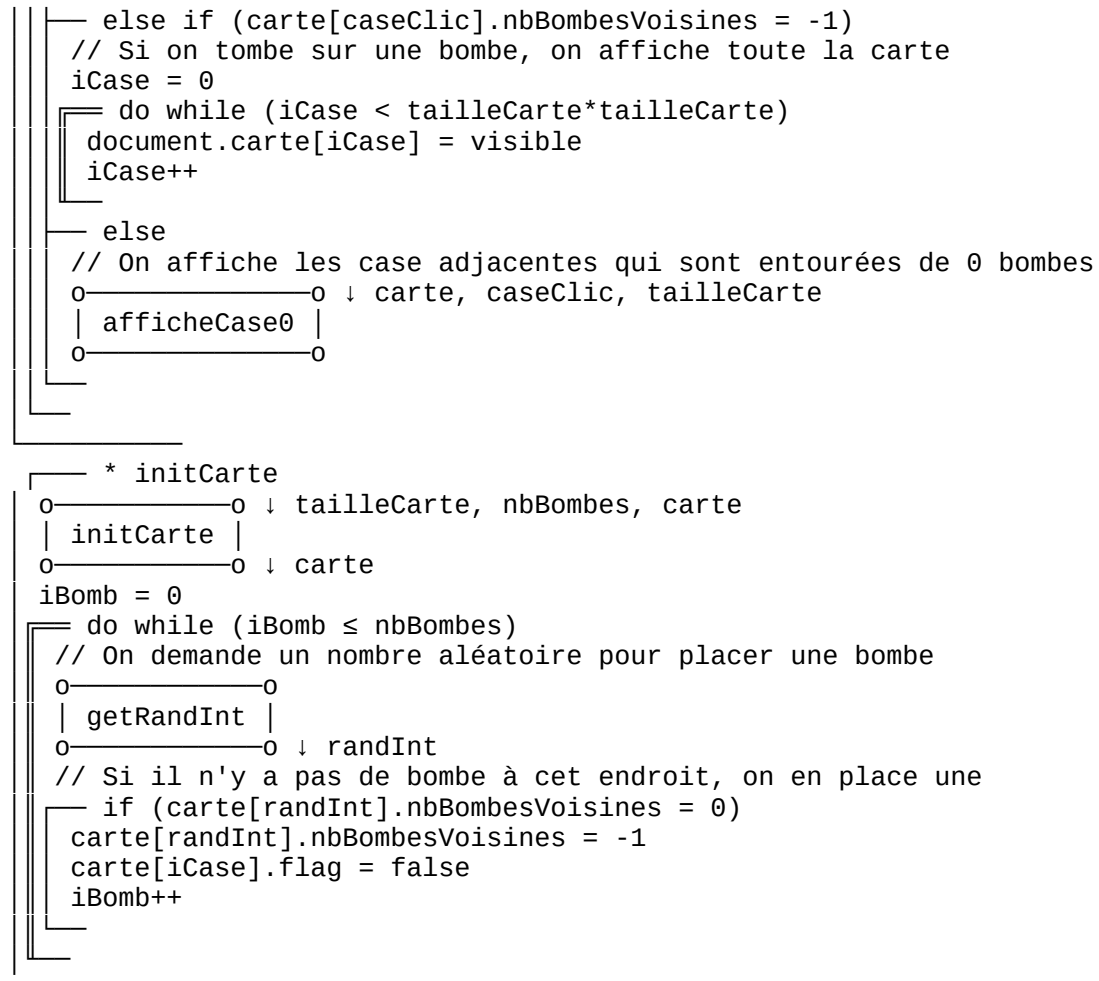
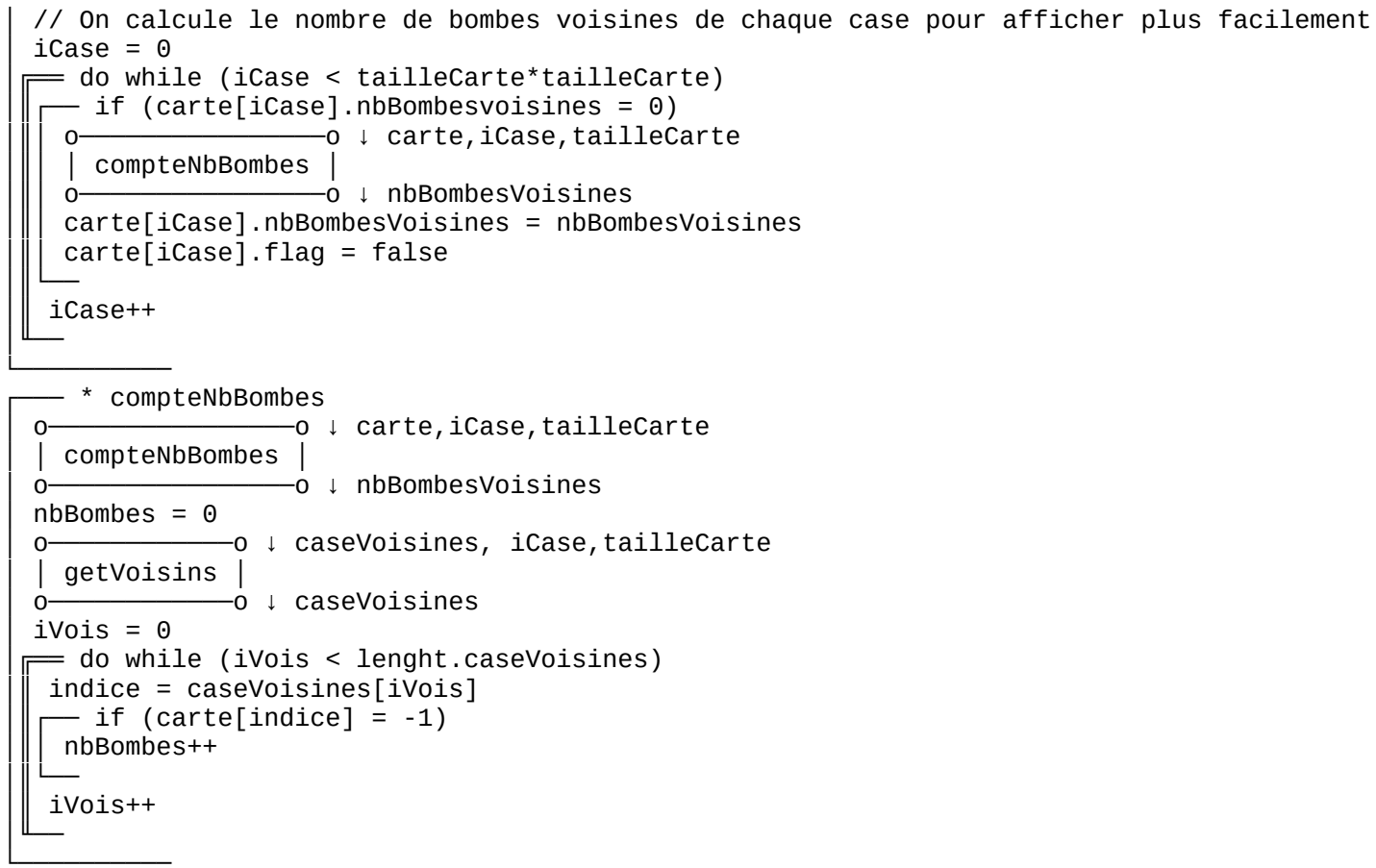
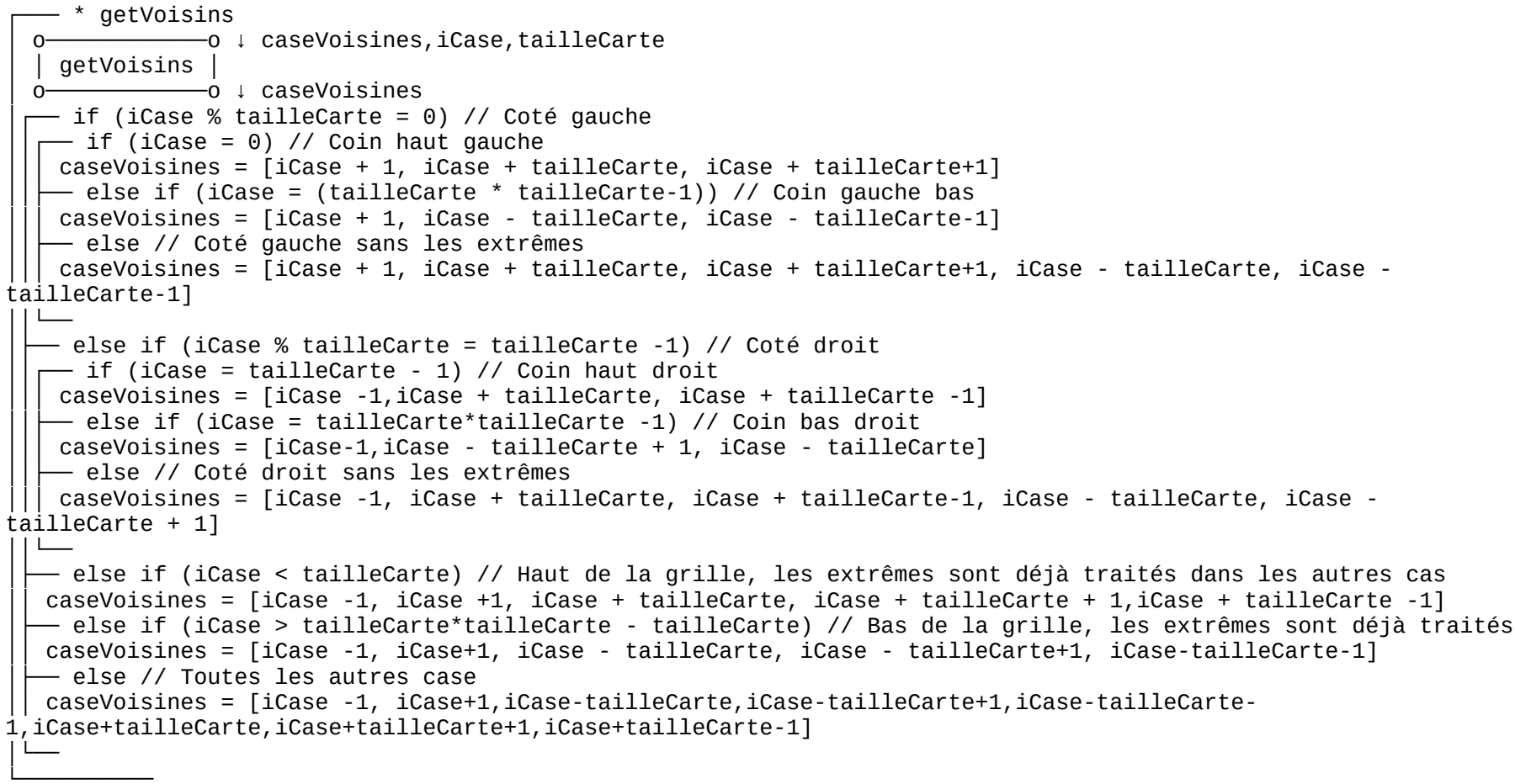
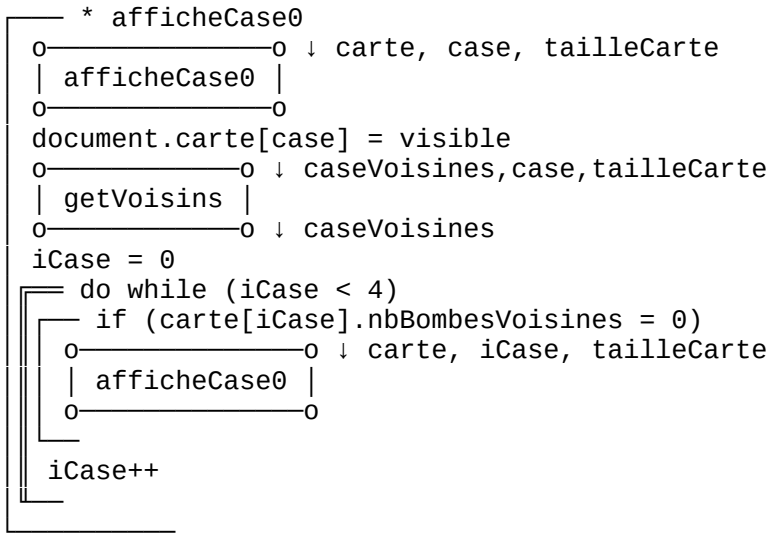
```
// A créer :
 // ---------
 // carte n* : case

 // Case : nbBombesVoisines (-1 si une bombe est sur la case)
 //      : flag (bool)
 // n = tailleCarte²

 // La carte est un simple tableau mais utlisé comme un double [EX 5x5] :
 // 00 01 02 03 04
 // 05 06 07 08 09
 // 10 11 12 13 14
 // 15 16 17 18 19
 // 20 21 22 23 24

 // "document" est un élement du DOM qui affiche la case

┌── * Démineur
│ Obtenir difficulté
│ // Définition des paramètres de la partie
│┌── if (difficulté = "facile")
│ tailleCarte = 10
│ nbBombes = 10
│├── else if (difficulté = "moyen")
│ tailleCarte = 15
│ nbBombes = 50
│├── else
│ tailleCarte = 20
│ nbBombes = 100
│└──
│ o──────────o ↓ tailleCarte, nbBombes, carte
│ │ initCarte │
│ o──────────o ↓ carte
│ Obtenir clic, caseClic
│┌── if (clic = rightClic)
│ // Ajout d'un drapeau si l'utilisateur fait un clic droit
│ carte[caseClic].flag = true
