

Projet ISN

SARTORI Tom

Lycée Albert Schweitzer

2017 / 2018

Professeur d'ISN : Corinne Kesser

Groupe : Maurer Axel, Seiler Charles, Sartori Tom



Sommaire :

1. Principe du jeu et pourquoi ce projet
2. Notre manière de fonctionner
3. Avancement du projet
4. Différentes procédures et répartition du travail
5. Logiciels utilisés
6. Explication des procédures du blocage
7. Les difficultés
8. les aides
9. Futures améliorations
10. Conclusion
11. Annexe

1. Principe du jeu :

Notre jeu est un jeu de réflexion basé sur un principe assez ancien à savoir ; le labyrinthe. Les graphismes sont quant à eux pris en grande partie du jeu *The Binding of Isaac*.

Le personnage doit se déplacer de case en case dans différentes salles indépendantes plus ou moins difficiles. Pour chaque salle, le personnage dispose de vingt-et-un déplacements maximum ce qui ne facilite pas sa sortie du labyrinthe. De plus, dans certaines salles, des monstres peuvent apparaître pour l'attaquer.

Lorsque le personnage meurt ou qu'il ne peut plus se déplacer, il choisit dans un menu d'abandonner la partie ou de retourner au dernier checkpoint en perdant une de ses trois vies de départ. Le joueur peut également faire ces deux choix à n'importe quel moment de la partie en ouvrant un menu avec la touche echap.

L'intégralité de ses déplacements se font à l'aide des flèches directionnelles du clavier et les différents choix dans les menus se font avec la souris.

The Binding of Isaac



Notre jeu



J'ai imaginé le fondement de ce jeu au cours de l'année et je l'ai proposé au groupe pour le projet final. Nous étions tous d'accords pour produire ce projet et avons trouvé de nouvelles idées ensemble pour enrichir le contenu.

2. Notre manière de fonctionner :

Nous avons commencé par exposer toutes nos idées sur papier et groupe puis nous avons rapidement fait une conversation de groupe (sur facebook) pour partager nos idées et donner chacun notre point de vue. A l'aide de cette conversation, on pouvait s'informer sur notre avancement et exposer nos difficultés. Pour nous entraider, on s'appelait en vocal (sur discord) pour résoudre nos problèmes ou améliorer nos parties mais également lorsqu'une partie de programme était finie. En général Charles ou Axel me l'envoyait et je l'incorporait au programme principal en faisant les adaptations avec eux. Les modifications étaient donc rapidement mises dans le programme principal que j'ajoutais sur dropbox après.

Nous avions pas de rôle prédéfinis au sein du groupe cependant, j'étais plus ou moins le leader. Je disais aux autres membres du groupe quoi faire et je les aidais en cas de besoins.

3. Avancement du projet :

- **Étape 1** : travail sur feuille (→ 24/03)
 - Trie de toutes nos idées (→ Annexe 1)
 - Croquis
 - Dessin des différentes salles et de leurs cases bloqués

- **Étape 2** : début du code (→ 31/03)
 - Schéma graphique d'une salle aux bonnes dimensions (→ Annexe 2)
 - Code du déplacement du personnage case par case
 - Gestion du bord de la salle
 - Gestion du décompte de pas et blocage lorsqu'on arrive à zéro
 - Passage de salle en salle

- **Étape 3** : premiers menus et améliorations du designe (→ 16/04)
 - Menu lorsqu'on est bloqué
 - Compteur de salle
 - Designe du personnage (→ Annexe 3)
 - Code du menu principal
 - Modification du graphisme de la salle (→ Annexe 4)
 - Début de la création de l'IA du monstre qui se déplace en direction du personnage
 - Ajout de la touche echap pour afficher et désafficher le menu en jeu

- **Étape 4** : gestion du blocage et améliorations (→ 14/05)
 - Réglage de tous les menus pour diminuer la surcharge du canevas
 - Fin du code de l'IA
 - Gestion complète du blocage du personnage et de l'IA
 - Système de victoire et checkpoint
 - Améliorations des graphismes du blocage, des boutons et du monstre (→ Annexe 5)
 - Gestion de l'orientation du personnage

4. Les différentes procédures et la répartition du travail :

MenuPrincipal() : (Charles)

- Affiche les boutons BJouer, BChrono, BHard BQuitter qui lancent respectivement les procédures **ModeNormal()**, **ModeChrono()**, **ModeHard()**, **quitter()**.
- Supprime les boutons BReturnMenu et BReturnC lorsque *AffichageIng* est actif

