Le programme lit les paramètres du jeu et initialise deux tableaux de lignes x colonnes. Si *nbS* est égal à 0, il n’y a jamais d’affichage, sinon il commence par imprimer l’image initiale. Il va ensuite faire les *nbJ* étapes d’analyse en alternant analyse de T1->T2 puis analyse de T2->T1. Pour chaque case du tableau qui est analysé, il va regarder si cette case se trouve dans une bordure, si c’est le cas, il va voir quelle bordure elle se trouve afin d’appeler la fonction ***caseVivante*** avec un nombre à 4 bits (décimal de 0 à 15) **[haut|droite|gauche|bas]** représentant la position de la case dans le tableau, par exemple le nombre binaire 0011 correspond à gauche & bas donc le coin en bas à gauche, 0000 à une case par dans les bordures etc… cela permet à la fonction ***caseVivante*** d’analyser seulement les cases possédant une valeur et de faire un grand gain en terme de performances temporelles et spatiales. La fonction ***caseVivante*** compte donc le nombre de voisins vivant et en fonction de la case concernée, elle écrit dans l’autre tableau l’état de la cellule à étape+1. Quand l’analyse de l’étape est terminée, le programme appelle la fonction d’impression si les conditions sont remplies. Cette fonction d’impression va imprimer une ligne noire saut lors de l’impression de l’étape initiale. Elle imprime ensuite le tableau agrandi zoom fois en dupliquant zoom fois l’impression de chaque colonne, puis de chaque ligne. Si l’impression de la colonne dupliquée dépasse les 60 caractères, un retour à la ligne est imprimé d’abord.

Algorithmiwque

Start  
 Étape 🡨 1  
 tant que étape <= nbJ  
 si étape paire  
 analyse(tab2, tab1)  
 sinon  
 analyse(tab1, tab2)  
 si nbS !=0 et nbS divise étape  
 si étape paire  
 output(tab1, étape)  
 sinon  
 output(tab2, étape)