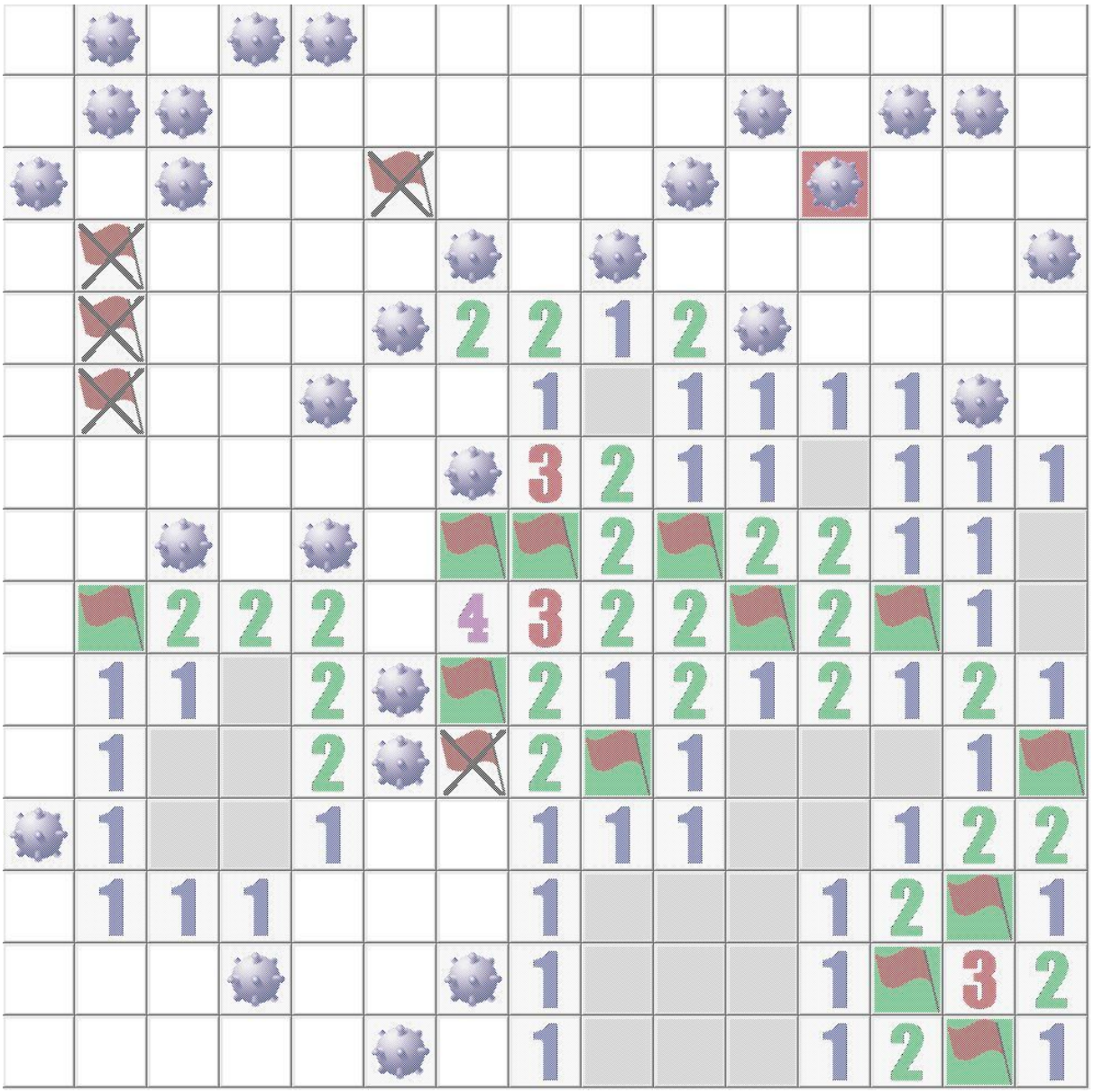
2018-2019 Terminales
S

ISN D-MINEUR



Matéo Boukhobza - Gradus Raphael - Leroy Aristide

Sommaire :

Présentation du projet :

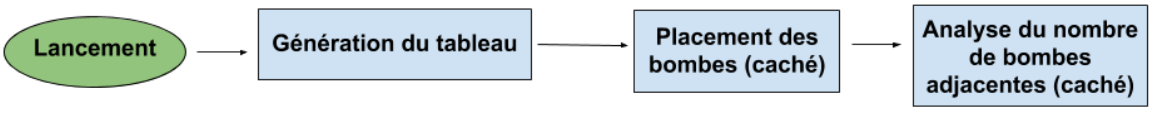
Pourquoi le démineur ?

