Marc Hu
François Thiébeauld de La Crouée

Démineur

1) Le cahier des charges du projet.

L'algorithme a pour objectif de gérer une partie de démineur. Il est placé dans dans le terrain un nombre de mine inférieur ou égal à 20% des cases. Le joueur doit indiquer la case qu'il veut ouvrir :

➡ S'il tombe sur une mine, alors le jeu s'arrête et le joueur a perdu.

➡ Sinon quand il tombe sur une case vide toutes les cases vides adjacentes sont dévoilées ainsi que toutes les cases contenant un numéro (ce numéro indique le nombre de mines adjacentes à cette case), cette étape se répète pour toutes les case vides ouvertes à ce moment-là.

➡ S'il tombe sur une case avec un chiffre alors il ne dévoile que cette case.

Le jeu s'arrête quand tout les cases ne comprenant pas de mine sont découvertes.

2) Le comportement attendu de l'algorithme.

Phase 1 : Choisir les dimensions du tableau.

Phase 2 : Initialiser le tableau.

Phase 3 : On affiche le tableau avec pour les cases découvertes le nombre de mines adjacentes.

Phase 4 : Demander à l'utilisateur les coordonnées d'une case.

Phase 5 : Vérifier la présence d'une mine aux coordonnées visées, s'il en a une la partie est perdue et on quitte la partie. Dans le cas contraire on continue en phase 6.

Phase 6 : La case est découverte ainsi que toutes les cases adjacentes qui ne contiennent pas de mines. De plus il y a propagation tant qu'il y a des cases vides adjacentes à celles découvertes.

Phase 7 : Si toutes les case vides sont découvertes alors la partie est gagnée ; sinon on recommence la phase 3.

3) Développement de l'algorithme.

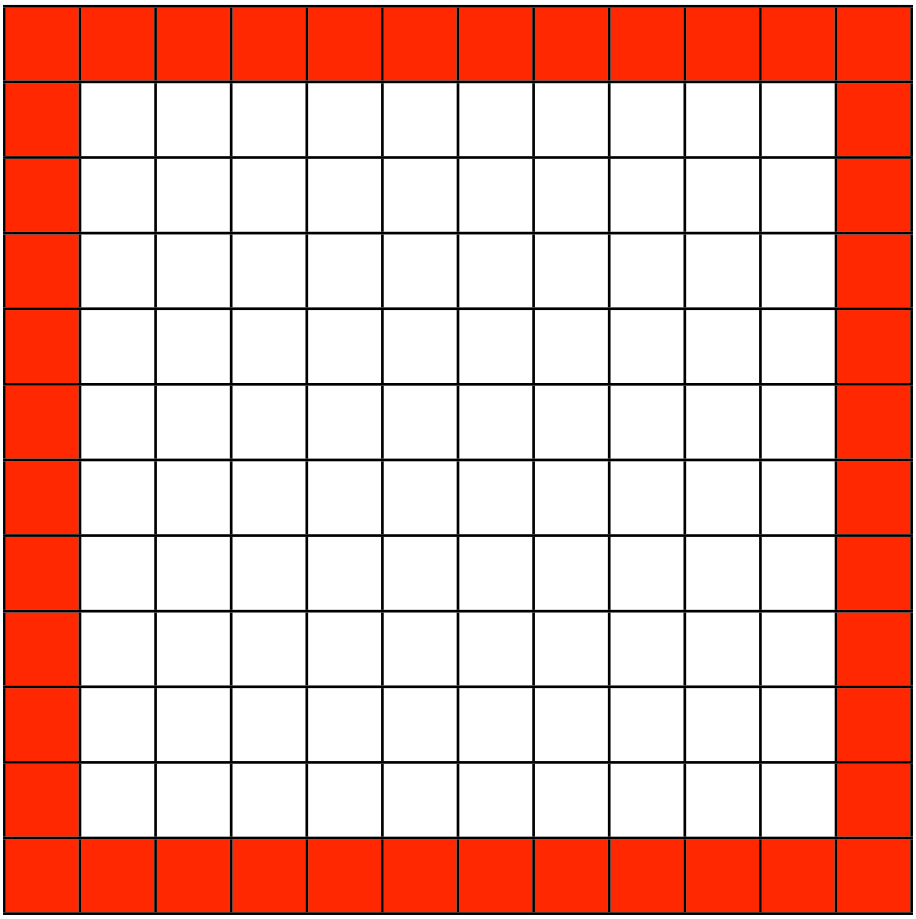
Etape 1 : Variable et constantes.

