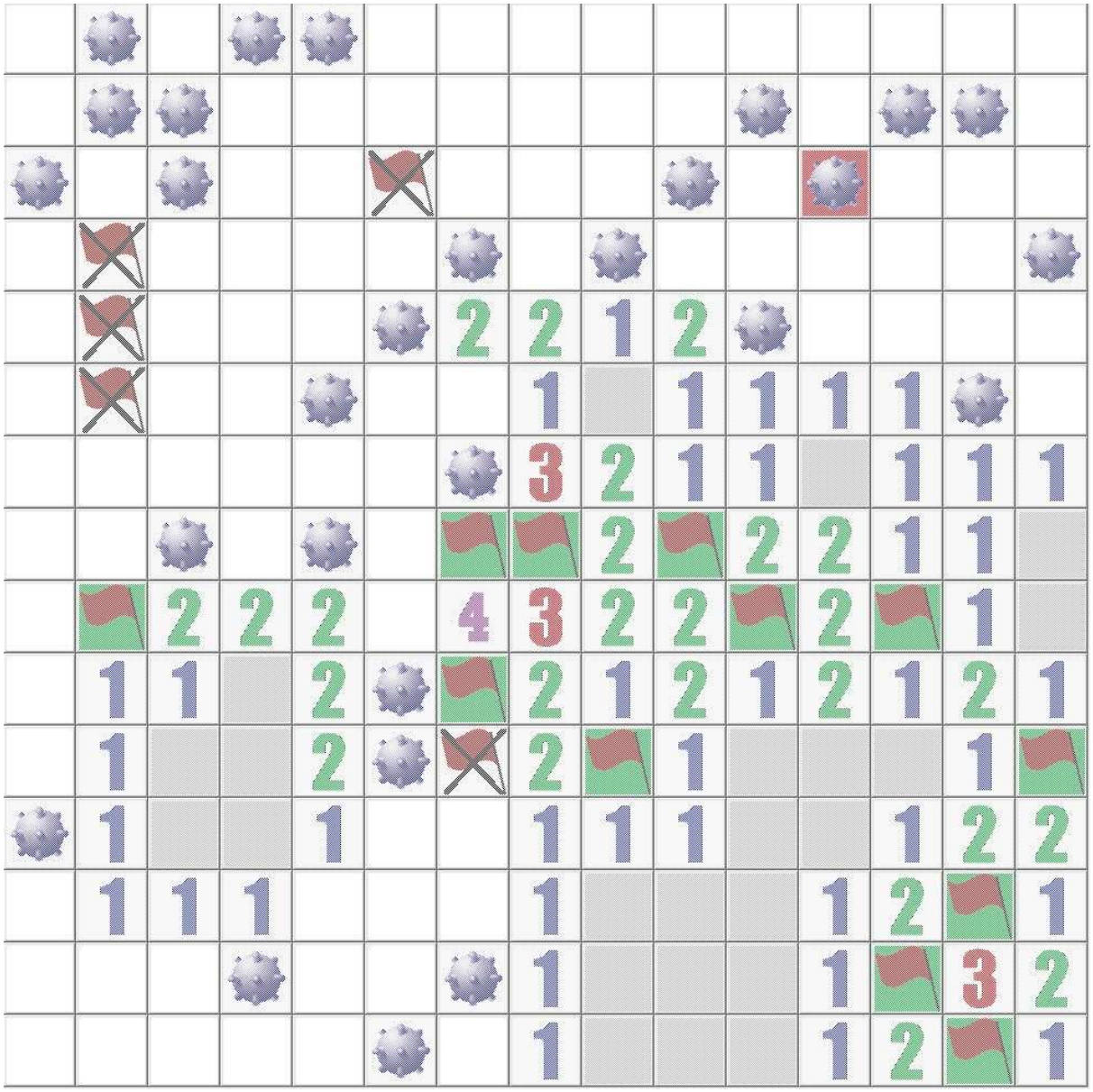
Matéo BOUKHOBZA, Raphaël GRADUS, **Aristide LEROY**

D-MINEUR



Sommaire :

Présentation du projet :

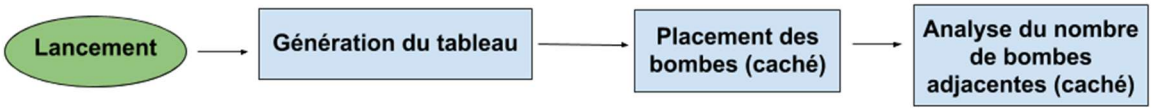
Pourquoi le démineur ?