Ce choix ne s'est pas fait directement. En effet, nous avions au début commencé à essayer de faire un résolveur de rubik's cube. Cependant, au bout de deux semaines, notre difficulté à résoudre un rubik's cube nous avons décidé de changer et opter pour faire un démineur, un projet auquel nous avions déjà pensé auparavant.

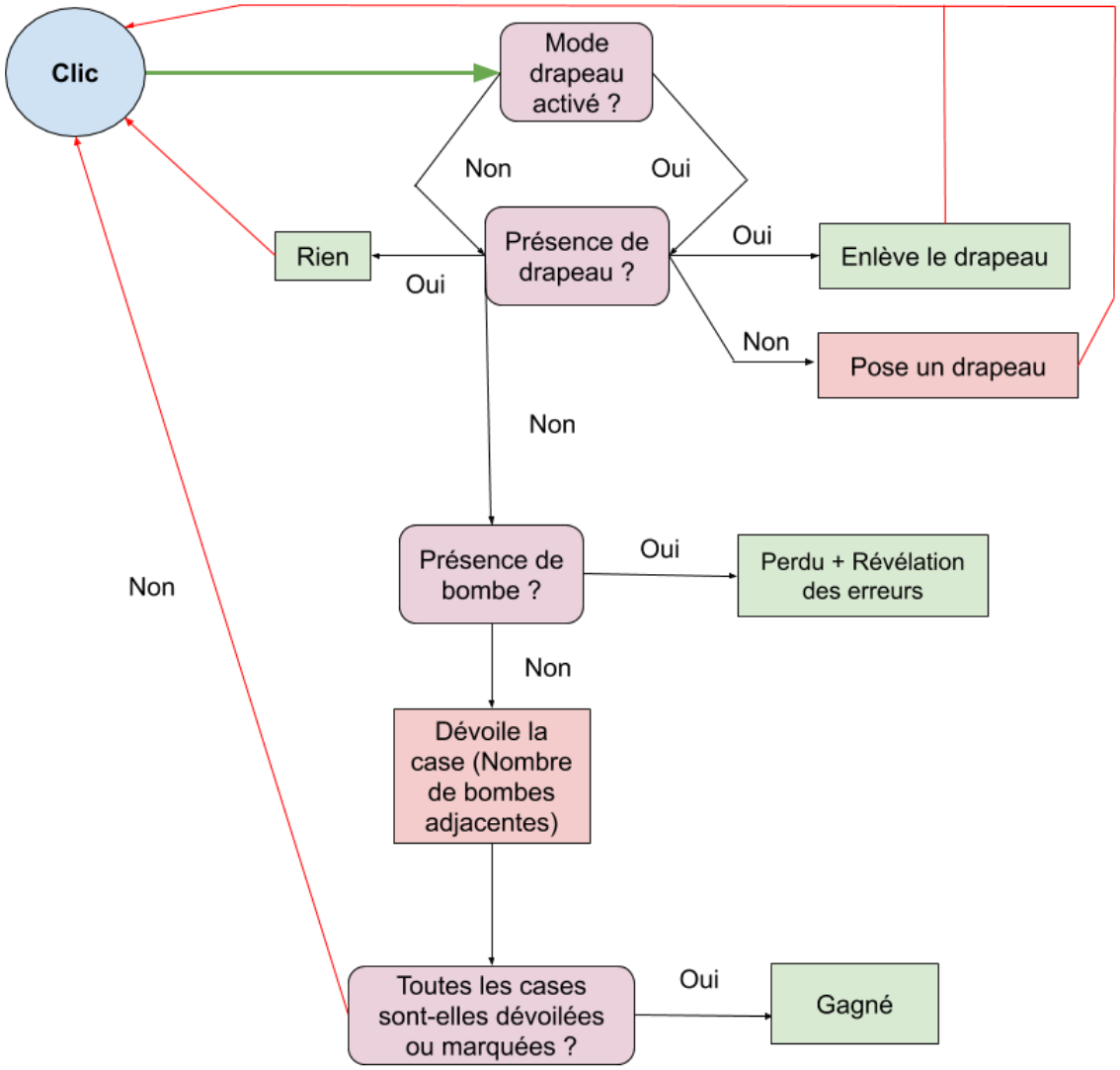
Le démineur est un jeu classique, iconique et indémodable auquel nous avions tous déjà joué auparavant, c'est pourquoi on s'est dit que c'était probablement le projet dans lequel nous devions nous lancer.

