le nombre de case cachées restantes égale celui des mines
         1. Si oui, la partie est gagnée
         2. Sinon, la partie continue

## Use case



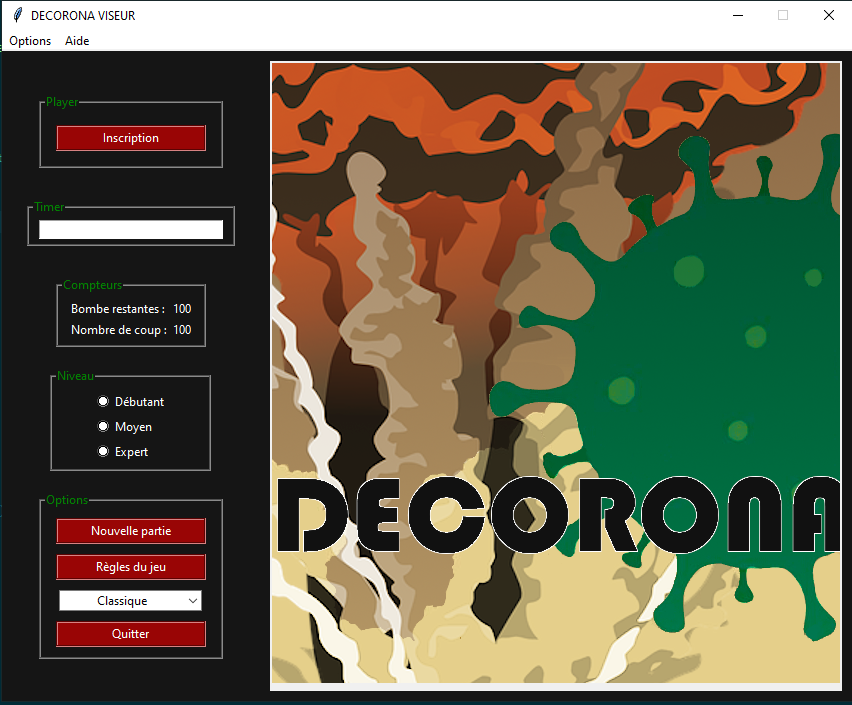
# Ressources

## Logiciels

Python 3.8.2 

## Graphisme

### Vue ingame



### Tableau d’Icones du jeu

|  |  |  |  |
| --- | --- | --- | --- |
| virusA.png | virusB.png | virusC.png | virusD.png |
| casePleine.png | caseVide.png | Docteur.png | bg.png |

### Affiches

 **

# Test

## Politique de test

1.[Dev]développement module(s).

2.[Test]test module(s).

3.[Test]mise à jour du rapport de bug.

4.[Dev]Correction Bug.

5.[Dev]Mise à jour de correction dans Rapport bug.

6.[Test] suivit et documenter.

En plus de cette procédure séquentiel, « une fiche d’identité » est attribuée à chaque classe par les développeurs. Le but étant de faciliter le travail de test en aval.

**Format fiche d’identité**

--Fiche de classe/fonction--

nom : Case

fonction/utilité attendue :

- Construction d'une Case

- Retour de son contenu en fonction de son état.

- Définir une case comme Bombe

- Augmenter le nombre de Bombe dans le voisinage

- Changer l'état Drapeau

type valeur en entrée : /

type valeur en sortie :

- Objet Case

- Bool : EstVisible, EstDrapeau

- Int : ANbrBombesVoisins, EstBombe

liste appel d'autre fonction : /

"""

**Template de rapport de bug**

|  |
| --- |
| **Type test :** |
| **Module(s) :** |
| **Description bug :** |
| **Commentaire :** |
| **Etat :** Pas corrigé |

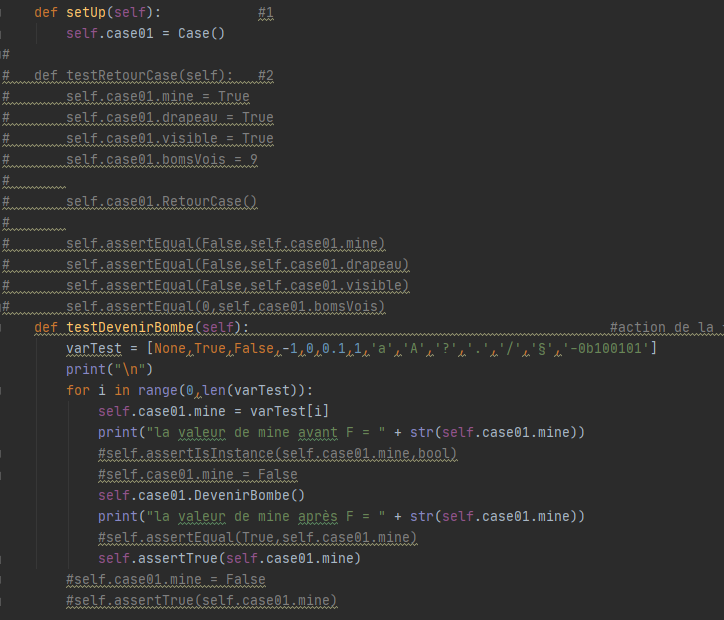
**Aide au test**

Unittest est utilisé pour les tests unitaire et d’intégration.