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 faible motivation dans la réalisation de ce projet nous a amené à en changer et opter pour faire un démineur, un projet auquel nous avions déjà pensé auparavant.

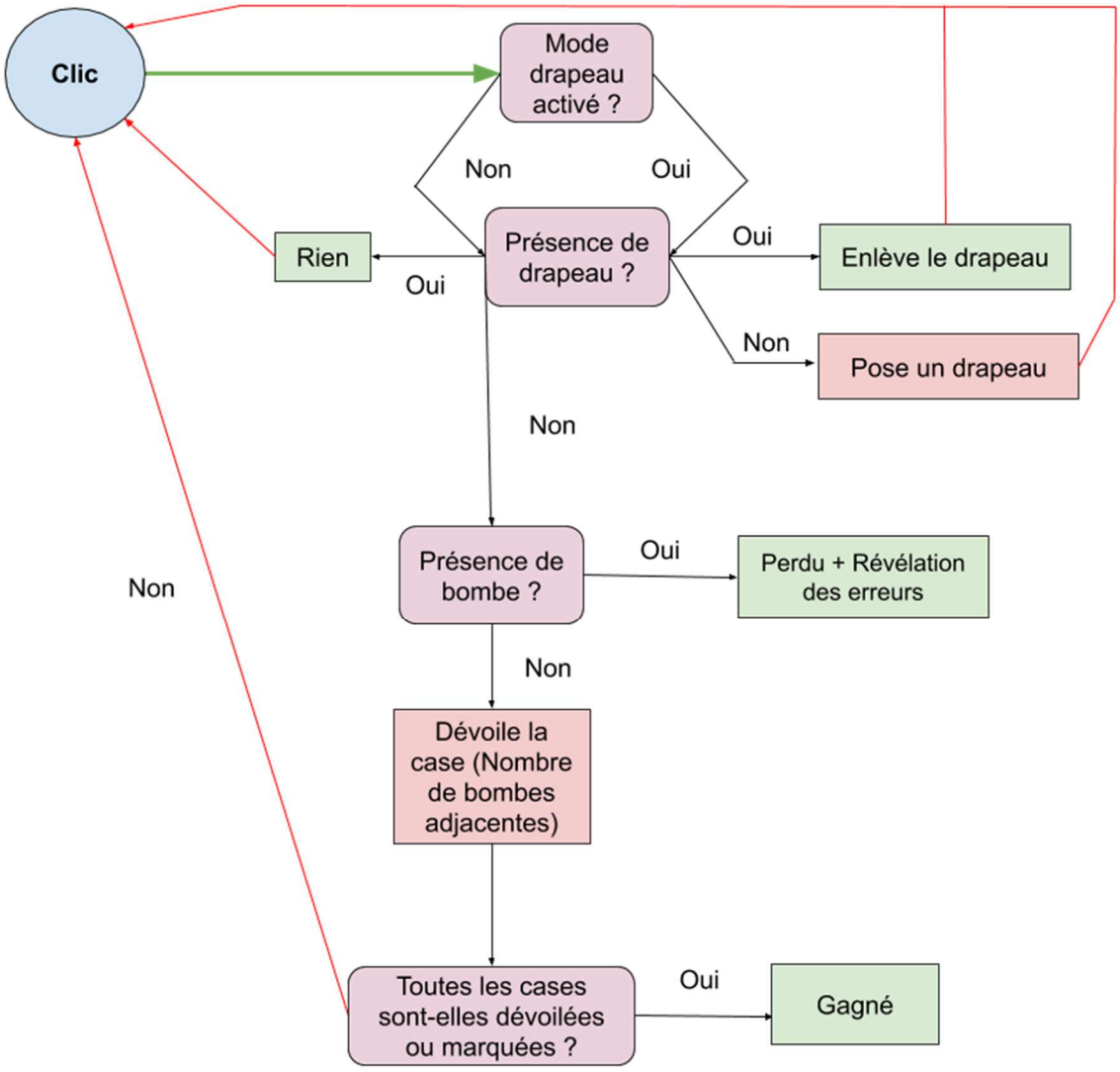
Le démineur est un jeu classique, iconique et indémodable auquel nous avions tous déjà joué auparavant, c'est pourquoi on s'est dit que c'était probablement le projet dans lequel nous devions nous lancer.