Constante : a et b qui sont les dimensions maximale du tableau (a pour les colonne et b pour les ligne)

```
constante
        a←52
        b←52
```

les dimensions maximales du tableau sont : a-2 et b-2, car il faut laisser un pourtour de case autour du tableau pour éviter les cas particulier.

exemple : on veut un démineur de 10 sur 10 donc on vas faire un tableau de 12 sur 12



Les case en rouge sont là pour éviter les cas particulier lors de la phase 6

Variable :

- Deux tableau tab et tabvis qui seront respectivement le tableau contiendra le jeux et le tableau qui sera afficher.

```
tab (1..a , 1..b) chaîne de caractère
// tableau qui contiendra le jeux
tabvisi (1..a , 1..b) chaîne de caractère
// tableau qui permettra d'afficher le démineur
```

- Il faut ensuite définir des variable pour le taille du tableau ainsi que pour les opération qui nécessite un balayage du tableau.

```
nbcolonne, nbligne : entier
// ils sont les deux variable qui contienne la taille du tableaux de jeux
i, j : entier
// ils permettent balayage complaît du tableau
k, l : entier
// ils permettent balayage pareille du tableau
```

- Il faut aussi une variable qui contienne le nombre de mine et une autre pour comptée le nombre de mine présente dans le tableau lors de la création du terrain.

```
nbmine : entier
// il comptent le nombre de mine a mètre dans le tableau
comptmine : entier
// il compte le nombre de mine présente dans le tableau lors de ça création
```

- On vas définir deux variable, une pour connaître le nombre de case découverte et l'autre pour savoure le nombre de case a découvrir.

```
nbcasedécou : entier
// compte le nombre de case découverte
nbcaseàdécou : entier
// compte le nombre de case à découvrir
```

-Une ligne nous permettras d'affichée les case du démineur.

ligne : caractère

- Une variable mineadja nous permettra de compter les mine adjacente à une case.

mineadja : entier

- Les variable m et n comporteront les cordonnées de la case que le joueur veut découvrir.

m, n : entier

- La variable perdu nous permettras d'arrêter le jeux si le ouvre une case ou il y a une mine.

perdu : entier

➡ **récapitulatif des constante et variable :**

```
Constante
        a←52
        b←52
Variable
        tab (1..a , 1..b) chaîne de caractère
        tabvisi (1..a , 1..b) chaîne de caractère
        i, j : entier
        nbcolonne, nbligne : entier
        nbmine : entier
        comptmine : entier
        nbcasedécou : entier
        nbcaseàdécou : entier
        ligne : caractère
        k, l : entier
        mineadja : entier
        m, n : entier
        perdu : entier
```

Etape 2 : Structure générale.

➡saisir le nombre de colonne

➡csaisir le nombre de ligne

➡saisir le nombre de mine

➡générée le tableau

➡répété

- ➡saisir une case
  - ➡si il y a une mine alors fin du jeux
  - ➡si non
    - ➡si la case est un chiffre alors afficher le chiffre
    - ➡si non afficher la case sélectionné ainsi que tout les case vide et contenant un chiffre adjacente

➡tant que soit aucune mine n'ai été découverte soit le nombre de case découverte et différent du nombre de case sans mine

➡si une mine et découverte affiché «perdu»

➡si non affiché «gagnée»

Etape 3 : Algorithme de la phase 1 - choix des dimension du tableau et du nombre de mines.

Dans cette phase on doit demander au joueur les dimension du démineur qu'il veut jouer ainsi que le nombre de mine qu'il veut placer. Il faut aussi vérifier si les dimension du démineur qu'il demande son valide (pas de valeur négative ni de valeur supérieur au dimension maximale défini dans les constantes) et si le nombre de mine est valide (pas plus de 20% de mine).