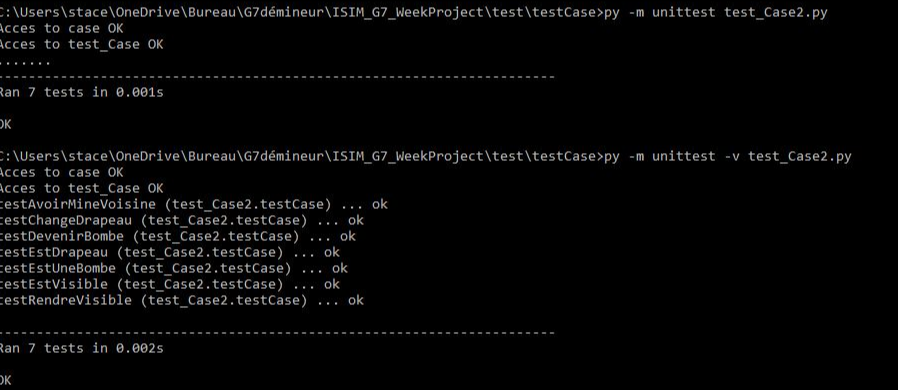
Pour les tests complémentaires il convient de se mettre à la place d’un utilisateur lambda pour vérifier les problèmes éventuels tels que le manque d’ergonomie, le manque de cohérence graphique, les réactions « inattendues » de l’utilisateur etc.

## Test unitaire

**Exemple d’une classe de test unitaire**



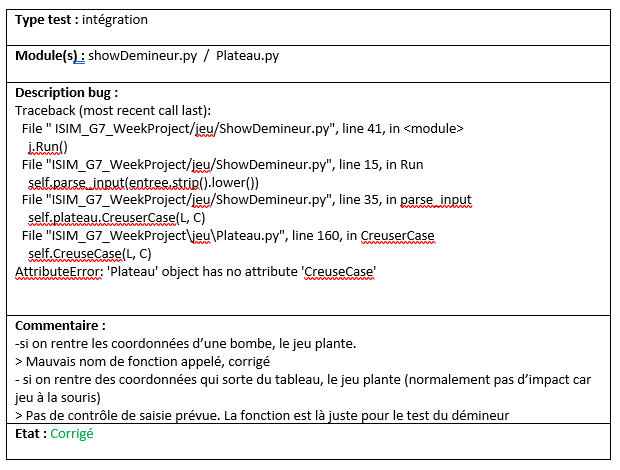
**Exemple d’un résultat de test unitaire**



## Test d’intégration

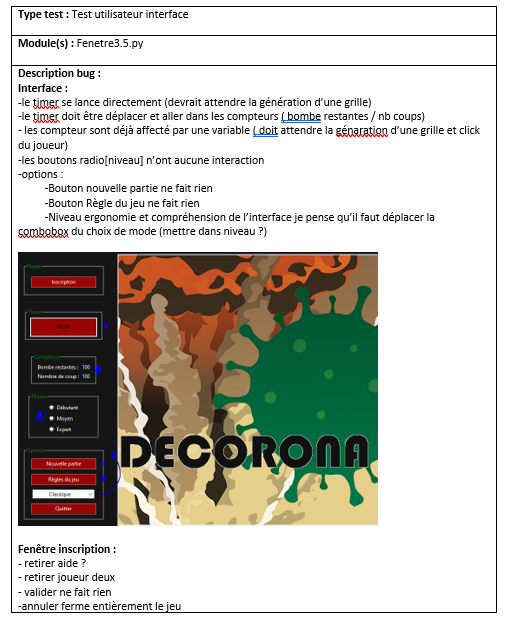
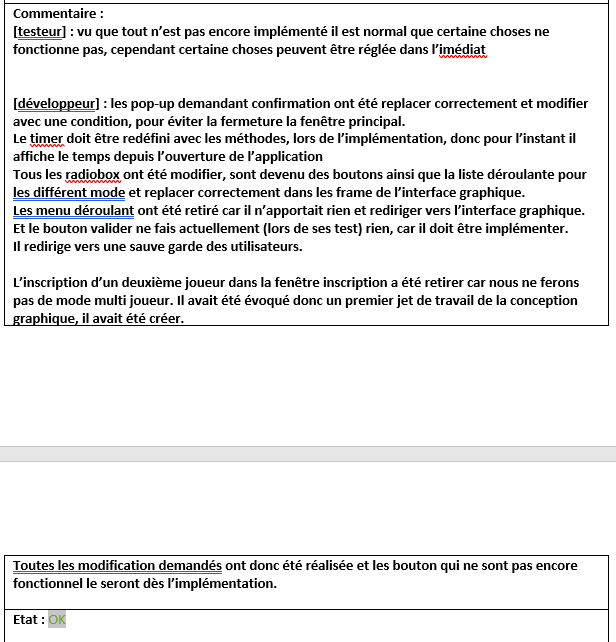
Nous avons envisagé la possibilité de réaliser les tests d’intégration en utilisant Unittest mais considérant le peu de classe à tester et le temps imparti, nous n’avons pas jugé pertinent de mettre en place un context mannager a cette fin. Les tests d’intégrations ont donc été réalisés de manière empirique