Quel démineur créer pendant ce projet ?

Nous voulions à tout prix créer un démineur classique tout en y ajoutant une caractéristique supplémentaire qui le démarquera des autres démineurs existants.. En effet, nous voulions apporter à notre démineur une capacité de solveur automatique qui montrerait à l'utilisateur quel mouvement faire à quel moment et pourquoi les faires.
Notre programme devait fonctionner en deux parties distinctes : l'initialisation et le fonctionnement en jeu.
Voici deux représentation schématique du projet que nous voulons réaliser :



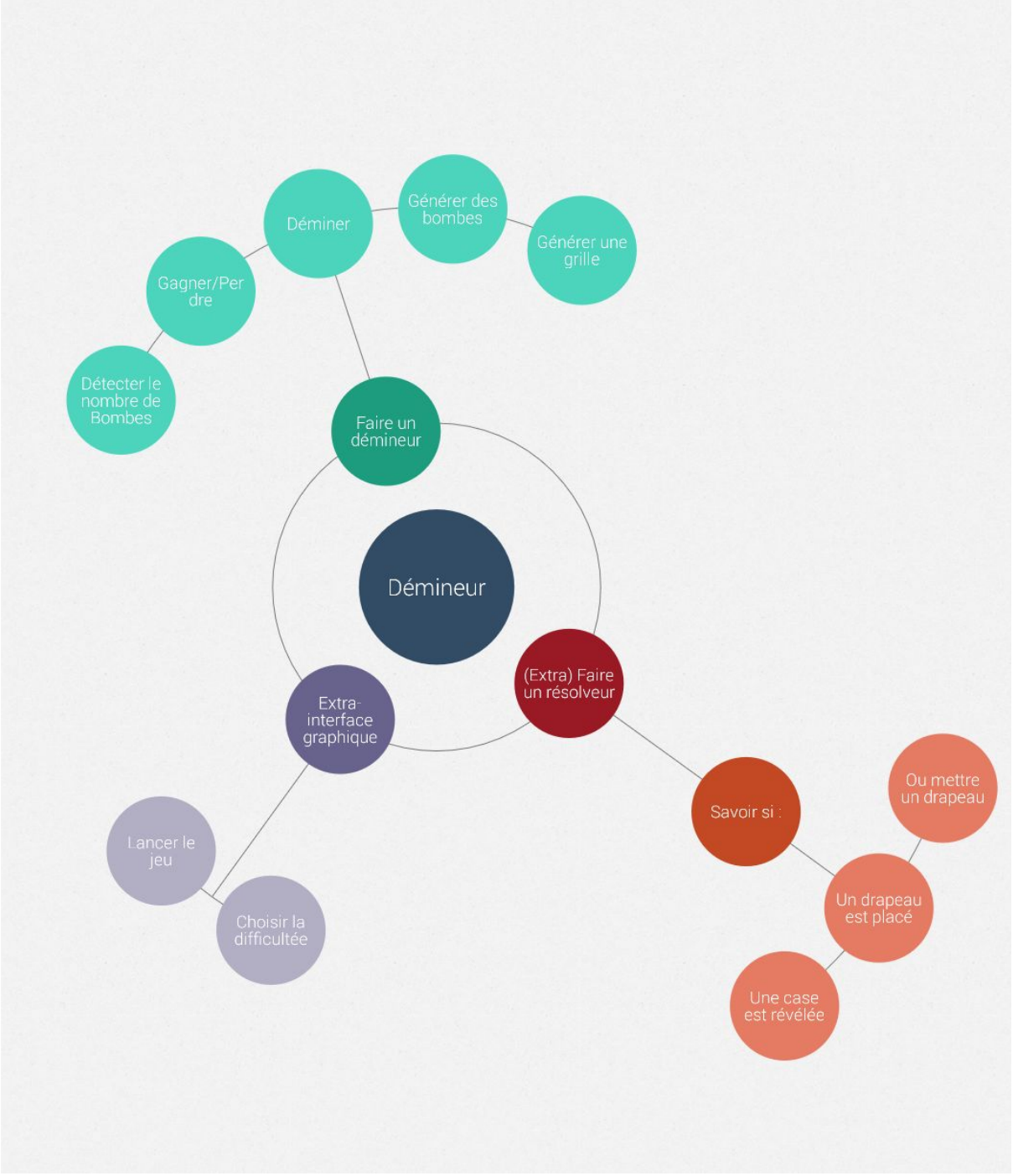
Initialisation



Fonctionnement de l'algorithme "Clic"

Cahier des charges :

Une fois notre projet choisi et après avoir réfléchi en équipe sur ce que notre démineur devait accueillir comme fonctionnalité pour fonctionner correctement, nous sommes arrivés à en déduire un cahier des charges que nous avons complété au fur et à mesure de l'avancé du projet afin de le perfectionner au maximum.
Nous sommes finalement arrivés à déterminer le cahier des charges ci-dessous :



Ce cahier des charges satisfait parfaitement nos ambitions pour le projet que étions sur le point de débuter.