- Demande des dimensions et vérification de la validité des dimensions du démineur demander :

```
répéter
        afficher «saisir un nombre de colonne entre», 1, «et»,a - 2
        saisir nbcolone

        si nbcolonne > (a - 2) alors
                afficher «le démineur ne peut compter plus de», a - 2, «colonne»
        finsi

        si nbcolonne < 1
                afficher «le démineur ne peut pas avoir un nombre de colonne inférieur à 1»
        fin si


tant que nbcolonne > (a - 2) ou nbcolonne < 1

répéter
        afficher «saisir un nombre de ligne entre», 1, «et», b - 2
        saisir nbligne

        si nbligne  > (b - 2) alors
                afficher «le démineur ne peut compter plus de», b - 2, «ligne»
        fin si

        si nbligne < 1
                afficher «le démineur ne peut pas avoir un nombre de ligne inférieur à 1»
        fin si


tant que nbligne > (b - 2) ou nbcolonne < 1
```

- Demande du nombre de mine et vérification de sa validité :

```
répéter
        afficher «saisir le nombre de mine, il ne boit pas dépasser», (20/100)*(nbcolone* nbligne)
        saisir nbmine

        si nbmine > (20/100)*(nbcolone* nbligne)
                afficher «le démineur ne peut pas contenir plus de», (20/100)*(nbcolone*
                nbligne), «mine»

        si nbmine < 1 alors
                afficher «on ne peut pas avoirs moins d'une mine»
        fin si

        fin si

tant que  nbmine > (20/100)*(nbcolone* nbligne) et nbmine< 1
```

➡récapitulatif de la phase 1 :

```
répéter
        afficher «saisir un nombre de colonne entre», 1,»et»,a - 2
        saisir nbcolone
        si nbcolonne > (a - 2) alors
                afficher «le démineur ne peut compter plus de», a - 2, «colonne»
        fin si
        si nbcolonne < 1
                afficher «le démineur ne peut pas avoir un nombre de colonne inférieur à 1»
        fin si

tant que nbcolonne > (a - 2) ou nbcolonne < 1
répéter
        afficher «saisir un nombre de ligne entre», 1, «et», b - 2
        saisir nbligne
        si nbligne  > (b - 2) alors
                afficher «le démineur ne peut compter plus de», b - 2, «ligne»
        fin si
        si nbligne < 1
                afficher «le démineur ne peut pas avoir un nombre de ligne inférieur à 1»
        fin si
tant que nbligne > (b - 2) ou nbcolonne < 1
répéter
        afficher «saisir le nombre de mine, il ne boit pas dépasser», (20/100)*(nbcolone* nbligne)
        saisir nbmine
        si nbmine > (20/100)*(nbcolone* nbligne)
                afficher «le démineur ne peut pas contenir plus de», (20/100)*(nbcolone*
                nbligne), «mine»
        si nbmine < 1 alors
                afficher «on ne peut pas avoirs moins d'une mine»
        fin si
        fin si
tant que  nbmine > (20/100)*(nbcolone* nbligne) et nbmine< 1
```

Etape 4 : Algorithme de la phase 2 - Initialisation du tableau.

Il nous faut maintenant générait le tableau aléatoirement. Pour cela nous allons choisir aléatoirement une case du tableau puis choisir aléatoirement si on mets une mine ou pas.

- Il faut tout d'abord initialisé le compteur de mine a 0 et remplir les cases laisser en plus par qu'elle que chose qui ne risque pas de crée un problème. On utilise deux tableau car un contiendra le jeux et l'autre sera affichée (tad contient le jeux et tabvisi sera affichée) :

```
comptmine←0

pour i←1 à (nbcolonne + 2)

        pour j←1 à (nbcolone + 2)
                tab (i, j)←«Z»
                tabvisi (i, j)←«Z»
        fin pour

fin pour
```