ModeNormal() : (Tom)

- Enlève les boutons du menu principal
- Affiche ImFondIng1, l'image de fond en jeu
- Initialise les labels NbPas et NbSalle
- Lance **InitPerso()**

InitPerso() : (Tom)

- Est lancé après **ModeNormal()** et à chaque entrée dans une salle du personnage par **haut()**
- Modifie *x* et *y* aux coordonnées du personnage à l'entrée dans une salle et affiche PersoDos aux coordonnées *x* et *y*
- Initialise *passage* et *pas*, lance **DecomptPas()**
- Lance **InitBlocage()**
- Modifie *salle* et rafraîchit le label NbSalle
- Initialise le monstre en salle 4 (modifications de *xIA* et *yIA*, affichage de Monstre1) et supprime le monstre en salle 5
- Affiche l'image de victoire en salle 10

DecomptPas() : (Tom)

- Est lancé par **InitPerso()** et à chaque déplacement du personnage par **haut()**, **bas()**, **gauche()**, **droite()**
- Soustrait 1 à *pas* si *pas* est strictement supérieure à 0
- Rafraîchit le label NbPas
- Lance **IA()** en salle 4

InitBlocage() : (Tom)

- Est lancé par **InitPerso()** et permet de gérer le blocage du personnage pour chaque salle
- Initialise la liste *LBloc* avec quatre-vingt-onze 0
- Met dans la liste *s* la liste *s1* ou *s2* ou *s3* ou... suivant la salle correspondante et trie les valeurs de *s* dans l'ordre croissant
- Crée une liste aléatoire pour *s7*
- Avec une boucle de la longueur de *s*, on interprète les coordonnées correspondant à chaque valeur de *s* en horizontal dans LxBloc et en vertical dans LyBloc

Les fonctions **haut(event)**, **bas(event)**, **gauche(event)**, **droite(event)** : (Axel / Tom)

- Sont lancés par les flèches directionnelles du clavier
- Si *AffichageIng* ou *AffichagePri* est actif alors on sort de la fonction
- Modifient *xFut* et *yFut* puis lancent **CasePerso()**
- Si *case* est dans la liste *s*, alors on sort de la fonction
- On modifie *passage* à 1 lorsque *x* et *y* correspondent à une des avant dernières cases avant la sortie de la salle
- Dans **haut()** on lance **InitPerso()** lorsque le personnage se trouve sur la dernière case avant la sortie
- On modifie *x* et *y* et on change l'image du personnage par rapport à chaque déplacement (image de dos, des cotés et de face).
- On lance **MenuIng()** lorsque *pas* vaut 0

CasePerso() : (Tom)

- Est lancé par les quatre fonctions directionnelles
- Permet grâce à une double boucle de retranscrire les coordonnées *xFut* et *yFut* à la case correspondante dans la variable *case* (donc de récupérer une valeur entre 0 et 90 comme dans la liste *s*)

MenuIng() : (Tom)

- Est lancé par les fonctions directionnelles lorsque *pas*=0 ou par la fonction **echap(event)**
- Met *AffichageIng* à True
- Affiche les boutons BReturnMenu et BReturnC qui lancent les procédures **MenuPrincipal()** et **Checkpoint()**

echap(event) : (Tom)

- Est lancé par la touche escape du clavier
- Permet d'afficher les boutons de **MenuIng()** ou de les enlevés suivant l'état de *AffichageIng*




checkpoint() : (Tom)

- Est lancé par le bouton BReturnC
- Permet lorsque le joueur a échoué dans une salle de retourner dans une salle précédente sans recommencer au début (retour possible en salle 3 et 6)
- Modifie la variable *check* à 3 puis 6 lorsque *salle* est supérieure à ces valeurs
- lance **InitPerso()**

IA() : (Tom)

- Est lancé par les fonctions directionnelles lorsque *salle*=4
- On calcul la distance horizontale et verticale entre le perso et le monstre dans *Rx* et *Ry* avec la valeur absolue de *xIA*-*x* et *yIA*-*y*
- Lorsque *xIA* et *yIA* sont égaux on ajoute ou on enlève 1 à *xIA* de manière aléatoire
- On modifie *xIA* ou *yIA* suivant *Rx* et *Ry* pour que le monstre se déplace dans l'axe où il est le plus proche du personnage
- On vérifie avec une double boucle similaire à celle dans **CasePerso()** que *xIA* et *yIA* ne correspondent pas un emplacement bloqué
- Lorsque le perso est aux coordonnées du monstre, (le joueur perd) et on lance **MenuIng()**