Partie collaborative :

Avant le début de notre projet, nous nous sommes dispersés le travail à faire. Au fur et à mesure que le projet avançait, nous avons été amenés à nous compléter les uns les autres afin que chacun apporte au groupe ce qu'il savait pour réaliser au mieux le projet et permettre au groupe d'avancer plus rapidement.

Voici le tableau récapitulatif de ce que chacun a réalisé pendant le projet :

Raphael	Aristide	Matéo
Entretien du cahier des charges du projet Que va faire notre démineur et comment le faire.	**< février** Découverte et compréhension de Tkinter, différents outils, commandes et objets proposés	**7 février 2019** Création de la grille
Recherches et compréhensions des différents outils de Tkinter et les attributs des objets de tkinter (aide aux membres de l'équipe)	Disposition des images dans un tableau	**14 -> 17 février 2019** Génération des bombes
Commentaires des fonctions et amélioration de la compréhension de ces dernières	Résolution des différents bugs sur le calcul du nombre de bombes adjacentes	**v1 : 16-17 février 2019** **v2 : 23-24 février 2019** Générations du détecteur de nombre de bombes adjacentes
Détection des bugs et solution de ces derniers (problèmes des cases adjacentes)	Résolution du problème des bombes qui se superposent à la génération du tableau	Semaine avant vacances de fin mars : Récupérer les informations lorsqu'on clique sur une case avec Aristide
Réalisation de l'interface graphique (menu), inachevée	Résolution du problème : Comment notre programme fait-il pour savoir sur quelle case cliquons-nous ?	**21 Mars - 27 Mars** Détection des cases sans bombes autour. Partie Résolveur (extra - que l'on a pas eu le temps de finir)

Résolution du problème : Comment rassembler tout les boutons au centre de l'écran (frame)	Développement de la commande de clic	Résolution de divers bug
Début de résolution du problème : Comment faire démarrer le programme principal à partir du programme du menu.	Développement du "Mode drapeau" pour pouvoir poser des drapeaux sur les cases susceptibles de contenir des bombes	Détection des premières bombes par le résolveur

Pour le travail en groupe et pour échanger plus facilement, nous avons beaucoup utilisé la plateforme en ligne Github pour rester à jour dans nos tâches. Nous avons également utiliser différents outils collaboratifs comme les outils de Google : Drive pour partager des dossiers et y accéder en temps voulu, Docs pour faire du traitement de texte à plusieurs et en simultanée. De plus, pour communiquer plus simplement nous avons créé un groupe messenger.
Enfin, pour échanger nos nouveautées dans les programmes, il nous arrivait d'utiliser la messagerie de l'établissement : IAL.

Réalisations personnelles :

Durant ce projet notre équipe a eu l'occasion d'élaborer beaucoup de fonctions : des fonctions obsolètes, les fonctions de débogage et les fonctions actuelles.

I- Démineur

a- présentation :

Le démineur est un jeu dans lequel on doit trouver des mines disséminés dans une grille. Lorsque l'on clique sur une case on a la connaissance du nombre de mines autour. Le but est de trouver toutes les mines en posant un drapeau dessus. Si on clique sur une mine, on perd.

Notre D-mineur est composé d'une suite de boutons (Tkinter) disposés sous forme d'une grille.
Chaques bouton est un objet caractérisé par 2 attributs :

- Le type de case, c'est à dire; est-ce que la case est une Bombe, ou le nombre de bombes adjacentes à la case.

b- fonctions du démineur :

Durant ce projet j'ai réalisé de nombreuses fonctions servant à générer le démineur.

- attribut des cases (partie orientée objet) (class Case)
- génération de la grille
- génération des numéros de bombes adjacentes (DetectBombe)
- base de la gestion des exceptions (Si la case est sur la bordure de la grille)

Extra (Résolveur):

- Détection des cases sans bombes autour d'une case
- Action lors du premier clic
- Affichage des cases autour de la première case cliquée sans bombe.
- Détection des coins exemple ci dessous

0	0	1
2	1	2
3	2	

Exemple d'un coin parmis d'autres :

- Affichage des cases autour d'un case de type 1 autour d'un drapeau.