- Maintenant nous pouvons remplir le tableau avec les mines (pour cette étape on ne prend en compte que les cases en blanc sur l'exemple au dessus), ainsi que mettre les espaces quand il n'y a pas de mine. Pour positionner les mines nous allons d'abord choisir une colonne au hasard puis une ligne, il faut vérifier s'il n'y a pas déjà une mine dans cette case et si il n'y en a pas on mette une mine, il faut aussi compter le nombre de mines que l'on place pour ne mettre le bon nombre :

```
répéter
        i←(HASARD()*nbcolonne) + 1
        j←(HASARD()*nbligne) + 1
        si tab (i, j) < > «B» alors
                tab (i, j)←«B»
                comptmine←comptmine + 1

        fin si
tant que comptmine < nbmine
pour i←2 à (nbcolone + 1)
        pour j2 à (nbligne + 1)
                si tab (i, j) < > «B» alors
                        tab (i, j)←« »
                fin si
fin pour
```

- Ramollissons maintenant le tableau qui affichera le jeu par des # pour montrer que la case n'est pas découverte :

```
pour i←2 à (nbcolone + 1)

        pour j←2 à (nbligne + 1)
                tabvisi (i, j)←«#»
        fin pour

fin pour
```

Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z
Z	#	#	#	#	#	#	#	#	#	#	Z
Z	#	#	#	#	#	#	#	#	#	#	Z
Z	#	#	#	#	#	#	#	#	#	#	Z
Z	#	#	#	#	#	#	#	#	#	#	Z
Z	#	#	#	#	#	#	#	#	#	#	Z
Z	#	#	#	#	#	#	#	#	#	#	Z
Z	#	#	#	#	#	#	#	#	#	#	Z
Z	#	#	#	#	#	#	#	#	#	#	Z
Z	#	#	#	#	#	#	#	#	#	#	Z
Z	#	#	#	#	#	#	#	#	#	#	Z
Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z

seule les case en blanc seront visible, le principe et le même pour le tableau tab sauf qu'a la place des # il y a soit des B, ce qui indique qu'il y a des mine, soit des espace.

- Il ne nous reste plus qu'a indiquer le nombre de mines adjacentes aux cases. Le nombre de mine adjacentes peut aller de 0, on laisse l'espace dans ce cas là, à 8. Pour cela on va regarder toutes les cases du tableau et pour chaque case contenant un espace on va compter le nombre de mines adjacentes, les cases en rouge remplies d'un Z nous permettent d'éviter d'avoir à vérifier si la case où l'on regarde son contenu existe :

```
pour i← 2 à (nbcolone + 1)

        pour j←2 à (nbligne + 1)

                si tab (i, j) < > «B»
                        mineadja←0

                        pour k←i-1 à i+1

                                pour l←j-1 à j+1

                                        si tab (k, l) = «B» alors
                                                mineadja ←mineadja + 1
                                        fin si

                                fin pour

                        fin pour
                        tab (i, j)← mineadja

                fin si

        fin pour

fin pour
```

Exemple :

B	2	
B	3	2
1	2	B

➡**récapitulatif de la phase 2 :**

```
comptmine←0
pour i←1 à (nbcolonne + 2)
        pour j←1 à (nbcolone + 2)
                tab (i, j)←«Z»
                tabvisi (i, j)←«Z»
        fin pour
fin pour
répéter
        i←(HASARD()*nbcolonne) + 1
        j←(HASARD()*nbligne) + 1
        si tab (i, j) <> «B» alors
                alea←HASARD()
                si alea > 0,5 alors
                        tab (i, j)←«B»
                        comptmine←comptmine + 1
                fin si
        fin si
tant que comptmine < nbmine
pour i←2 à (nbcolone + 1)
        pour j←2 à (nbligne + 1)
                si tab (i, j) <> «B» alors
                        tab (i, j)←« »
                fin si
fin pour
pour i←2 à (nbcolone + 1)
        pour j←2 à (nbligne + 1)
                tabvisi (i, j)←«#»
        fin pour
fin pour
pour i← 2 à (nbcolone + 1)
        pour j←2 à (nbligne + 1)
                si tab (i, j) <> «B»
                        mineadja←0
                        pour k←i-1 à i+1
                                pour l←j-1 à j+1
                                        si tab (k, l) = «B» alors
                                                mineadja ←mineadja + 1
                                        fin si
                                fin pour
                        fin pour
                        tab (i, j)← mineadja
                fin si
        fin pour
fin pour
```

Etape 5 : Algorithme de la phase 3 - Afficher le tableau.