5. Logiciels utilisés :

	<p>PyCharm ; environnement de développement intégré pour programmer en Python. Permet d'analyser le code et d'afficher ou de réduire certaines parties de code ce qui est utile pour la visibilité lorsque le code principal devient long.</p>
	<p>Gimp ; logiciel d'édition d'image gratuit et libre. Employé principalement pour modifier le format des images utilisées dans le programme.</p>
	<p>PhotoFiltre Studio ; logiciel de traitement d'images que j'ai utilisé pour produire les ébauches d'images utilisés aux dimensions souhaités qui ont été refaites ensuite par Charles.</p>

6. Explication des procédures du blocage

La partie du blocage est rendue possible grâce à trois procédures :

- **InitBlocage()** ; qui permet à partir d'une liste composé des numéros des cases que l'ont veut bloquer d'afficher une image du blocage à tous les endroits voulus. (→ Annexe 8)
- Les fonctions directionnelles **haut(event)**, **bas(event)**, **gauche(event)**, **droite(event)** ; qui permettent de déterminer les futurs coordonnées ($xFut$; $yFut$) du personnage et de lancer **CasePerso()**. (→ Annexe 6)
- **CasePerso()** ; qui permet de retranscrire les coordonnées futurs du personnage au numéro de la case correspondante. (→ Annexe 7)

→ **InitBlocage()** : lancé par **InitPerso()** à chaque entrée dans une salle

- LBloc est une liste de 91 valeurs pour les 91 cases d'une salle (13x7). Si LBloc vaut 0 alors il n'y a pas de blocage à son indice et si LBloc vaut 1 alors il y a blocage.
- Chaque salle possède une liste, s1 pour la salle 1, s2 pour la salle 2... on met donc dans la liste s la liste correspondant à la salle dans laquelle le personnage est. Les valeurs contenues dans s sont les cases qui seront bloqués (case 0 en haut à gauche puis ligne par ligne)
- La salle 7 donc la liste s7 est composé de 27 valeurs aléatoires de 0 à 90 grâce à randint pour avoir des cases bloqués de manière aléatoire. On ne prend pas en compte la valeur 6 et 84 car ce sont les cases d'entrée et de sortie de chaque salle.
- On trie s dans l'ordre croissant avec s.sort()
- $xBloc$ et $yBloc$ sont les coordonnées des images de blocage (différents des coordonnées du perso car taille d'image différente).
- LxBloc et LyBloc sont des listes des coordonnées en x et y des cases bloquées.
- On lance une boucle de la longueur de s :
 - a est une simple variable qui changera à chaque tour
 - A chaque tour, a est la valeur de s à l'indice i (donc a prend chaque valeur contenue dans s)
 - A l'indice de a dans LBloc, on change la valeur 0 en 1.
 - Ensuite, avec les 7 conditions si, on détermine à quelle ligne correspond a (donc le numéro de la case , car si a est sur la première ligne, alors a est entre 0 et 12 compris puis de même pour les autres lignes).
 - On modifie de cette manière $yBloc$
 - Pour trouver $xBloc$, on boucle 13 fois pour les 13 colonnes. Avec la condition si, on cherche ce que vaut a modulo 13 et donc on trouve le numéro de la colonne.
 - $xBloc$ vaut 155 (coordonnée de la première colonne) + le numéro de la colonne*72 (chaque case fait 72px).
 - On affiche donc l'image du blocage (ImBloc1) aux coordonnées $xBloc$ et $yBloc$ avec can.create_image.
 - On ajoute les coordonnées du blocage $xBloc$ et $yBloc$ dans les listes LxBloc et LyBloc.
- On procède donc de cette même manière pour chaque valeur de s et on affiche donc toutes les cases bloqués.