Voici quelques difficultés auxquels j'ai été confronté. Dans le fichier "fonctions obsolètes" de github se trouve la première version de SetBombe(). Au début elle ne fonctionnait pas, jusqu'à ce que je prenne le temps de la poser sur papier et que je passe par le pseudo-code.
Dans le première version j'ai essayé de faire un système de placement de bombe grâce à un système de coordonnées. *Le code suivant est présenté à titre indicatif. Aucune explication n'est donnée car la fonction était obsolète et ne fonctionnait pas. Elle est présente dans le fichier "fonctions obsolètes" de la ligne 19 à 41 (22 lignes).*

```
19  def setBombe(nbBombe):
20
21      for tailleBombe in range(nbBombe):
22
23          Abscisse = randint(0,9)
24          print(Abscisse,";",end="")
25          Ordonnee = randint(0,9)
26          print(Ordonnee)
27          compteur = 0
28          Result = 0
29          while Bse_donnee[Result].ord != Ordonnee and Bse_donnee[Result].abse != Abscisse:
30              if (Bse_donnee[Result].abse == Abscisse) and (Bse_donnee[Result].ord != Ordonnee):
31
32                  Result += 10
33              elif Bse_donnee[Result].ord == Ordonnee and Bse_donnee[Result].abse != Abscisse:
34
35                  Result += 1
36              else:
37                  Result += 1
38              Bse_donnee[Result] = Case(Result, Abscisse, Ordonnee, "Bombe")
39              Button(fenetre, text="Bombe", width=8,height=4).grid(row=Abscisse, column=Ordonnee)
40              print(Bse_donnee[Result].type)
41              break
```

Voici la fonction actuelle (8 lignes).

```
19  def SETBOMBE(nombredebombes):
20
21  #Fonction plaçant les bombes dans la grille
22  #La liste d'interdits est une liste contenant les cases où les bombe
23      x = choice(ListeCaseNonBombe)
24      Bse_Bombe.append(x)
25      ListeCaseNonBombe.remove(x)
26      Bse_donnee[x].type = "Bombe"
27
```

La fonction est bien plus courte. On prend une valeur aléatoire de la liste des cases n'étant pas une bombe. On la retire de la liste et on change son type en bombe.
Elle est plus courte et plus simple. Nous avons été confrontés à ce problème car nous n'avons pas bien pensé le programme. La preuve, au début, nous voulions utiliser des coordonnées. Elles n'ont jamais servie.

Nous avons pris beaucoup de temps à résoudre un autre problème. Lors de la génération des Bombes, à des moments, il n'y avait pas les bons numéros autour des Bombes. Tout était censé fonctionner, pourtant le problème persistait. On avait un problème du type :

1	Bombe	3
0	4	Bombe x2
1	Bombe	3

Le problème persistait: les bombes se généraient plusieurs fois au même endroit.

La plus grande difficulté dans cette partie de ce projet a été de savoir sur quelle case clique le joueur.
En effet, nous n'arrivions pas à savoir sur quelle case clique le joueur.
Au final, un de mes coéquipier a réussi à résoudre le problème en utilisant la

fonction lambda, grâce à l’aide d’un de mes coéquipiers.

Une troisième difficultée a été de créer la fonction near() dans le fichier fonctions obsolètes.
Cette fonction buguait souvent. Cette fonction avait pour but de générer les numéros de cases. La fonction traitait d’abord les exceptions (cases sur le bord de la grille).
Le principe de la fonction est : pour chaque case compter le nombre de bombes autour pour assignér le type de chaques cases.

```
46  def near():
47      for case in range(len(Bse_donnee)):
48
49          print(case)
50          if Bse_donnee[case].compteur == 0:
51              if Bse_donnee[case+1].type == "Bombe":
52                  Bse_donnee[case].type += 1
53
54              if Bse_donnee[case+11].type == "Bombe"
55                  Bse_donnee[case].type += 1
56              if Bse_donnee[case+10].type == "Bombe"
57                  Bse_donnee[case].type += 1
58              Bse_Case[case].config(text=Bse_donnee[
59          elif Bse_donnee[case].compteur == 9 :
60              if Bse_donnee[case-1].type == "Bombe":
61                  Bse_donnee[case].type += 1
62              if Bse_donnee[case+9].type == "Bombe":
63                  Bse_donnee[case].type += 1
64              if Bse_donnee[case+10].type == "Bombe"
65                  Bse_donnee[case].type += 1
66              Bse_Case[case].config(text=Bse_donnee[
67          elif Bse_donnee[case].compteur == 99:
68              if Bse_donnee[case-1].type == "Bombe":
69                  Bse_donnee[case].type += 1
70              if Bse_donnee[case-11].type=="Bombe":
71                  Bse_donnee[case].type += 1
72              if Bse_donnee[case-10].type=="Bombe":
73                  Bse_donnee[case].type += 1
74              Bse_Case[case].config(text=Bse_donnee[
75          elif Bse_donnee[case].compteur == 90:
76              if Bse_donnee[case-9].type == "Bombe"
77                  Bse_donnee[case].type += 1
78              if Bse_donnee[case+11].type=="Bombe" :
```