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 représentations schématiques du projet que nous voulons réaliser :



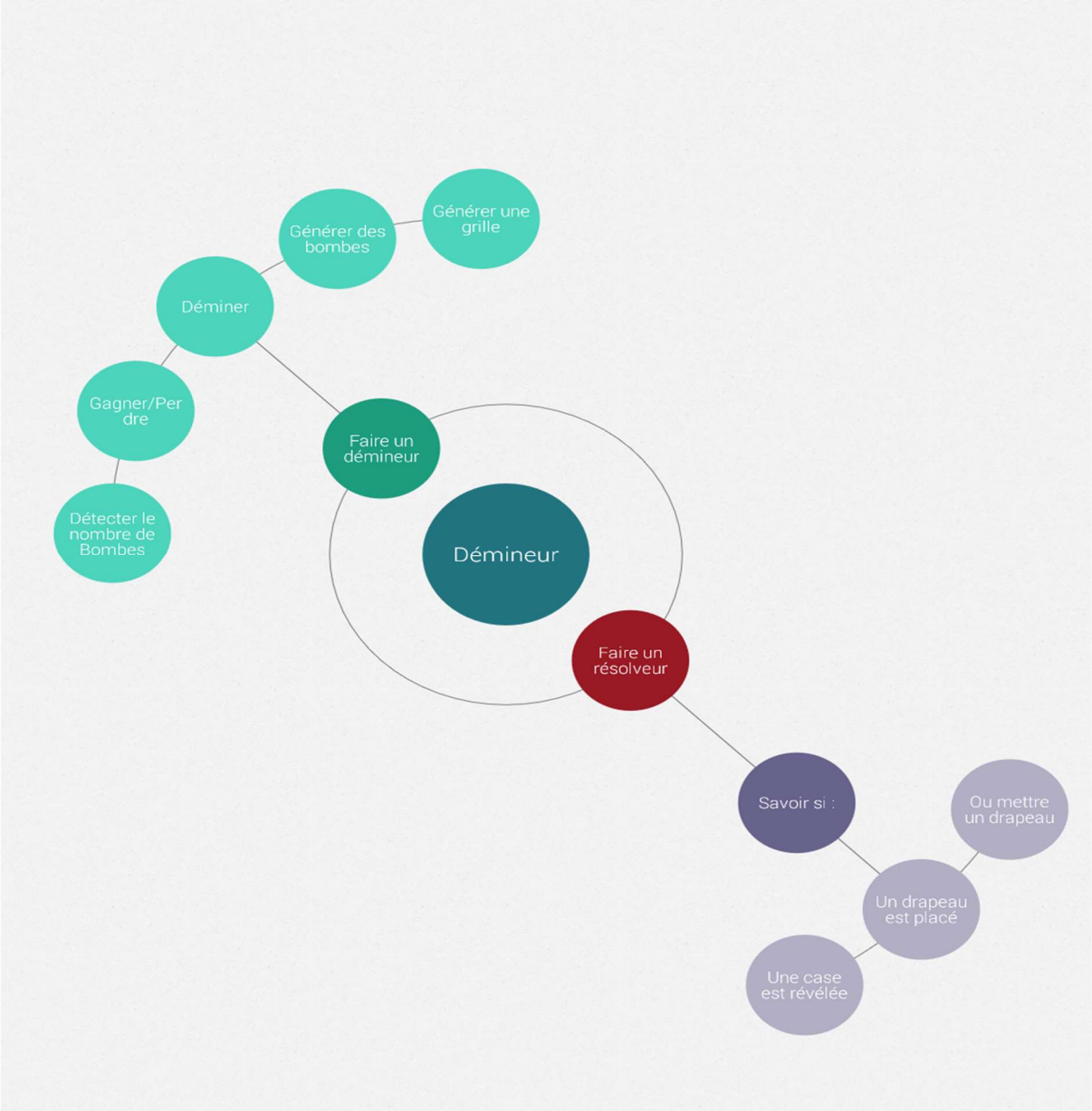
Initialisation



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'avancée 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 nous étions sur le point de débuter.