- **haut(event), bas(event), gauche(event) ou droite(event)** : lancés par les flèches directionnelles du clavier.
(La fonction étant assez grande, nous allons cibler la partie en rapport avec le blocage vue sur le ppt ou entre les lignes 12 et 19).
 - $xFut$ et $yFut$ sont globalisés et correspondent aux coordonnées x et y après que le personnage ait effectué le mouvement correspondant à sa fonction sans qu'il ne se soit encore déplacé)
 - s est une liste globalisée provenant de **InitBlocage()** et $case$ est une variable globalisée pour **CasePerso()**.
 - On exécute **CasePerso()** qui produira la variable $case$ qui correspond au numéro de la case où le personnage va se trouver.
 - Si le numéro de $case$ est dans la liste s (donc que le numéro de la prochaine case du personnage est dans la liste des numéros des cases bloquées) alors, on sort de la fonction et le déplacement avec `return()` et le déplacement ne sera pas effectué.
- **CasePerso()** ; lancé par chaque fonction directionnelle.
 - A partir des coordonnées $xFut$ et $yFut$, on cherche le numéro de la case correspondante.
 - On globalisé $yFut$ et $xFut$ produits dans les fonctions directionnelles et $case$ pour ces mêmes fonctions.
 - On initialise $case$ à n'importe quel nombre
 - On effectue une boucle `for` de longueur 7 (pour les 7 lignes d'une salle) avec j comme variable
 - On effectue une seconde boucle `for` à l'intérieur de la première, de longueur 13 (13 colonnes) avec la variable i .
 - Avec la condition `si`, on cherche pour quelle valeur de j , $yFut = 228$ (y de la première ligne) + $j*72$ (72 px par case). On cherche donc à quelle ligne est $yFut$. De même pour $xFut$.
 - Ainsi, on a i et j correspondants donc $case$ (le numéro de la case) = i (le numéro de la colonne) + $j * 13$ (le nombre de lignes qu'on rajoute, car pour descendre d'une ligne, on ajoute 13).

7. Difficultés :

- **Travail en groupe** ; le fait de devoir obligatoirement attendre certaines parties du reste du groupe pour pouvoir continuer était assez contraignant. J'ai perdu du temps au début à cause de ça et j'ai donc rapidement décidé d'avancer à un rythme plus soutenu pour mener à bien le projet au risque de devoir en faire plus que les autres.
- **Liaison entre les procédures** pas toujours évidente à cause des variables globalisés, de leurs initialisations mais également par rapport à la suite des événements pour que tout soit dans le bon ordre. C'est cependant une difficulté assez intéressante.
- **Gestion de l'interface graphique** ; ajout et suppression des boutons / images dans les différents menus, gestion de tous les éléments dans un même canevas unique.
- **Blocage du personnage** ; conception du code.
- Je me préoccupais de **l'optimisation et la propreté** du code de manière à avoir un résultat lisible et compréhensible sans trop d'efforts ce qui rendait cette partie pas toujours évidente.

8. Aides :

- Principalement [openclassrooms](https://openclassrooms.com)
- developpez.com / apprendre-python.com / python.org

9. Futures améliorations :

- Meilleurs graphismes
- Ajout d'une monnaie et d'un magasin dans le jeu
- Création de nouveaux niveaux
- Création de nouveaux modes
- Classement des meilleurs joueurs (en local)
- Format du jeu au choix ; pouvoir modifier la taille de la fenêtre du jeu suivant la taille de son écran