Voici l'actuelle fonction near() servant à détecter les bombes :

```
92  def DetectBombe():
93  #Fonction déterminant le nombre de bombes adjacentes (en appelant les
94      global NBBOMBES
95      for loop in range(NBBOMBES):
96          Bombe = Bse_Bombe[loop]
97          if Bse_Bombe[loop] in LINEDOWN:
98              lineDown(Bombe)
99          elif Bse_Bombe[loop] in LINEUP:
100             lineUp(Bombe)
101         elif Bse_Bombe[loop] in LINERIGHT:
102             columnRight(Bombe)
103         elif Bse_Bombe[loop] in LINELEFT:
104             columnLeft(Bombe)
105         elif Bse_Bombe[loop] == CORNERRIGHTU:
106             cornerRightU(Bombe)
107         elif Bse_Bombe[loop] == CORNERLEFTD:
108             cornerLeftD(Bombe)
109         elif Bse_Bombe[loop] == CORNERRIGHTD:
110             cornerRightD(Bombe)
111         elif Bse_Bombe[loop] == CORNERLEFTU:
112             cornerLeftU(Bombe)
113         else:
114             ActualiserUneCaseAutourDUneBombe(Bombe,+1)
115             ActualiserUneCaseAutourDUneBombe(Bombe,-1)
116             ActualiserUneCaseAutourDUneBombe(Bombe,+LargeurGrille)
117             ActualiserUneCaseAutourDUneBombe(Bombe,-LargeurGrille)
118             ActualiserUneCaseAutourDUneBombe(Bombe,-LargeurGrille-1)
119             ActualiserUneCaseAutourDUneBombe(Bombe,+LargeurGrille+1)
120             ActualiserUneCaseAutourDUneBombe(Bombe,-LargeurGrille+1)
121             ActualiserUneCaseAutourDUneBombe(Bombe,+LargeurGrille-1)
```

On peut voir qu'elle est beaucoup plus courte.

Pendant que mes coéquipiers s'occupaient de finir le démineur j'ai commencé à faire le résolveur. Même, si le résolveur a beaucoup de bugs, et qu'il ne résout pas entièrement le démineur il arrive à détecter des coins et à révéler toutes les cases sans bombes autour.
Dans la partie qui va suivre, je présenterai le fonctionnement du Résolveur sous la forme d'un schéma.

II- Résolveur

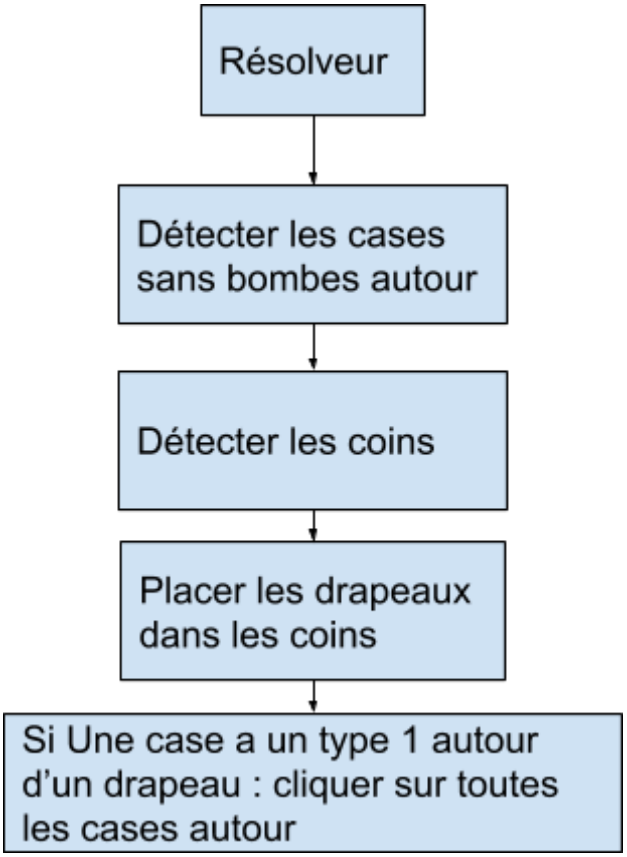
Le choix de présenter le Résolveur sous la forme de schéma me permet de rendre compte de cette partie bien que cette fonction ne soit pas terminée. En effet il est composé d'environ 200 lignes qui serait trop long à développer dans le temps imparti. *Le résolveur est inclu à titre démonstratif dans la partie annexe.*