Pour afficher le tableau tabvisi nous allons nous servir d'une variable caractère (la variable ligne) qui nous permettra d'afficher le tableau ligne par ligne. Pour ce faire nous allons rentrer dans cette variable un espace puis ce que contient une case puis ainsi de suite et ce ligne par ligne :

```
pour j←2 à (nbligne + 1)
        ligne←« »

        pour i←2 à (nbcolonne + 1)
                ligne←ligne, « », tabvisi (i, j)
        fin pour

        afficher ligne
fin pour
```

Exemple : voici ce que le joueur verra à l'écran (le premier tour de jeux)

```
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
```

Etape 6 : Algorithme de la phase 4 - Coup du joueur.

Maintenant que le jeu est créé et affiché il nous faut permettre au joueur de sélectionner une case et vérifier si la case sélectionnée est valide :

```
répéter
        afficher «saisir les coordonnées de la case à découvrir, sous la forme (colonne, ligne)»
        afficher «les coordonnées des cases ne doive pas dépasser (», nbcolonne, «,», nbligne,
        «)»
        saisir m, n
        m←m + 1
        // on rajoute 1 a m car il y a une case en plus qui ne sert pas pour le jeu (la case
        en rouge dans les exemple précédant.
        n←n + 1
        // on rajoute 1 a m car il y a une case en plus qui ne sert pas pour le jeux (la case
        en rouge dans les exemple précédant.
        si m > (nbcolonne + 1)) alors
                afficher «la colonne n'existe pas»
        fin si

        si m < 2 alors
                afficher «la colonne n'existe pas»
        fin si

        si n > (nbligne + 1) alors
                afficher «la ligne n'existe pas»
        fin si

        si n < 2 alors
                afficher «la ligne n'existe pas»
        fin si

        si tabvisi (m, n) < > «#» alors
                afficher «case déjà découverte»
        fin si
```

Etape 7 : Algorithme de la phase 5 - Vérification de la présence d'une mine aux coordonnées visées.

Maintenant que nous avons vérifié si le coup du joueur était valide nous allons maintenant vérifier s'il n'y a pas de mine dans la case sélectionnée par le joueur.

```
perdu←0
// cette variable nous permettra d'arrêté le jeux quand une mine et découverte

si tab (m, n) = «B» alors
        perdu←1
// quand la variable perdu est à 1 alors le jeux ce termine .
si non
        tabvisi (m, n)←«@»
        // on marque la case a découvrir par un @ ce qui nous permettra par la suit de
        pouvoir l'ouvrir et de pouvoir ouvrir tout les case vide adjacente.
        nbcaseàdecou ←1
fin si
```

Etape 8 : Algorithme de la phase 6 - Découvrir la case vide et les cases vides adjacentes.

Dans le cas où la case visée ne contient pas de mine il y a deux possibilités :

- Soit la case contient un chiffre et dans ce cas là juste cette case est découverte.
- Soit la case contient un espace et dans ce cas là toutes les cases adjacentes à cette case sont découvertes, cette opération se répète tant qu'une case contenant un espace est découverte.

Pour ce faire nous allons regarder les cases qui sont marquées d'un @ :

- Si il y a un chiffre on rentre juste la valeur de cette case dans le tableau d'affichage.
- Si elle est vide (elle contient un espace dans ce cas là) on met un @ sur toutes les cases adjacentes et on remplace l'@ du tableau d'affichage par un espace. Seules les cases qui contiennent un # peuvent voir leur valeur remplacée par un @. Les cases que nous avons remplies par des Z nous servent à ce niveau là car cela nous évite de devoir vérifier si les coordonnées de la case où l'on veut voir si elle contient toujours un # sont valides.

```
répéter
        pour i←2 à (nbcolonne + 1)
                pour j←2 à (nbligne + 1)
                        si tabvisi (i, j) = «@» alors
                                nbcaseàdécou←nbcaseàdécou - 1
                                si tad (i, j) = « » alors
                                        tabvisi←« »
                                        nbcasedécou←nbcasedécou + 1
                                        pour k←i-1 à k+1
                                                pour l← j-1 à j+1
                                                        si tabvisi (k, l) = «#» alors
                                                                tabvisi (k, l)←«@»
                                                                nbcaseàdécou←nbcaseàdécou + 1
                                                        fin si
                                                fin pour
                                        fin pour
                                si non
                                        tabvisi (i, j)←tad (i, j)
                                fin si
                        fin si
                fin pour
        fin pour
tant que nbcaseadécou < > 0
```