**Exemple de resultat d’un test d’intégration**



## Test utilisateur

**Exemple de test utilisateur**



Vous trouverez toute la documentation des suivis de tests dans **\test\documentation\**

# Annexe

## Codes

|  |  |
| --- | --- |
| """  --Fiche de classe/fonction--  nom : Case  fonction/utilité attendue :  - Construction d'une Case  - Retour de son contenu en fonction de son état.  - Définir une case comme Bombe  - Augmenter le nombre de Bombe dans le voisinage  - Changer l'état Drapeau  type valeur en entrée : /  type valeur en sortie :  - Objet Case  - Bool : EstVisible, EstDrapeau  - Int : ANbrBombesVoisins, EstBombe  liste appel d'autre fonction : /  """  class Case():  #Case du champs de mine    """ Attributs:  - Mine -> Int : 0 = pas de Mine, 1 = Léthal, 2 = Propagation, 3 = Timer+, 4 = Coup+  - Drapeau -> Bool : True = Drapeau, False = Pas Drapeau  - Visible -> Bool : True = La case est découverte  - BombesVoisins -> Int : Nombre de bombes dans les cases abjacentes  """    def \_\_init\_\_(self):  #Initialisation de la case. Par défaut :    self.\_\_mine = 0 #Une case n'a pas de mine.  self.\_\_drapeau = False #Une case n'a pas de drapeau.  self.\_\_visible = False #Son contenu n'est pas visible  self.\_\_bomsVois = 0 #Elle n'a pas de bombes dans son voisinnage    def DevenirBombe(self, level): #La case devient piègée  self.\_\_mine = level    def EstUneBombe(self):  return self.\_\_mine    def RendreVisible(self): #La case est découverte  self.\_\_visible = True    def EstVisible(self):  return self.\_\_visible    def ChangeDrapeau(self): #On pose ou on retire un drapeau  self.\_\_drapeau = not self.\_\_drapeau    def EstDrapeau(self):  return self.\_\_drapeau    def AvoirMineVoisine(self): #La case a une mine parmi ses voisins  self.\_\_bomsVois += 1 #On augment le nombre de ses voisins de 1    def ANbrBombesVoisins(self):  return self.\_\_bomsVois | Case.py |
| """  --Fiche de classe/fonction--  nom : Plateau  fonction/utilité attendue : Construction du plateau contenant toutess les cases du jeu  type valeur en entrée :  - Hauteur du plateau(int)  - Largeur du plateau (Int)  - Nombres de Mines (Int)  type valeur en sortie :  - Objet plateau contenant toutes les cases du Démineur  liste appel d'autre fonction :  - Constructeur  - RemplirCase  - PlacerMines(int Nombre\_de\_mines)  - RemplirCase  - Constructeur de la classe Case  - PlacerMines  - SignalerMineAuxVoisins  """  from abc import ABC  from Case import Case  import random  class PlateauTemplate(ABC):  #Plateau du Démineur    """Attributs :  - cases -> Liste : Contient toutes les cases de la grille  - hauteur -> Int : Nombre de lignes du plateau  - largeur -> Int : Nombre de colonne du plateau  - nbr\_Mines -> Int : Nombre de mines sur le plateau  - nbr\_drapeau -> Int : Nombre de drapeau sur la grille  - nbr\_Cases\_Caches -> Int (hauteur \* largeur) : Nombre de cases cachées au départ  - gameOver -> Bool : Indique la fin de partie. True = Partie Perdue  """    def \_\_init\_\_(self, hauteur, largeur, nbr\_mines):  #Initialisation du plateau    self.\_hauteur = hauteur  self.\_largeur = largeur  self.\_nbrMines = 0 #Nombre de drapeau sur la grille  self.\_nbrDrapeaux = 0 #Nombre de mines sur la grille  self.\_nbrCoups = 0  self.\_nbrCasesCachees = hauteur \* largeur  self.\_gameOver = False  self.\_cases = []    self.remplirCases() #Initialisation du plateau  self.PlacerMines(nbr\_mines)    def remplirCases(self):  #Remplit le plateau de cases initialisée    self.\_cases = [Case() for x in range(self.\_hauteur \* self.\_largeur)]    def determinerMines(self, type\_partie, nbr\_mines):  nbr\_spcl\_mines = nbr\_mines // 5  self.PlacerMines(nbr\_mines - 3\*nbr\_spcl\_mines, 1)  self.PlacerMines(nbr\_spcl\_mines, 2)  self.PlacerMines(nbr\_spcl\_mines, 3)  self.PlacerMines(nbr\_spcl\_mines, 4)  def PlacerMines(self, nbr\_mines):  #Place les mines aléatoirement dans le plateau    for x in range(nbr\_mines):  caseIndex = random.randint(0, self.\_hauteur \* self.\_largeur - 1) #On choisit aléatoirement une case du plateau  while self.\_cases[caseIndex].EstUneBombe() or self.\_cases[caseIndex].EstVisible() : #Tant que la case choisi à déjà une bombe...  caseIndex = random.randint(0, self.\_hauteur \* self.\_largeur - 1) #On recommence  ligne = caseIndex // self.\_largeur  """  La ligne de la case est obtenue en divisant l'index de la case par la largeur du plateau,  soit le nombre de colonnes  """  colonne = caseIndex % self.