Si le résolveur n'a pas été fini dans les temps, c'est parce que dans le code il y avait beaucoup de bug et que j'utilisais parfois les mauvaises solutions techniques à un certain problème donné. Par exemple dans ClicCase j'avais créé un dictionnaire pour stocker les cases révélées :
Dictionnaire[cases] = type
alors que une liste pour chaque type aurait suffit.
De plus une liste dans une fonction peut être sauvegadée hors de la fonction grâce au Liste.append(valeur).
Au lieu d'avoir un dictionnaire DictCasesRévélées = {Case1: type, Case2,type} on aurait :
Si le type de la case = 0:
    Liste0.append(case)
3 lignes de la fonction Resolve dans "Résolveur buggé non fini sans explications" qui devraient être remplacé par les lignes ci-dessus.

```
210                     Bse_donnee[x].etat = "révélée"
211                     ListeCasesRévélées.append(x)
212                     Dico[x] = Bse_donnee[x].type #inutile, une list
213                     if Bse_donnee[x].type == 0:
```

La difficulté à été dû à un manque de connaissance de ma part. En effet, j'avais oublié que les listes dans une fonction ne se sauvegardent pas.

La deuxième difficulté à laquelle j'ai été confronté a été de ne pas réussir à créer une condition avec un liste augmentant au fur et à mesure que la condition existe. Quand j'ai appris comment fonctionne la récursivité il était trop tard.

