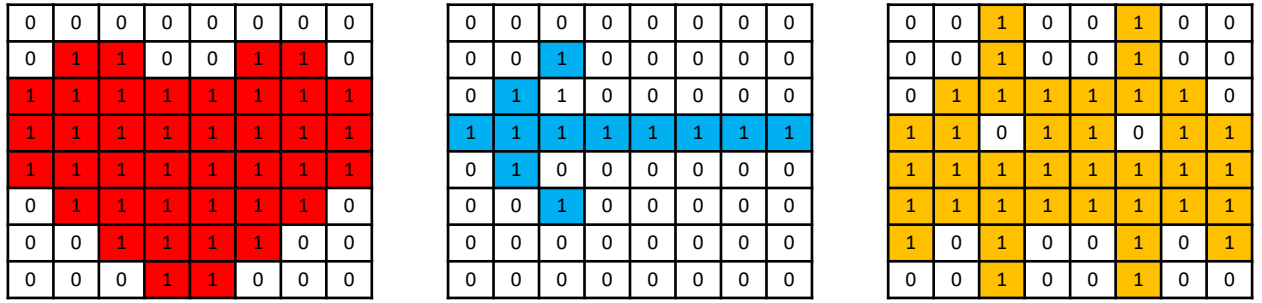
Petit guide pour la détection des collisions au pixel près

Jérôme Clet-Ortega, Raymond Namyst



Sprites et transparence

- Détecter les collisions « au pixel près » de manière efficace
  - Prétraitement des images
    - Chargement de l'image dans une surface SDL (et non une texture directement)
    - Scan des pixels en utilisant un seuil sur la composante alpha pour décider « opaque » (1) ou transparent (0)
    - Construction d'un tableau de bits

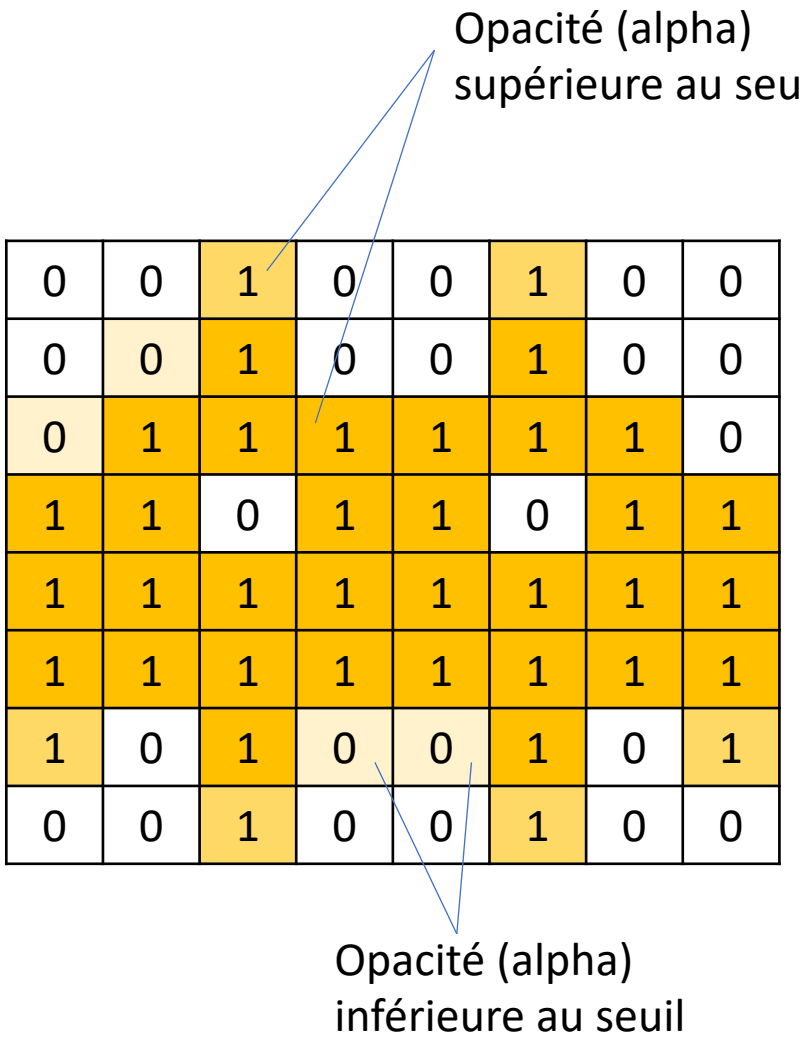
0	0	1	0	0	1	0	0
0	0	1	0	0	1	0	0
0	1	1	1	1	1	1	0
1	1	0	1	1	0	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	0	1	0	0	1	0	1
0	0	1	0	0	1	0	0