Partie collaborative :

Avant le début de notre projet, nous nous sommes réparti 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 n'a pas eu le temps de finir)
Résolution du problème : Comment rassembler tous 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 utilisé 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.
Enfin, pour échanger nos nouveautés dans les programmes, il nous arrivait d’utiliser la messagerie de l’établissement : IAL.

Réalisations personnelles :

J’ai d’abord travaillé sur la disposition des images dans les différents boutons du tableau.

Plusieurs outils de disposition graphique des boutons étaient disponibles. Il existe entre autre le .pack() , l’utilisation des coordonnées en pixels dans la fenêtre ou bien le .grid().

Le .pack() permet de choisir directement de quel côté de la fenêtre placer notre bouton dans mais c’est une méthode qui paraissait un peu brutale.
Les coordonnées en pixels est méthode très précise mais également compliquée à utiliser pour des boutons redondants. De plus, notre tableau étant sensé être modulable, cela aurait été un frein à notre projet.
Enfin, le .grid() semblait être la méthode adéquate car elle nous permettait de placer les boutons dans une grille virtuelle, régie par des coordonnées de tableau (lignes et colonnes). De plus, celle-ci est facilement modulable.

Avant de placer des images dans le tableau, nous placions ce que nous voulions afficher sous forme de texte et cela fonctionnait sans aucun souci. Cependant, lorsque nous avons voulu remplacer le texte par des images, nous n’y arrivions pas.
Ce qui m’a amené vers la solution est le fait qu’un bouton généré seul arrivait à afficher son image, mais que les boutons générés par boucle dans un tableau ne fonctionnaient pas. Après de multiple recherche, j’en suis arrivé à la conclusion que pour les boutons affichent leurs images en étant situés dans un tableau, il fallait que toutes ces images aient la même taille pour pouvoir être ordonnés.
Nous avons donc défini la taille de nos images à 60x60 pixels.

Pour savoir combien de bombes adjacentes chaque case possède, on pense logiquement que chaque case calcule combien de bombes sont adjacentes à elle-même. Mais nous avons préféré faire en sorte que chaque bombe indique aux cases adjacentes leur présence (cela permet entre autres d’effectuer moins de calcul).

Ainsi, une chaque bombe devrait dire aux 8 cases adjacentes qu'elle est présente. Cependant, nous avons dû traiter des cas à part. En effet, les lignes sur les bordures et les coins du tableau ne possèdent pas 8 cases adjacentes.
Nous nous y sommes donc pris de la manière suivante :

Nous avons d'abord défini des listes pour contenants les numéros des cases faisant partie de chaque cas à part. Par exemple, la liste de la ligne du haut correspond à :

```
LargeurGrille = int(input())
LINEUP = [ ]
for k in range(LargeurGrille - 2):
    LINEUP.append(1 + k)
```

Ensuite, il s'agit de créer une fonction pour rajouter "+1" à la valeur du nombre de bombes adjacentes à une case adjacente de celle contenant la bombe :

```
def ActualiserUneCaseAutourDUneBombe(NumeroCase, NombreAAjouter):

#Fonction servant plusieurs fois dans les fonctions suivantes
#Pour chaque case ayant une bombe, cette fonction ajoute "+1" aux cases adjacentes (pour la valeur du nombre de bombes adjacentes)

    if Bse_donnee[NumeroCase+NombreAAjouter].type != "Bombe":
        Bse_donnee[NumeroCase+NombreAAjouter].type += 1
```

Cette fonction est utilisée plusieurs fois pour indiquer la présence d'une bombe aux cases adjacentes dans notre cas spécifique qu'est la ligne du haut :

```
def lineUp(case):
    if case in LINEUP:
        ActualiserUneCaseAutourDUneBombe(case,+LargeurGrille)
        ActualiserUneCaseAutourDUneBombe(case,+1)
        ActualiserUneCaseAutourDUneBombe(case,-1)
        ActualiserUneCaseAutourDUneBombe(case,+LargeurGrille+1)
        ActualiserUneCaseAutourDUneBombe(case,+LargeurGrille-1)
```