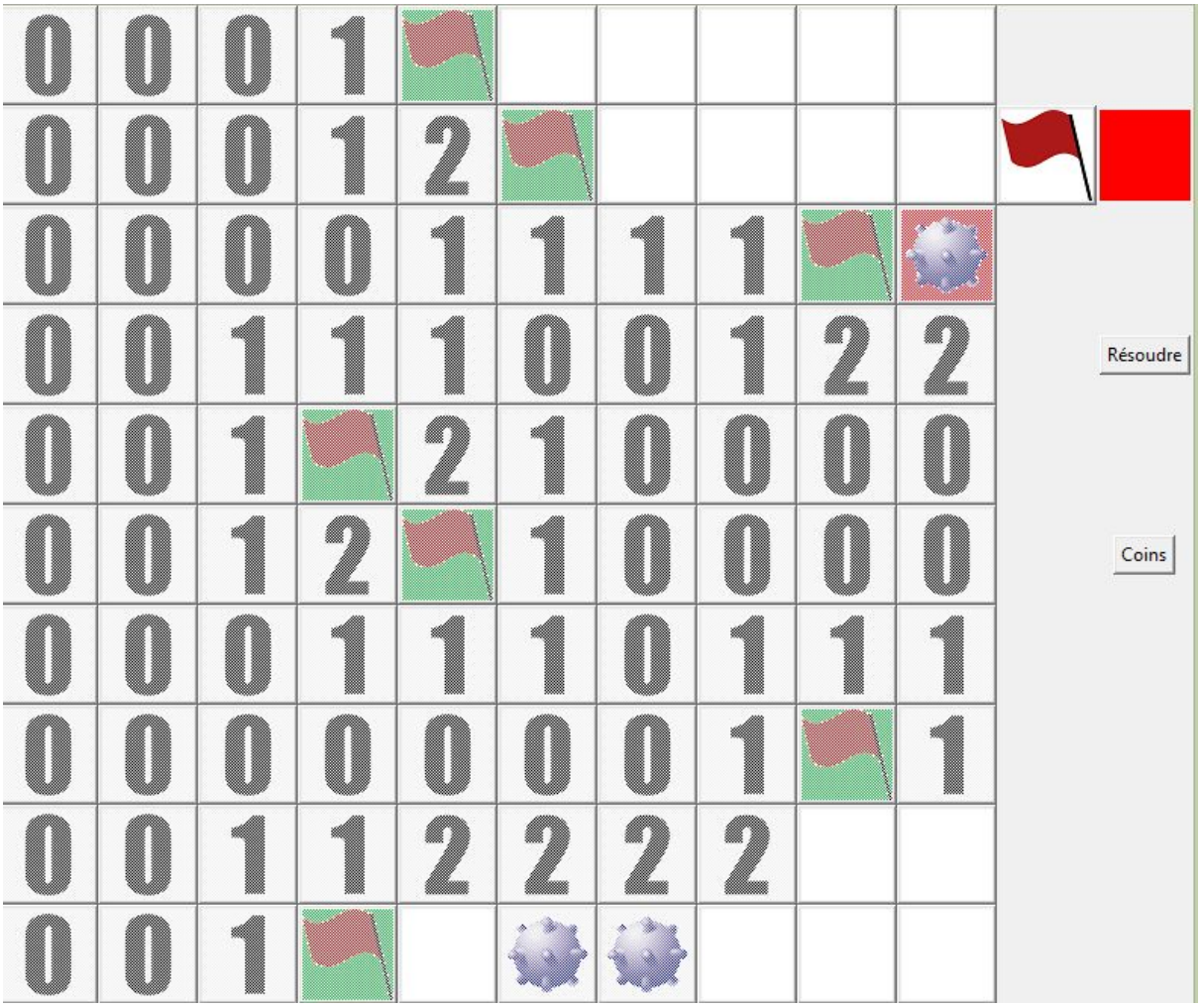

Résolveur
Détecter les cases sans bombes autour
Détecter les coins
Placer les drapeaux dans les coins
Si Une case a un type 1 autour d'un drapeau : cliquer sur toutes les cases autour

Résultat final :

Voici ma partie fonctionnelle lorsque je demande au programme de m'afficher le type de chaque case. On peut voir que tout fonctionne. La génération des bombes est bonne et les numéros le sont également.

0	0	1	1	1	0	0	0	0	0
1	1	2	Bombe	1	0	0	0	0	0
1	Bombe	2	1	1	0	0	1	1	1
1	1	1	0	1	1	2	2	Bombe	1
1	1	0	0	1	Bombe	2	Bombe	3	2
Bombe	1	0	0	1	1	2	2	3	Bombe
1	1	0	0	0	0	1	3	Bombe	3
0	0	0	0	0	0	1	Bombe	Bombe	2
0	0	0	0	0	0	1	2	2	1
0	0	0	0	0	0	0	0	0	0

Ensuite voici le résultat final lorsque toutes les parties sont intégrées:



C’est ma version finale. Cette partie du démineur a été résolue par le résolveur. Même s’il a perdu, à cause d’un bug, on peut voir que le démineur fonctionne car :

- le démineur nous indique lorsque l’on a perdu,
- nous indique quels drapeaux sont bons
- nous indique quel drapeau est mauvais
- nous indique où sont les bombes.

On constate que le démineur respecte le cahier des charges.

Perspectives futures

Voici quelques points que serait sans doutes amené à reprendre une potentielle future équipe si elle devait reprendre le projet :

- Elle devrait d'abord rendre l'interface graphique parfaitement fonctionnelle (lancer le programme lorsqu'on appuie sur jouer, modifier le niveau de difficulté lorsqu'on sélectionne un niveau etc…)
- Elle devrait développer l'effacement de toutes les cases vides adjacentes lors d'un clic sur une case vide, dans le but de se rapprocher du vrai démineur et d'offrir une expérience de jeu plus agréable.
- Elle devrait aussi sans doute améliorer le design du programme. Pourquoi pas donner une meilleur couleur de fond que le blanc, mieux intégrer le "menu drapeau" dans la fenêtre etc…

Enfin, elle pourrait aussi adapter le programme à différentes taille d'écrans et pourquoi pas aussi développer une application pour smartphones.

Faire un Résolveur fonctionnel

Bilan personnel :

Durant ce projet, j'ai été confronté à plusieurs difficultées:

- travailler en groupe lorsque la motivation n'est pas la même pour chaque personne
- Trouver une idée de projet consensuel : Le projet initial n'a pas donné suite. Nous avons dû réfléchir à un autre projet qui fasse l'unanimité.
- Expliquer à mes camarades ce que j'ai fait

Ce projet m'a aussi permis :

- de découvrir ce qu'est un projet de programmation
- d'améliorer mon travail en groupe.

D'être confronté à une deadline en programmation.

- M'a permis, de voir ce qu'est de travailler en groupe sur le long terme.
- Même s'il y a eu beaucoup de bugs, les résoudre a été très gratifiant.

Annexes

Le programme est disponible dans sa dernière version (v3.1.2) sur notre page Github au lien suivant : https://github.com/lasource2019/D-mineur
Le résolveur est également disponible.

Matéo Boukhobza