10. Conclusion :

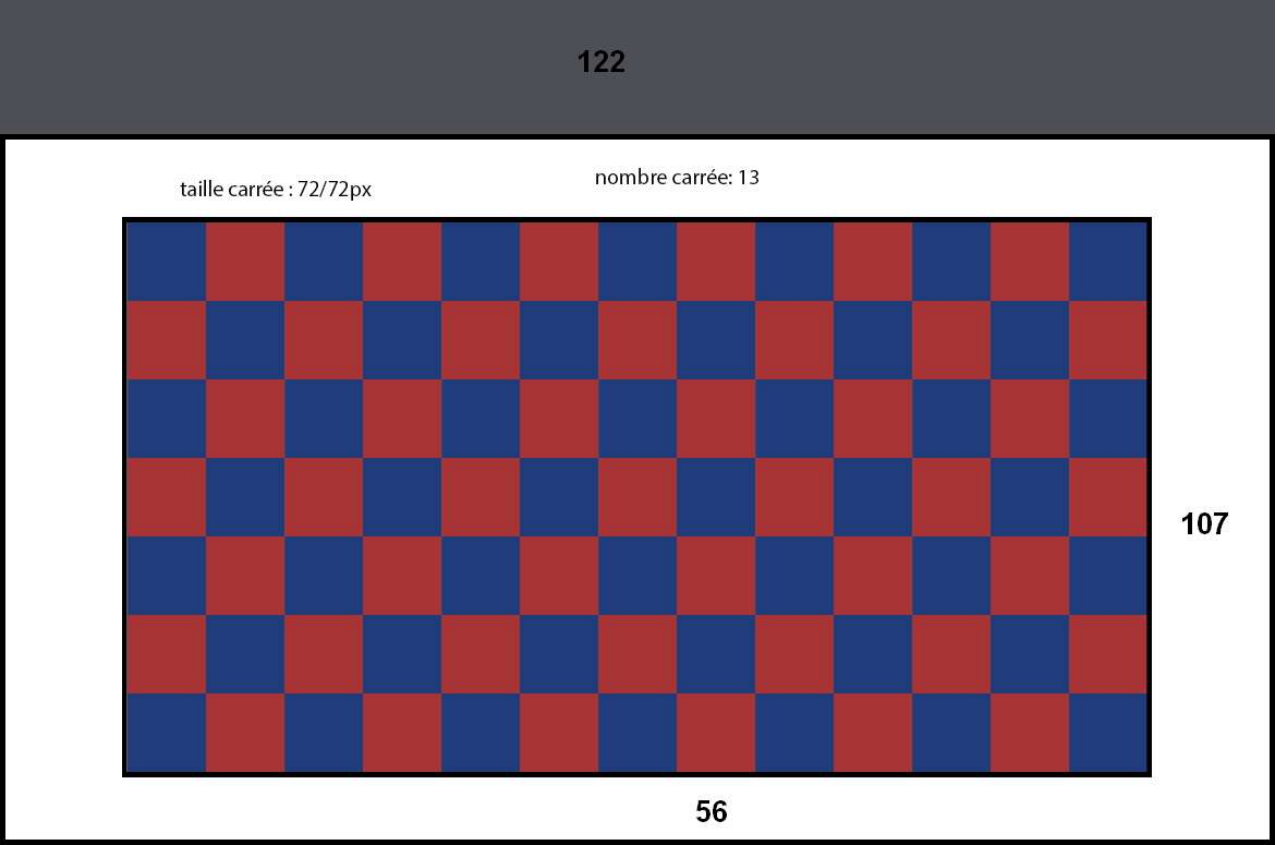
Ce projet était assez intéressant à réaliser. Il m'a apporté de nouvelles connaissances surtout au niveau de l'interface graphique mais il m'a également permis d'acquérir de nouveaux automatismes pour coder plus rapidement. Une optimisation globale du code est clairement possible même si je trouve le code assez propre en général. Beaucoup d'ajouts sont envisageables et seront fait pour avoir un jeu totalement fini. Je suis tout de même assez content du résultat même si le travail d'équipe n'était pas toujours présent. Ce projet pourra tout de même m'être utile dans mes futures études en informatique.

11. Annexe :

1.

- Designe salles et interface (schéma)
- Designe des salles sur papier en rapport à la difficulté.
- Designe du fond des salles
- Designe des obstacles
- Designe du fond du menu
- Designe des boutons du menu
- Code du bonhomme qui avance case par case
- Gestion des rochers
- Décompte des pates
- Gestion des entrées et sorties des salles
- Gestion du problème lorsqu'on dépasse le nombre de copies
- Création des checkpoints
- Gestion des vies
- Gestion de la monnaie
- Création du shop
- Designe salle du shop
- Designe interface du shop
- Code d'entrée dans l'interface du shop
- Gestion du shop
- Création de l'IA des monstres
- Designe des monstres
- Code du SUMP
- Code du SPEED
- Code du SHIELD
- Code du Restart
- Code victoire

2.



3.



4.



5.



6.

```
def haut(event):
    global x, y, v, passage, pas, AffichageIng, AffichagePri, yFut, xFut, case, s, PersoDro, PersoDos, PersoGau, PersoHaut, ImPersoHaut

    if AffichageIng or AffichagePri: # lorsqu'on est dans le menu en jeu
        return ()

    xFut = x
    yFut = y - v # yFut est le coordonnée en y que le perso devrait avoir apres son déplacement
    CasePerso()

    if case in s: # si la futur case du perso est dans la liste des cases bloqués alors on sort de la fonction pour que le perso
        return () # n'avance pas

    if x == 585 and y == 300: # on test si le perso est sur l'avant derniere case verticale lorsqu'il appuie pour avancer mais avant
        passage = 1 # que les coord aient avancés

    if passage == 1 and x == 585 and y == 228 and pas > 0: # si passage est à 1, que le perso est sur la derniere case et qu'on appuie
        InitPerso() # pour monter alors on remet au depart pour la salle d'apres
    else:
        if y - v > 188 and pas > 0: # tant que le perso ne sort pas du cadre en hauteur, on le fait avancer case par case
            y = y - v # (variable v pour la taille de l'avancement)

        try: # on essaie d'enlever toutes les images orientés et seulement celle du dernier déplacement est possible. on devrait
            can.delete(PersoDos) # donc avoir NameError pour les autres images
        except NameError: # on excepte NameError et ainsi le programme ne fait pas d'erreur
            rien = 0

        try:
            can.delete(PersoGau)
        except NameError:
            rien = 0

        try:
            can.delete(PersoDro)
        except NameError:
            rien = 0

        ImPersoHaut = PhotoImage(file='Images/haut.gif')
        PersoHaut = can.create_image(x, y, image=ImPersoHaut)
        can.coords(PersoHaut, x, y)

        print("haut ", "x= ", x, "y = ", y)
        DecompPas()

    if pas == 0:
        MenuIng()
```