\_largeur  """  La colonne est donc le reste de la division ci-dessus  """  self.\_cases[caseIndex].DevenirBombe(1) #La Case devient une bombe  self.SignalerMineAuxVoisins(ligne, colonne)  self.\_nbrMines +=1    def SignalerMineAuxVoisins(self, ligne, colonne):  """  Signale la présence d'une mine à tous les voisins de la case    Chaque case est représentée comme le croissement d'une ligne et d'une colonne, et par un numéro d'index  Chaque index peut être calculée de la manière suivante : Ligne \* LargeurPlateau + Colonne.  Le plateau se représente da la manière suivante :  Chaque colonne est numérotée de 0 à n-1 largeur  Chaque ligne est numérotée de 0 à n-1 hauteur  Il y a donc (Hauteur \* Largeur) Index    | C0 | C1 | C2 | C3 |  L0 | 0 | 1 | 2 | 3 |  L1 | 4 | 5 | 6 | 7 |  L2 | 8 | 9 | 10 | 11 |    Les voisins d'une case en ligne L et colonne C sont donc la partie du tableau allant de  - L-1 à L+1  - C-1 à C+1  """  for L in range(max(0,ligne-1), min(ligne+2, self.\_hauteur)):  for C in range(max(0,colonne-1), min(colonne+2, self.\_largeur)):  self.\_cases[L \* self.\_largeur + C].AvoirMineVoisine() #On signale à la case qu'elle a une mine parmi ses voisins    def Gagner(self):  #Verifie que le nombre de case cachées est égale au nombre de mine  return self.\_nbrCasesCachees == self.\_nbrMines    def FinirPartie(self):  self.\_gameOver = True    def Perdre(self):  return self.\_gameOver    def JouerCoup(self):  self.\_nbrCoups += 1    def CalculerScore(self, timer):  return timer    def Draper(self, index):    #Pose ou retire un drapeau à la position (ligne, colonne)  #Incrémente ou décrémente le nombre de nbr\_drapeau du plateau en fonction  #ne fait rien si le contenu de la cellule est visible      #On récupère l'index de la case  case = self.\_cases[index]  if not case.EstVisible(): #Si la case n'est pas visible  case.ChangeDrapeau() #On change l'état Drapeau  if case.EstDrapeau():  self.\_nbrDrapeaux +=1  else:  self.\_nbrDrapeaux -=1    def CreuserCase(self, index):  #Creuse la case à la position (ligne, colonne)    #On récupère l'index de la case  case = self.\_cases[index]    #Si la case est visible ou a un drapeau, on ne fait rien  if case.EstVisible() or case.EstDrapeau():  return  #Sinon, on rend visible le contenu et on diminue le nombre de cases cachées  else:  case.RendreVisible()  self.\_nbrCasesCachees -=1    #Si la case est minée, la partie est perdue  #Donc, si EstBombe est différent de 0  if (case.EstUneBombe() > 0):  self.\_gameOver = True    #Si la case n'a aucune bombe dans parmi ses voisins  if (case.ANbrBombesVoisins() == 0):  ligne = caseIndex // self.\_largeur  colonne = caseIndex % self.\_largeur  #On parcout ses voisins  for L in range(max(0,ligne-1), min(ligne+2, self.\_hauteur)):  for C in range(max(0,colonne-1), min(colonne+2, self.\_largeur)):  #Et on les joue  self.CreuserCase(L \* self.\_largeur + C)  #On finit par vérifier si la partie est gagnée  self.\_gameOver = self.Gagner()    def DessinerTableau(self):  offsetY = 0  for Y in self.\_hauteur:  offsetX = 0  for X in self.\_largeur:  case = self.\_cases[Y \* self.\_largeur + X]  #case.Dessiner(X + offsetX, Y + offsetY, self.\_\_Taille, self.\_\_Taille)  offsetX += self.\_\_Taille  offsetY += self.\_\_Taille    def EstPosition(self, PosX, PosY):  offsetY = 0  for Y in self.\_hauteur:  offsetX = 0  for X in self.\_largeur:  if (X + offsetX <= PosX and PosX <= X + offsetX and Y + offsetY <= PosY and PosY <= Y + offsetY):  return Y, X  offsetX += self.\_\_Taille  offsetY += self.\_\_Taille      def AfficherTableau(self):  #Retourne la grille sous forme de string, Affichage Test    for L in range(self.\_hauteur):  print(" ", end='')  for C in range(self.\_largeur) :  case = self.\_cases[L \* self.\_largeur + C]  if self.\_gameOver and case.EstUneBombe():  print("M ", end ='')  elif not case.EstVisible():  if case.EstDrapeau():  print("D ", end='')  else:  print("X ", end='')  else:  print("%d " % (case.ANbrBombesVoisins()), end ='')  print ()  print()  print("Mines restantes = %d" % (self.\_nbrMines-self.\_nbrDrapeaux))  print("Coups = %d" % (self.\_nbrCoups))  print()    class PlateauNormal(PlateauTemplate):    def \_\_init\_\_(self, hauteur, largeur, nbr\_mines):  PlateauTemplate.\_\_init\_\_(self, hauteur, largeur, nbr\_mines)  class PlateauPropagation(PlateauTemplate):  def \_\_init\_\_(self, hauteur, largeur, nbr\_mines):  PlateauTemplate.