│├── else if (clic = leftClic)
│┌── if (carte[caseClic].flag)
│ // Si on clique sur un drapeau, on l'enlève
│ carte[caseClic].flag = false
```


else if (carte[caseClic].nbBombesVoisines = -1)
// Si on tombe sur une bombe, on affiche toute la carte
iCase = 0
do while (iCase < tailleCarte*tailleCarte)
document.carte[iCase] = visible
iCase++
else
// On affiche les case adjacentes qui sont entourées de 0 bombes
afficheCase0
carte, caseClic, tailleCarte
* initCarte
initCarte
tailleCarte, nbBombes, carte
carte
iBomb = 0
do while (iBomb ≤ nbBombes)
// On demande un nombre aléatoire pour placer une bombe
getRandInt
randInt
// Si il n'y a pas de bombe à cet endroit, on en place une
if (carte[randInt].nbBombesVoisines = 0)
carte[randInt].nbBombesVoisines = -1
carte[iCase].flag = false
iBomb++


// On calcule le nombre de bombes voisines de chaque case pour afficher plus facilement
iCase = 0
do while (iCase < tailleCarte*tailleCarte)
if (carte[iCase].nbBombesvoisines = 0)
compteNbBombes
carte[iCase].nbBombesVoisines = nbBombesVoisines
