

Le schéma conditionnel :

Exercice n°1 : Si alorssinon

Effacer de manière conditionnelle une partie de la figure :

A partir de l'exercice précédent, on souhaite faire afficher une flèche comme celle utilisée sur les camions de service sur l'autoroute. A la place d'appeler la fonction **Effacer** de la bibliothèque SenseHat, si la ligne et la colonne sont différentes, les leds vont être allumées avec la couleur NOIR, comme le montre l'extrait du programme.

```
for (colonne = 0 ; colonne <= ligne ; colonne++)
{
    if (ligne != colonne)
    {
        Allumer(ligne,colonne,NOIR);
        Allumer(colonne,ligne,NOIR);
    }
}
```

	0	1	2	3	4	5	6	7
0								
1								
2								
3								
4								
5								
6								
7								

	0	1	2	3	4	5	6	7
0								
1								
2								
3								
4								
5								
6								
7								

Après avoir remplacé la fonction **Effacer** par les lignes ci-dessus, la figure obtenue représente toutes les étapes ci-contre.

Que se passe-t-il si l'algorithme devient :

Avant	Après
<u>si</u> ligne ≠ colonne <u>alors</u> Allumer(ligne,colonne,NOIR) Allumer(colonne,ligne,NOIR) <u>finSi</u>	<u>si</u> ligne = colonne <u>alors</u> Allumer(ligne,colonne,ROUGE) <u>sinon</u> Allumer(ligne,colonne,NOIR) Allumer(colonne,ligne,NOIR) <u>finSi</u>

Remarques :

- Le test d'égalité se code avec un double égal « == » pour le différencier du simple égal « = » qui représente l'affectation : ←
- Le **alors** ne se code pas en C, mais le **sinon** se code avec le mot clé : **else**

Codez cette deuxième version pour vérifier le résultat attendu.

Exercice n°2 : cas parmi... Le jeu de la « Chasse au trésor »

Dans un premier temps, on souhaite promener une led sur la matrice en agissant avec le joystick.

L'algorithme et le programme sont les suivants :

Algorithme	Langage C
<p>Lexique des variables : posX entier posY entier touche entier sortie entier</p> <p>Définition de constantes : FAUX 0 VRAI 1</p>	<pre> 1 #include "senseHat.h" 2 3 #define FAUX 0 4 #define VRAI 1 5 int main() 6 { 7 int posX = 4; 8 int posY = 4; 9 int touche ; 10 int sortie = FAUX; 11 12 InitialiserLeds(); 13 InitialiserJoystick(); 14 15 do 16 { 17 Allumer(posX,posY,ROUGE); 18 touche = ScannerJoystick(); 19 switch(touche) 20 { 21 case KEY_ENTER: 22 sortie = VRAI; 23 break; 24 case KEY_RIGHT: 25 posY++; 26 break; 27 case KEY_LEFT: 28 posY--; 29 break; 30 case KEY_UP: 31 posX--; 32 break; 33 case KEY_DOWN: 34 posX++; 35 break; 36 } 37 } while(sortie == FAUX); 38 39 return 0; 40 } </pre>
<p>début</p> <p>InitialiserLeds() InitialiserJoystick() posX ← 4 posY ← 4 sortie ← FAUX <u>faire</u></p> <p> Allumer(posX,posY,ROUGE) touche ← ScannerJoystick() <u>cas</u> touche parmi</p> <p> KEY_ENTER : sortie ← VRAI KEY_RIGHT : posY = posY + 1 KEY_LEFT : posY = posY - 1 KEY_UP : posX = posX - 1 KEY_DOWN : posX = posX + 1</p> <p> <u>finCas</u></p> <p> <u>tantQue</u> sortie = FAUX</p> <p>fin</p>	

Remarque :

La structure **cas...parmi...** est une succession de **si...alors...sinon si...alors...sinon...**

Faites un nouveau projet nommé **ChasseAuTresor** et codez la fonction `int main()`. Après avoir testé ce programme, que constatez-vous lors du déplacement de la led rouge ?

On souhaite maintenant supprimer cette trace laissée par le déplacement de la led.

Pour cela, on peut retenir l'idée :

Avant la modification des valeurs posX et posY, on les mémorise dans des variables preX et preY pour conserver la valeur précédente. Puis, après la structure de contrôle « **cas...parmi...** », si posX est différent de preX ou posY est différent de preY on éteint la led à la position preX, preY (allumer la led en Noire).

Modifiez votre programme avec cette première amélioration.

Que se passe-t-il lorsque l'on dépasse les limites de la matrice avec le joystick ?

Pour cela il est nécessaire de limiter les valeurs de posX et posY pour qu'elles restent dans l'intervalle [0 .. 7]. Après chaque opération sur une des variables, il faut vérifier si la valeur n'est pas en dehors des limites par exemple :

Extrait de l'algorithme modifié	
début	
.....	
<u>cas</u> touche parmi	
.....	
KEY_RIGHT :	
posY = posY+1	
si posY > 7	Pour chaque cas, une condition doit être utilisée
alors posY ← 7	pour vérifier si la limite est dépassée. La valeur de
finSi	posY reste à 7 si elle devient trop grande
KEY_LEFT :	
.....	
<u>finCas</u>	
.....	
<u>fin</u>	

Complétez votre programme pour tous les cas de figure pour que la led rouge se déplace bien dans la matrice sans jamais en sortir.

Maintenant, il faut placer un trésor sur la matrice. Ce trésor sera récupéré en déplaçant la led rouge avec le joystick.

Le trésor s'affiche en orange à la position **tresorX** et **tresorY** deux variables qui sont initialisées respectivement avec les valeurs 2 et 6 par exemple. Faites afficher le trésor sur la matrice de led.

Modifier ensuite votre programme pour qu'après le déplacement de la led rouge si **posX** est égale à **tresorX** et **posY** est égale à **tresorY**, la led devienne verte : on a ramassé le trésor.

Le trésor ayant à présent disparu, il serait bien qu'un nouveau trésor apparaisse à une autre position de manière aléatoire.

La fonction **int rand()** de la librairie **<stdlib.h>** retourne un nombre entier de manière pseudo aléatoire compris entre 0 et RAND_MAX le plus souvent égale à 32767.

Pour obtenir un nombre compris entre 0 et 7, il faut utiliser le reste de la division par 8.

Exemple : x = rand() % 8 ;

Complétez votre programme pour qu'après chaque disparition du trésor un nouveau soit positionné de manière aléatoire. Que remarquez-vous sur le tirage aléatoire des nombres ?

Pour résoudre ce problème, on peut améliorer le programme en initialisant la séquence aléatoire à partir de l'heure courante. Au début de votre programme, il faut ajouter la ligne :

```
srand(time(NULL));
```

Enfin, il peut arriver que le trésor soit en posX et posY. Dans ce cas tant que les coordonnées du trésor sont celles de la led rouge, on retire aléatoirement de nouvelles coordonnées pour le trésor. Modifiez votre programme pour réaliser ce traitement.