\_\_init\_\_(self, hauteur, largeur, nbr\_mines)  self.\_\_nbrMinesItinial = nbr\_mines    def CalculerScore(self, timer):  score = timer + self.\_nbrMines - self.\_\_nbrMinesItinial  def CreuserCase(self, ligne, colonne):  PlateauTemplate.CreuserCase(self, ligne, colonne)  \_ = self.\_nbrCoups % 3  if \_ == 0 :  nbrCasesLibres = self.\_nbrCasesCachees - self.\_nbrMines  self.PlacerMines(nbrCasesLibres // 15)    class PlateauApocalypse(PlateauTemplate):  def \_\_init\_\_(self, hauteur, largeur, nbr\_mines):  PlateauTemplate.\_\_init\_\_(self, hauteur, largeur, nbr\_mines)  self.\_\_timerPlus = 0  self.\_\_nbrBombesNonLétales = 0  def PlacerMines(self, nbr\_mines):  nbr\_spcl\_mines = nbr\_mines // 5  self.PlacerSuperMines(nbr\_mines - 3\*nbr\_spcl\_mines, 1)  self.PlacerSuperMines(nbr\_spcl\_mines, 2)  self.PlacerSuperMines(nbr\_spcl\_mines, 3)  self.PlacerSuperMines(nbr\_spcl\_mines, 4)    def PlacerSuperMines(self, nbr\_mines, type):  for x in range(nbr\_mines):  caseIndex = random.randint(0, self.\_hauteur \* self.\_largeur - 1) #On choisit aléatoirement une case du plateau  while self.\_cases[caseIndex].EstUneBombe() or self.\_cases[caseIndex].EstVisible() : #Tant que la case choisi à déjà une bombe...  caseIndex = random.randint(0, self.\_hauteur \* self.\_largeur - 1) #On recommence  ligne = caseIndex // self.\_largeur  """  La ligne de la case est obtenue en divisant l'index de la case par la largeur du plateau,  soit le nombre de colonnes  """  colonne = caseIndex % self.\_largeur  """  La colonne est donc le reste de la division ci-dessus  """  self.\_cases[caseIndex].DevenirBombe(type) #La Case devient une bombe  self.SignalerMineAuxVoisins(ligne, colonne)  self.\_nbrMines +=1    def CalculerScore(self, timer):  score = timer\*2 + self.\_nbrCoups - self.\_\_nbrBombesNonLétales    def CreuserCase(self, ligne, colonne):  #Creuse la case à la position (ligne, colonne)    #On récupère l'index de la case  case = self.\_cases[ligne \* self.\_largeur + colonne]    #Si la case est visible ou a un drapeau, on ne fait rien  if case.EstVisible() or case.EstDrapeau():  return  #Sinon, on rend visible le contenu et on diminue le nombre de cases cachées  else:  case.RendreVisible()  self.\_nbrCasesCachees -=1    #Si la case est minée, la partie est perdue  #Donc, si EstBombe est différent de 0  if (case.EstUneBombe() > 0):  if(case.EstUneBombe() > 1):  self.\_\_nbrBombesNonLétales += 1  if (case.EstUneBombe() == 1):  self.\_gameOver = True  if (case.EstUneBombe() == 2):  for X in range(1, 5):  nbrCasesLibres = self.\_nbrCasesCachees - self.\_nbrMines  if nbrCasesLibres > 10:  self.PlacerSuperMines(1, X)  if (case.EstUneBombe() == 3):  self.\_\_timerPlus += 10  if (case.EstUneBombe() == 4):  self.\_nbrCoups += 5    #Si la case n'a aucune bombe dans parmi ses voisins  if (case.ANbrBombesVoisins() == 0):  #On parcout ses voisins  for L in range(max(0,ligne-1), min(ligne+2, self.\_hauteur)):  for C in range(max(0,colonne-1), min(colonne+2, self.\_largeur)):  #Et on les joue  self.CreuserCase(L, C)  #On finit par vérifier si la partie est gagnée  self.\_gameOver = self.Gagner() | Plateau.py |
| """  ================================================CLASS FENETRE =========================  ++++++++++++ STRUCURE DU FICHIER ++++++++++++++++++++++++  - IMPORTATION  - DECLARTION VARIABLE (PATH)  - FONCTION regle\_jeu  - FONCTION inscription  - CLASS appTK  - DECLARATION de la loop()  +++++++++++++ STRUCURE DE LA CLASS application ++++++++++++++++++++++++++  class application  |--->def \_\_init\_\_  |---> configuration de la fenetre (titre,bg,taille,centré)  |---> Barre de menu (var = bar\_menu,creation,déclarations)  |---> Frame  |--->  |--->  |--->  - fonction  +++++++++++++++ STRUCTURE TKinter ++++++++++++++++++++++++++++++++  pack LEFT - pack RIGHT  grid in pack LEFT  canvas in pack RIGHT  """  # IMPORTATION  import os  from tkinter import \*  from tkinter import ttk  from timeit import default\_timer  from tkinter import messagebox  PATH = os.path.dirname(os.path.realpath('\_\_file\_\_'))  def resultat():  fen=Toplevel()  fen.title("Sore")  fen.configure(bg="#151515")  fen.resizable(width=False, height=False)    #Permet un affichage centré sur l'écran  screen\_x=int(fen.winfo\_screenwidth())  screen\_y=int(fen.winfo\_screenheight())  window\_x=500  window\_y=400  posX=(screen\_x // 2) - (window\_x // 2)  posY=(screen\_y // 2) - (window\_y // 2)  geo="{}x{}+{}+{}".format(window\_x, window\_y, posX, posY)  