carte[iCase].flag = false
iCase++
* compteNbBombes
compteNbBombes
nbBombesVoisines
nbBombes = 0
getVoisins
caseVoisines
iVois = 0
do while (iVois < lenght.caseVoisines)
indice = caseVoisines[iVois]
if (carte[indice] = -1)
nbBombes++
iVois++


* getVoisins
getVoisins
↓ caseVoisines,iCase,tailleCarte
↓ caseVoisines
if (iCase % tailleCarte = 0) // Coté gauche
if (iCase = 0) // Coin haut gauche
caseVoisines = [iCase + 1, iCase + tailleCarte, iCase + tailleCarte+1]
else if (iCase = (tailleCarte * tailleCarte-1)) // Coin gauche bas
caseVoisines = [iCase + 1, iCase - tailleCarte, iCase - tailleCarte-1]
else // Coté gauche sans les extrêmes
caseVoisines = [iCase + 1, iCase + tailleCarte, iCase + tailleCarte+1, iCase - tailleCarte, iCase - tailleCarte-1]
else if (iCase % tailleCarte = tailleCarte -1) // Coté droit
if (iCase = tailleCarte - 1) // Coin haut droit
caseVoisines = [iCase -1,iCase + tailleCarte, iCase + tailleCarte -1]
else if (iCase = tailleCarte*tailleCarte -1) // Coin bas droit
caseVoisines = [iCase-1,iCase - tailleCarte + 1, iCase - tailleCarte]