Enfin dans la fonction DetectBombe() qui indique le nombre de bombes adjacentes à chaque case, on va regarder pour chaque case contenant une bombe si elle fait partie d'un ensemble spécifique (la ligne du haut ici), et si oui, nous faisons appel à la fonction précédente en utilisant comme paramètre "case" le numéro de la case dans laquelle se trouve la bombe.

À partir de ces informations, j'ai lancé plusieurs fois le programme pour tenter d'avoir une idée de quel(s) cas particulier(s) les bugs provenaient, et j'ai dû passer en revue plusieurs fois les fonctions des différents cas particulier pour résoudre les problèmes de calculs des numéros des cases adjacentes.

Lors de la génération des bombes dans le tableau, nous utilisons une boucle qui va prendre au hasard un nombre correspondant à une case du tableau pour y placer la bombe. Cependant, nous nous sommes rendu compte que les calculs des nombres de bombes adjacentes étaient mauvais dans certains cas. Mais en comptant le nombre de bombes présents dans la grille et en comparant ce nombre avec la valeur que nous avions indiqué, il y avait souvent une différence. Je me suis donc rendu compte que parfois, les bombes lors de leur génération se superposaient. En effet, la boucle n'interdisait pas à la prochaine bombe générée de retomber sur une case déjà occupée par une bombe. J'ai donc créé une liste d'interdits, contenant les numéros des cases déjà occupées par les bombes pour ne pas retomber dessus :

```
def SETBOMBE(nombredebombes):

#Fonction plaçant les bombes dans la grille
#La liste d'interdits est une liste contenant les cases où les bombes ont déjà été placées, pour éviter de les superposer
#x correspond au numéro de la case tirée au hasard

    for loop in range(nombredebombes):
        x = randint(0,((LargeurGrille*LargeurGrille) - 1))
        while x in Bse_Bombe:
            x = randint(0,((LargeurGrille*LargeurGrille) - 1))
        Bse_donnee[x].type = "Bombe"
        Bse_Bombe.append(x)
```

Lorsque nous commençons une partie, toutes les informations sont déjà présentes dans la mémoire du programme, les bombes sont placées et les calculs déjà effectués. Il ne manque plus qu'à afficher les informations lorsque l'on clique sur une case. Or, l'action effectuée par un bouton est la même que pour n'importe quel bouton, d'où la nécessité d'utiliser une fonction. Ainsi, cette fonction révèlerait les informations de la case sur laquelle on clique et serait donc en fonction du numéro de la case. Mais comment savoir sur quelle case nous cliquons ?
Dans les attributs de l'objet Bouton, il y a "command" qui renvoie à la fonction que l'on veut attribuer à ce bouton. En pseudo code, cela donnerait pour une fonction de clic :

command = Clic(NuméroDeLaCase)

Il faut savoir que le NuméroDeLaCase est une valeur qui évolue au fur et à mesure de la génération du tableau.
De plus, le numéro de la case est donné au moment de la génération à l'objet Case, mais ne l'est pas pour la commande. En réalité, la commande va enregistrer la commande "brute" mais sans les différentes valeurs qui vont avec. Ainsi, lorsqu'elle est sollicitée en cliquant sur le bouton, c'est là que la fonction Clic va se demander "Quelle est la valeur de NuméroDeLaCase ?". Mais cette valeur est déjà arrivée à la fin de la boucle et a pour valeur le numéro de la dernière case.
En fait, la valeur "NuméroDeLaCase" est prise en compte au moment où l'on clique sur le bouton, et non au moment où l'on créé ce bouton. Ainsi, il fallait réussir à enregistrer cette valeur pendant la création.
Après de longues heures de recherches, j'ai trouvé la fonction lambda
Si dans la commande, on écrit :

```
command = lambda NuméroDeLaCase=NuméroDeLaCase:Clic(NuméroDeLaCase)
```

C’est en quelques sortes une “sous-fonction”, qui s’écrit sur une ligne et qui est directement utilisée sur la même ligne de code.
Cela nous a permis de pouvoir utiliser une fonction générale clic() pour tous les boutons de notre tableau.