fen.geometry(geo)    btn\_fermer=Button(fen, text="Fermer", width=15, relief=GROOVE, bg="#990505", fg="white", cursor="spraycan", command=fen.destroy)  btn\_fermer.pack(side=BOTTOM, padx=10, pady=10)    def regle\_jeu():  fene=Toplevel()  fene.title("Règle du Decorona viseur")  fene.configure(bg="#151515")  fene.resizable(width=False, height=False)    #Permet un affichage centré sur l'écran  screen\_x=int(fene.winfo\_screenwidth())  screen\_y=int(fene.winfo\_screenheight())  window\_x=500  window\_y=400  posX=(screen\_x // 2) - (window\_x // 2)  posY=(screen\_y // 2) - (window\_y // 2)  geo="{}x{}+{}+{}".format(window\_x, window\_y, posX, posY)  fene.geometry(geo)    btn\_fermer=Button(fene, text="Fermer", width=15, relief=GROOVE, bg="#990505", fg="white", cursor="spraycan", command=fene.destroy)  btn\_fermer.pack(side=BOTTOM, padx=10, pady=10)    #fonction pour ouverture de la fenêtre inscription, qui contient la possibilité d'entré ce nom  #ainsi que un bouton pour valider.  def inscription():  fenetre=Toplevel()  fenetre.title("CORONA VISEUR : Inscription")  fenetre.configure(bg="#151515")  fenetre.resizable(width=False, height=False)    #Permet un affichage centré sur l'écran  screen\_x=int(fenetre.winfo\_screenwidth())  screen\_y=int(fenetre.winfo\_screenheight())  window\_x=500  window\_y=400  posX=(screen\_x // 2) - (window\_x // 2)  posY=(screen\_y // 2) - (window\_y // 2)  geo="{}x{}+{}+{}".format(window\_x, window\_y, posX, posY)  fenetre.geometry(geo)    #zone de saisie  frame\_entree=Frame(fenetre)  frame\_entree.configure(bg="#151515")    label\_joueur1=Label(frame\_entree, text="Joueur un", bg="#151515", fg="white", padx=20, pady=20)  label\_joueur1.pack()  entree=Entry(frame\_entree, width=30)  entree.pack()  frame\_entree.pack(side=TOP, padx=20, pady=20)    frame\_btn\_utilisateur=Frame(fenetre)  frame\_btn\_utilisateur.configure(bg="#151515")    #label texte d'avertissement  label\_text=Label(fenetre, text="Tu dois t'inscrire si tu veux pouvoir jouer !!", bg="#151515", fg="green")  label\_text.pack(padx=40, pady=40)    #bouton de validation  btn\_valider=Button(frame\_btn\_utilisateur, text="Valider", width=15, relief=GROOVE, bg="#990505", fg="white", cursor="spraycan")  btn\_valider.pack(side=LEFT, padx=10, pady=10)    #bouton annuler  btn\_annuler=Button(frame\_btn\_utilisateur, text="Annuler", width=15, relief=GROOVE, bg="#990505", fg="white", cursor="spraycan", command=fenetre.destroy)  btn\_annuler.pack(side=RIGHT, padx=10, pady=10)    frame\_btn\_utilisateur.pack(side=BOTTOM, padx=20, pady=20)  #initialisation de la fenêtre principale du jeu :  #avec toutes les options de jeux.  class appTK:  def \_\_init\_\_(self, master):  #configuration de la fenêtre  maFenetre.title("DECORONA VISEUR")  maFenetre.configure(bg="#151515")  maFenetre.resizable(width=False, height=False)  #Permet un affichage centré sur l'écran  screen\_x=int(maFenetre.winfo\_screenwidth())  screen\_y=int(maFenetre.winfo\_screenheight())  window\_x=900  window\_y=750  posX=(screen\_x // 2) - (window\_x // 2)  posY=(screen\_y // 2) - (window\_y // 2)  geo="{}x{}+{}+{}".format(window\_x, window\_y, posX, posY)  maFenetre.geometry(geo)  #fonction pour quitter le jeux  def quitter():  val =messagebox.askokcancel("Quitter", "Voulez-vous vraiment partir ?", icon="error", default="ok")  if val==True:  maFenetre.quit()  else: pass    #Frame : compteurs, niveaux, boutons, timer, joueur  frame\_general=Frame(maFenetre, padx=5, pady=5) #ouverture du frame gauche principale  frame\_general.configure(bg="#151515") #configuration  #frame jouer: permet d'introduire info joueur :: A prévoir classe récupération dans un fichier texte les nom et les score  frame\_joueur=LabelFrame(frame\_general, text="Player", padx=5, pady=5)  frame\_joueur.configure(bg="#151515", fg="green")  label\_nom=Label(frame\_joueur, width=20, bg="#151515", fg="white")  label\_nom.pack()  #bouton pour introduire et enregistrer les données son nom  btn\_inscription=Button(frame\_joueur, text="Inscription", width=20, relief=GROOVE, bg="#990505", fg="white", cursor="spraycan", command=inscription)  btn\_inscription.pack(padx=5, pady=5)  frame\_joueur.grid(row=0, column=0, padx=5, pady=5)  #Frame pour les compteurs : voir pour l'affichage (fonction)  frame\_compteur=LabelFrame(frame\_general, text="Compteurs", padx=5, pady=5)  frame\_compteur.configure(bg="#151515", fg="green")  #affichage décompte bombe  