else // Coté droit sans les extrêmes
caseVoisines = [iCase -1, iCase + tailleCarte, iCase + tailleCarte-1, iCase - tailleCarte, iCase - tailleCarte + 1]
else if (iCase < tailleCarte) // Haut de la grille, les extrêmes sont déjà traités dans les autres cas
caseVoisines = [iCase -1, iCase +1, iCase + tailleCarte, iCase + tailleCarte + 1,iCase + tailleCarte -1]
else if (iCase > tailleCarte*tailleCarte - tailleCarte) // Bas de la grille, les extrêmes sont déjà traités
caseVoisines = [iCase -1, iCase+1, iCase - tailleCarte, iCase - tailleCarte+1, iCase-tailleCarte-1]
else // Toutes les autres case
caseVoisines = [iCase -1, iCase+1,iCase-tailleCarte,iCase-tailleCarte+1,iCase-tailleCarte-1,iCase+tailleCarte,iCase+tailleCarte+1,iCase+tailleCarte-1]


* afficheCase0
afficheCase0
carte, case, tailleCarte
document.carte[case] = visible
getVoisins
caseVoisines,case,tailleCarte
caseVoisines
iCase = 0
do while (iCase < 4)
if (carte[iCase].nbBombesVoisines = 0)
afficheCase0
carte, iCase, tailleCarte
iCase++