7.

```
def CasePerso():
    global yFut, xFut, case

    case = -1

    for j in range(7):
        for i in range(13):
            if yFut == 228 + j * 72 and xFut == 153 + i * 72:
                case = i + j * 13
```

```

def InitBlocage(): # on lance InitBlocage a chaque entrée dans une salle pour initialiser les cases bloqués
    global salle, xBloc, yBloc, bloc1, s

    LBloc = [0] * 91 # initialisation des 90 cases sans blocage

    s = []

    s1 = [3, 15, 27, 30, 31, 32, 33, 34, 43, 47, 56, 57, 58, 59, 60, 64, 76, 77, 88, 89, 90] # index des cases bloqués en salle 1
    s2 = [16, 19, 20, 21, 24, 26, 27, 29, 31, 43, 47, 51, 52, 58, 60, 61, 62, 67, 69, 70]
    s3 = [15, 19, 20, 21, 29, 30, 31, 36, 37, 38, 45, 46, 47, 53, 54, 55, 60, 61, 62, 71, 72, 73, 81, 82]
    s4 = [0, 3, 8, 12, 18, 26, 29, 33, 36, 44, 50, 54, 58, 69, 73, 76, 78, 80, 90]
    s5 = [5, 8, 15, 16, 24, 27, 31, 32, 33, 35, 36, 40, 42, 43, 44, 47, 62, 66, 67, 68, 71, 72, 73, 75, 83]
    s6 = [1, 14, 16, 17, 18, 21, 22, 30, 33, 36, 37, 38, 41, 42, 43, 45, 46, 47, 56, 62, 63, 67, 69, 70, 71, 75, 76]
    s7 = []
    s8 = [14, 15, 16, 18, 20, 21, 22, 23, 30, 32, 34, 36, 37, 40, 48, 49, 54, 56, 59, 61, 63, 67, 68, 70, 72, 79, 83, 85]
    s9 = [3, 4, 5, 7, 14, 16, 22, 27, 34, 35, 36, 40, 42, 43, 44, 45, 46, 50, 53, 54, 61, 69, 70, 72, 73, 76, 80, 88, 89]

    if salle == 1:
        s = s1
    if salle == 2:
        s = s2
    if salle == 3:
        s = s3
    if salle == 4:
        s = s4
    if salle == 5:
        s = s5
    if salle == 6:
        s = s6
    if salle == 7:
        for i in range(27):
            x = randint(0, 90)
            if x != 6 and x != 84:
                s7.append(x)
        s = s7
    if salle == 8:
        s = s8
    if salle == 9:
        s = s9

    s.sort() #permet de mettre la liste dans l'ordre croissant

    xBloc = 0
    yBloc = 0

    LxBloc = []
    LyBloc = []

    for i in range(len(s)): # placement de images du blocage
        a = s[i] # a change à chaque tour dans la boucle et est égale à la valeur de s à l'indice i
        LBloc[a] = 1

        if a <= 12: # première ligne
            yBloc = 242

        if a >= 13 and a <= 26: # deuxième ligne
            yBloc = 314

        if a >= 27 and a <= 39:
            yBloc = 386

        if a >= 40 and a <= 52:
            yBloc = 458

        if a >= 53 and a <= 65:
            yBloc = 530

        if a >= 66 and a <= 78:
            yBloc = 602

        if a >= 79 and a <= 91:
            yBloc = 674

        for j in range(13): #pour les colonnes
            if (a % 13) == j:
                xBloc = 155 + j*72

        bloc1 = can.create_image(xBloc, yBloc, image=ImBloc1)

        LxBloc.append(xBloc)
        LyBloc.append(yBloc)

    print(LxBloc)
    print(LyBloc)

```