texte\_bombe=Label(frame\_compteur, text="Bombe restantes :", bg="#151515", fg="white")  decompte\_bombe=Label(frame\_compteur, text="100", bg="#151515", fg="white")  texte\_bombe.grid(row=4, column=0, sticky="NW")  decompte\_bombe.grid(row=4, column=0, sticky="NE")  #affichage décompte coups  texte\_cases=Label(frame\_compteur, text="Nombre de coup :", bg="#151515", fg="white")  decompte\_cases=Label(frame\_compteur, text="100", bg="#151515", fg="white")  texte\_cases.grid(row=5, column=0, sticky="NW")  decompte\_cases.grid(row=5, column=0, sticky="NE")  def updateTime(): #fonction pour le timer    now = default\_timer() - start  minutes, secondes = divmod(now, 60)  str\_time="%02d:%02d" % (minutes, secondes)  canvas\_times.itemconfigure(text\_clock, text=str\_time)  maFenetre.after(100, updateTime)    #frame minuteur  frame\_timer=Frame(frame\_compteur)  frame\_timer.configure(bg="#151515",)  start=default\_timer()  canvas\_times=Canvas(frame\_timer, width=120, height=40, bg="#990505")  canvas\_times.pack()  text\_clock=canvas\_times.create\_text(60, 20)  frame\_timer.grid(row=2, column=0, padx=10, pady=10)  frame\_compteur.grid(row=1, column=0, padx=5, pady=5)  #frame général bouton d'option jeux  frame\_option=LabelFrame(frame\_general, text="Options", padx=5, pady=5)  frame\_option.configure(bg="#151515", fg="green")  #frame bouton  frame\_niveau=LabelFrame(frame\_option, text="Niveau", bg="#151515", fg="#FC8C00")  frame\_niveau.configure(bg="#151515")  #bouton niveau débutant  btn\_debutant=Button(frame\_niveau, width=20, text="Débutant", relief=GROOVE, bg="#990505", fg="white", cursor="spraycan")  btn\_debutant.pack(padx=5, pady=5)  #bouton niveau moyen  btn\_moyen=Button(frame\_niveau, width=20, text="Moyen", relief=GROOVE, bg="#990505", fg="white", cursor="spraycan")  btn\_moyen.pack(padx=5, pady=5)  #bouton niveau expert  btn\_expert=Button(frame\_niveau, width=20, text="Expert", relief=GROOVE, bg="#990505", fg="white", cursor="spraycan")  btn\_expert.pack(padx=5, pady=5)  frame\_niveau.pack(side=TOP, padx=5, pady=5)  # frame bouton mode  frame\_mode=LabelFrame(frame\_option, text="Mode", bg="#151515", fg="#FC8C00")  frame\_mode.configure(bg="#151515")  #bouton mode classique  btn\_classique=Button(frame\_mode, width=20, text="Classique", relief=GROOVE, bg="#990505", fg="white", cursor="spraycan")  btn\_classique.pack(padx=5, pady=5)  #bouton mode propagation  btn\_propagation=Button(frame\_mode, width=20, text="Propagation", relief=GROOVE, bg="#990505", fg="white", cursor="spraycan")  btn\_propagation.pack(padx=5, pady=5)  #bouton mmode apocalypse  btn\_apocalypse=Button(frame\_mode, width=20, text="Apocalypse", relief=GROOVE, bg="#990505", fg="white", cursor="spraycan")  btn\_apocalypse.pack(padx=5, pady=5)  frame\_mode.pack(side=BOTTOM, padx=2, pady=2)  frame\_option.grid(row=3, column=0, padx=5, pady=5)  #Frame bouton : nouvelle partie, règles du jeux, quitter, score  frame\_bouton=LabelFrame(frame\_general, text="Action", padx=10, pady=5)  frame\_bouton.configure(bg="#151515", fg="green")  #bouton nouvelle partie  btn\_jouer=Button(frame\_bouton, width=20, text="Nouvelle partie", relief=GROOVE, bg="#990505", fg="white", cursor="spraycan")  btn\_jouer.pack(padx=5, pady=5)  #bouton score  btn\_score=Button(frame\_bouton, width=20, text="Résultat", relief=GROOVE, bg="#990505", fg="white", cursor="spraycan", command=resultat)  btn\_score.pack(padx=5, pady=5)  #bouton règles du jeux  btn\_guide=Button(frame\_bouton, width=20, text="Règles du jeu", relief=GROOVE, bg="#990505", fg="white", cursor="spraycan", command=regle\_jeu)  btn\_guide.pack(padx=5, pady=5)    #bouton quitter : fermeture  btn\_quitter=Button(frame\_bouton, width=20, text="Quitter", relief=GROOVE, bg="#990505", fg="white", cursor="spraycan",command=quitter)  btn\_quitter.pack(padx=5, pady=5)  frame\_bouton.grid(row=4, column=0, padx=5, pady=5)  frame\_general.pack(side=LEFT, padx=20, pady=10) #fermeture du frame général  # canvas : affiche l'image d'ouverture du jeu  BG = PhotoImage(file= PATH+r'\bg.png')  maFenetre.BG = BG  canvasBG = Canvas(maFenetre, width=780, height=800 ) # , bg="#151515"  canvasBG.create\_image(0, 0, anchor=NW, image=BG)  canvasBG.pack(side=RIGHT,padx=50,pady=50)  #appel de fonction : associé au timer  default\_timer()  updateTime()  #Déclaration de la fenêtre principal  maFenetre=Tk() #start loop  app = appTK(maFenetre)  maFenetre.mainloop() # re loop | Fenetre3.6.3.py |