➡ Il faut rajouter une boucle «répéter ... tant que ...» de l'étape 5 à l'étape 8 ainsi qu'afficher le résultat du jeu :

```
répéter
        Etape 5

        Etape 6

        Etape 7

        Etape 8
tant que nbcasedécou < ((nbligne*nbcolonne) - nbmine) ou perdu < > 1

si perdu = 1 alors
        afficher «BOUM vous avez perdu :-( !!!»
si non
        afficher «vous avez gagné :-) !!!»
```

Récapitulatif de l'algorithme.

```
Programme démineur

Constante

        a←52

        b←52

Variable

        tab (1..a , 1..b) chaîne de caractère

        tabvisi (1..a , 1..b) chaîne de caractère

        i, j : entier

        alea : réel

        nbcolonne, nbligne : entier

        nbmine : entier

        comptmine : entier

        nbcasedécou : entier

        nbcaseàdécou : entier

        ligne : caractère

        k, l : entier

        mineadja : entier

        m, n : entier

        perdu : entier

répéter
        afficher «saisir un nombre de colonnes entre», 1, «et»,a - 2
        saisir nbcolone

        si nbcolonne > (a - 2) alors
                afficher «le démineur ne peut compter plus de», a - 2, «colonnes»
        fin si

        si nbcolonne < 1
                afficher «le démineur ne peut pas avoir un nombre de colonnes inférieur à 1»
```

```
        fin si


tant que nbcolonne > (a - 2) ou nbcolonne < 1

répéter
        afficher «saisir un nombre de lignes entre», 1, «et», b - 2
        saisir nbligne

        si nbligne  > (b - 2) alors
                afficher «le démineur ne peut compter plus de», b - 2, «lignes»
        fin si

        si nbligne < 1
                afficher «le démineur ne peut pas avoir un nombre de lignes inférieur à 1»
        fin si

tant que nbligne > (b - 2) ou nbcolonne < 1

répéter
        afficher «saisir le nombre de mine, il ne doit pas dépasser», (20/100)*(nbcolone* nbligne)
        saisir nbmine

        si nbmine > (20/100)*(nbcolone* nbligne)
                afficher «le démineur ne peut pas contenir plus de», (20/100)*(nbcolone*
                nbligne), «mines»

        si nbmine < 1 alors
                afficher «on ne peut pas avoirs moins d'une mine»
        fin si

        fin si

tant que  nbmine > (20/100)*(nbcolone* nbligne) et nbmine< 1

comptmine←0

pour i←1 à (nbcolonne + 2)

        pour j←1 à (nbcolone + 2)
                tab (i, j)←«Z»
                tabvisi (i, j)←«Z»
        fin pour

fin pour

répéter
        i←(HASARD()*nbcolonne) + 1
        j←(HASARD()*nbligne) + 1

        si tab (i, j) < > «B» alors
```

```
                    tab (i, j)←«B»
                    comptmine←comptmine + 1


        fin si

tant que comptmine < nbmine

pour i←2 à (nbcolone + 1)

        pour j2 à (nbligne + 1)

                si tab (i, j) < > «B» alors
                        tab (i, j)←« »
                fin si

fin pour

pour i←2 à (nbcolone + 1)

        pour j←2 à (nbligne + 1)
                tabvisi (i, j)←«#»
        fin pour

fin pour

pour i← 2 à (nbcolone + 1)

        pour j←2 à (nbligne + 1)

                si tab (i, j) < > «B»
                        mineadja←0

                        pour k←i-1 à i+1

                                pour l←j-1 à j+1

                                        si tab (k, l) = «B» alors
                                                mineadja ←mineadja + 1
                                        fin si

                                fin pour

                        fin pour
                        tab (i, j)← mineadja

                fin si

        fin pour
```

```
fin pour

répéter

        pour j←2 à (nbligne + 1)
                ligne←« »

                pour i←2 à (nbcolonne + 1)
                        ligne←ligne, « », tabvisi (i, j)
                fin pour

                afficher ligne
        fin pour

        répéter
                afficher «saisir les coordonnées de la case a découvrir, sous la forme (colonne, ligne)»
                afficher «les coordonnéés des case ne doive pas dépassée (», nbcolonne, «,»,
                nbligne, «)»
                saisir m, n
                m←m + 1
                n←n + 1

                si m > (nbcolonne + 1)) alors
                        afficher «la colonne n'existe pas»
                fin si

                si m < 2 alors
                        afficher «la colonne n'existe pas»
                fin si

                si n > (nbligne + 1) alors
                        afficher «la ligne n'existe pas»
                fin si

                si n < 2 alors
                        afficher «la ligne n'existe pas»
                fin si

                si tabvisi (m, n) < > «#» alors
                        afficher «case déjà découverte»
                fin si

        tant que m > (nbcolonne + 1) ou m < 2 ou n > (nbligne + 1) ou n < 2 ou abvisi (m, n) < > «#»

        si tab (m, n) = «B» alors
                perdu←1
        si non
                tabvisi (m, n)←«@»
                nbcaseàdecou ←1
        fin si
```

```
répéter

        pour i←2 à (nbcolonne + 1)

                pour j←2 à (nbligne + 1)

                        si tabvisi (i, j) = «@» alors
                                nbcaseàdécou←nbcaseàdécou - 1

                                si tad (i, j) = « » alors
                                        tabvisi←« »
                                        nbcasedécou←nbcasedécou + 1

                                        pour k←i-1 à k+1

                                                pour l← j-1 à j+1

                                                        si tabvisi (k, l) = «#» alors
                                                        tabvisi (k, l)←«@»
                                                        nbcaseàdécou←nbcaseàdécou + 1

                                                        fin si

                                                fin pour

                                        fin pour
                                si non
                                        tabvisi (i, j)←tad (i, j)
                                fin si

                        fin si

                fin pour

        fin pour

tant que nbcaseadécou < > 0

tant que nbcasedécou < ((nbligne*nbcolonne) - nbmine) ou perdu < > 1

si perdu = 1 alors
        afficher «BOUM vous avez perdu :-( !!!»
si non
        afficher «vous avez gagné :-) !!!»
```