Pour rendre notre jeu fonctionnel, il fallait donc que l’on configure cette fonction de clic, celle-ci devait afficher la valeur de la case si l’on n’avait pas de bombes en dessous, et une bombe s’il y en avait une. En parallèle de cela, j’ai développé la fonctionnalité permettant de mettre des drapeaux sur les cases susceptibles de contenir des bombes. Le drapeau devait permettre de placer un repère visuel, mais aussi de ne pas dévoiler la case si l’on cliquait dessus par mégarde.
Ainsi, j’ai donc créé la possibilité de poser des drapeaux en créant un interrupteur. Il s’agit en fait d’un bouton accompagné d’un signal coloré indiquant si le “mode drapeau” est activé. Lorsque celui-ci est activé, la disposition ou le retrait des drapeaux est possible en cliquant. Dans le cas contraire, le clic permet de dévoiler la case.
La fonction clic est donc une succession de questions ou de conditions qui sont vérifiées et qui sont résumées dans le schéma page 4.
Il est important de noter que les conditions indiquant si la partie est perdue ou remportée sont inclue dans la fonction de clic.

Résultat final

Chaque étape de mon travail permet donc d'améliorer notre programme et de faciliter le travail des autres ainsi que l'utilisation même du programme :

- L'ajout d'images permet d'offrir une interface graphique intéressante et modulable. L'utilisation d'images toutes au même format permet une modulabilité facilitée, ainsi qu'une interface plus agréable à comprendre et une expérience utilisateur plus agréable. Cela permet également d'éviter les problèmes de disposition de boutons avec du texte ne possédant pas forcément les mêmes proportions que les autres.
  Pour tester cela, j'avais d'abord essayé de remplacer certaines images, mais cela n'a fonctionné que lorsque j'avais remplacé tous les boutons de texte avec des images. Cela nous a permis d'évoluer plus facilement par la suite

- La résolution des différents bugs dans le calcul des bombes adjacents était indispensable. Aucune case ne pouvait être épargnée car il est nécessaire de dévoiler la totalité des cases pour pouvoir remporter une partie. De même pour le bug de la superposition des bombes.
  Il s'agissait ici de lancer plusieurs fois le programme, d'essayer de résoudre le jeu et d'observer dans quels cas et où dans le tableau les erreurs de calculs se produisaient. Cela permettait d'identifier la source des erreurs et de les corriger.

- La découverte de la fonction lambda, permettant au programme de savoir sur quelle case nous cliquons, a elle aussi permis de rendre le jeu fonctionnel et de nous laisser évoluer sur le reste du programme. Sans celle-ci, nous étions bloqués et incapables d'avancer.

- La commande de clic est la source du fonctionnement du jeu. C'est grâce à celle-ci que nous pouvons évoluer dans notre partie, et elle contient toutes les conditions à tester pour chaque action que nous réalisons.

- Le mode drapeau n'était lui pas indispensable pour faire fonctionner le jeu, mais a permis de rendre le jeu plus intuitif et plus agréable dans son expérience d'utilisateur.

Perspectives

- Voici quelques points que serait sans doute 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 meilleure couleur de fond que le blanc, mieux intégrer le "menu drapeau" dans la fenêtre etc…

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

- Faire un résolveur fonctionnel

Bilan personnel

Je pense donc que ce projet nous a beaucoup appris. Il m'a appris ce qu'était le travail de groupe, comment se répartir le travail et puis le mettre en commun, tenir un cahier des charges et reporter nos avancées au fur et à mesure du projet. Il m'a également permis de découvrir plus en profondeur ce qu'était le "self-learning" puisque la majorité de ce que nous avons appris, nous l'avons fait par nous-même en apprenant et nous exerçant sur internet. Cela m'a fait me rendre compte d'autant plus que tout est disponible sur internet et que l'on peut toujours trouver des solutions grâce à cela, notamment grâce aux forums qui permettent de communiquer avec d'autres personnes ayant les connaissances que nous recherchons.

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