## Tests

|  |
| --- |
| **Type test :** unitaire |
| **Module(s) :** case.py |
| **Description bug :**  C:\Users\User\Documents\GitHub\G7\python>py -m unittest -v test\_Case2.py testAvoirMineVoisine (test\_Case2.testCase) ... ok testChangeDrapeau (test\_Case2.testCase) ... ok  testDevenirBombe (test\_Case2.testCase) ... ok  testRetourCase (test\_Case2.testCase) ... FAIL ====================================================================== FAIL: testRetourCase (test\_Case2.testCase) ---------------------------------------------------------------------- Traceback (most recent call last): File "C:\Users\User\Documents\GitHub\G7\python\test\_Case2.py", line 27, in testRetourCase self.assertEqual(False,self.case01.mine) AssertionError: False != True ---------------------------------------------------------------------- Ran 4 tests in 0.006s FAILED (failures=1) |
| **Commentaire :** fonction RetourCase va être modifiée. |
| **Etat :** OK |

|  |
| --- |
| **Type test :** intégration |
| **Module(s) :** showDemineur.py / Plateau.py |
| **Description bug :**  Traceback (most recent call last):  File " ISIM\_G7\_WeekProject/jeu/ShowDemineur.py", line 41, in <module>  j.Run()  File "ISIM\_G7\_WeekProject/jeu/ShowDemineur.py", line 15, in Run  self.parse\_input(entree.strip().lower())  File "ISIM\_G7\_WeekProject/jeu/ShowDemineur.py", line 35, in parse\_input  self.plateau.CreuserCase(L, C)  File "ISIM\_G7\_WeekProject\jeu\Plateau.py", line 160, in CreuserCase  self.CreuseCase(L, C)  AttributeError: 'Plateau' object has no attribute 'CreuseCase' |
| **Commentaire :**  **-**si on rentre les coordonnées d’une bombe, le jeu plante. > Mauvais nom de fonction appelé, corrigé  - si on rentre des coordonnées qui sorte du tableau, le jeu plante (normalement pas d’impact car jeu à la souris)  > Pas de contrôle de saisie prévue. La fonction est là juste pour le test du démineur |
| **Etat :** Corrigé |

|  |
| --- |
| **Type test :** Test utilisateur interface |
| **Module(s) :** Fenetre3.5.py |
| **Description bug :**  **Interface :**  **-**le timer se lance directement (devrait attendre la génération d’une grille)  -le timer doit être déplacer et aller dans les compteurs ( bombe restantes / nb coups)  - les compteur sont déjà affecté par une variable ( doit attendre la génaration d’une grille et click du joueur)  -les boutons radio[niveau] n’ont aucune interaction  -options :  -Bouton nouvelle partie ne fait rien  -Bouton Règle du jeu ne fait rien  -Niveau ergonomie et compréhension de l’interface je pense qu’il faut déplacer la combobox du choix de mode (mettre dans niveau ?)      **Fenêtre inscription :**  **-** retirer aide ?  - retirer joueur deux  - valider ne fait rien  -annuler ferme entièrement le jeu    **Fenêtre Quitter :**  **-annuler ferme quand même le jeu**  **- utiliser la croix en haut a droite ferme quand même le jeu** |
| **Commentaire :**  **[testeur] : vu que tout n’est pas encore implémenté il est normal que certaine choses ne fonctionne pas, cependant certaine choses peuvent être réglée dans l’immédiat**  **[développeur] : les pop-up demandant confirmation ont été replacer correctement et modifier avec une condition, pour éviter la fermeture la fenêtre principal.**  **Le timer doit être redéfini avec les méthodes, lors de l’implémentation, donc pour l’instant il affiche le temps depuis l’ouverture de l’application**  **Tous les radiobox ont été modifier, sont devenu des boutons ainsi que la liste déroulante pour les différent mode et replacer correctement dans les frame de l’interface graphique.**  **Les menu déroulant ont été retiré car il n’apportait rien et rediriger vers l’interface graphique.**  **Et le bouton valider ne fais actuellement (lors de ses test) rien, car il doit être implémenter.**  **Il redirige vers une sauve garde des utilisateurs.**  **L’inscription d’un deuxième joueur dans la fenêtre inscription a été retirer car nous ne ferons pas de mode multi joueur. Il avait été évoqué donc un premier jet de travail de la conception graphique, il avait été créer.**  **Toutes les modification demandés ont donc été réalisée et les bouton qui ne sont pas encore fonctionnel le seront dès l’implémentation.** |
| **Etat :** OK |