4) Variante / évolution .

Dans cette variante nous allons ajouter la possibilité de mettre des drapeaux. La structure principale de l'algorithme ne change pas. Seule l'étape 7 change ainsi que les conditions de victoire.

➡ Etape 7 :

Il y a deux changements dans cette étape :

- On doit vérifier si la case contient un drapeau, dans ce cas là on demande au joueur ce qu'il veut faire soit ne rien faire, soit enlever le drapeau.
- Si la case ne contient pas de drapeau le joueur à deux possibilités soit il pose un drapeau sur la case, soit il dévoile la case.

Pour cela il nous faut rajouter une variable, la variable drapeau qui comptera le nombre de drapeaux posés.

```
variable
        drapeau : entiers

si tadvisi (m, n) = «D» alors
        afficher «la casse contient un drapeau»
        afficher «si vous voulez enlever le drapeau taper 1»
        afficher «si non taper 0»
        saisir drapeau

        si drapeau = 1 alors
                tabvisi (m, n)← «#»
        fin si
si non
        afficher «si vous voulez poser un drapeau taper 2»
        afficher «si vous voulez découvrir la case taper 3»
        saisir drapeau

        si drapeau = 2 alors
                tabvisi (m, n)←«D»
        si non
                si tab (m, n) = «B» alors
                        perdu←1
                si non
                        tabvisi (m, n)←«@»
                        nbcaseàdecou ←1
                fin si
        fin si
fin si
```

➡ **Condition de victoire :**

Il faut ajouter dans les conditions de victoire que le nombre de drapeaux soit égal au nombre de mines.

```
répéter
        Etape 5

        Etape 6

        Etape 7

        Etape 8
tant que nbcasedécou < ((nbligne*nbcolonne) - nbmine) et drapeau < > nbmine ou perdu < > 1
```