Sprites et transparence

```
Uint8 r, b, g, alpha;
Uint32 *p;
SDL_PixelFormat *fmt = surface->format;
int bpp = fmt->BytesPerPixel;

for (y = 0; y < surface->h; y++)
  for (x = 0; x < surface->w; x++) {
    p = (Uint32 *)(
         (Uint8 *)surface->pixels +
         y * surface->pitch + x * bpp);

    SDL_GetRGBA (*p, fmt, &r, &g, &b, &alpha);
    if (alpha > 64) // exemple de seuil
      // pixel at (x,y) is opaque
      …
```



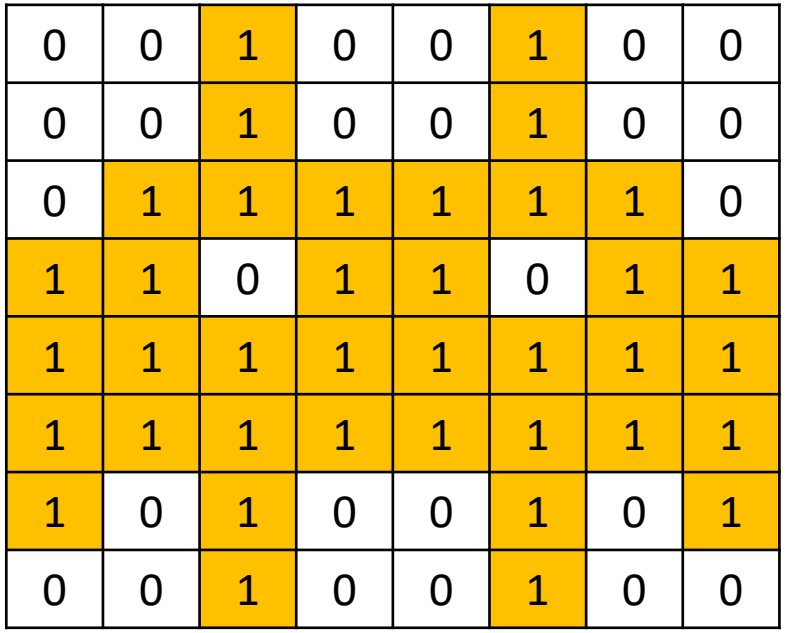
Sprites et transparence

- Stockage compact des bits
  - Pour l'image ci-contre (8x8), l'information peut être mémorisé dans une suite de 8 octets:

00100100
00100100
01111110
11011011
11111111
11111111
10100101
00100100

uint8_t collision_mask[8];

- Notez que pour des sprites 64x64, le type uint64_t est capable d'accueillir une ligne



0	0	1	0	0	1	0	0
0	0	1	0	0	1	0	0
0	1	1	1	1	1	1	0
1	1	0	1	1	0	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	0	1	0	0	1	0	1
0	0	1	0	0	1	0	0

Sprites et transparence

- Ces masques de collision doivent être pré-calculés
  - Pour tous les sprites d'objets « qui peuvent entrer en collision »
  - Pour tous les sprites d'éléments de la map qui sont « solides »
  - Et ce, pour chaque image du sprite…
- C'est une bonne idée de :
  - Créer un module « collision »
  - Ajouter un champ collision_mask_t dans le type sprite_t

0	0	1	0	0	1	0	0
0	0	1	0	0	1	0	0
0	1	1	1	1	1	1	0
1	1	0	1	1	0	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	0	1	0	0	1	0	1
0	0	1	0	0	1	0	0

Test de collision

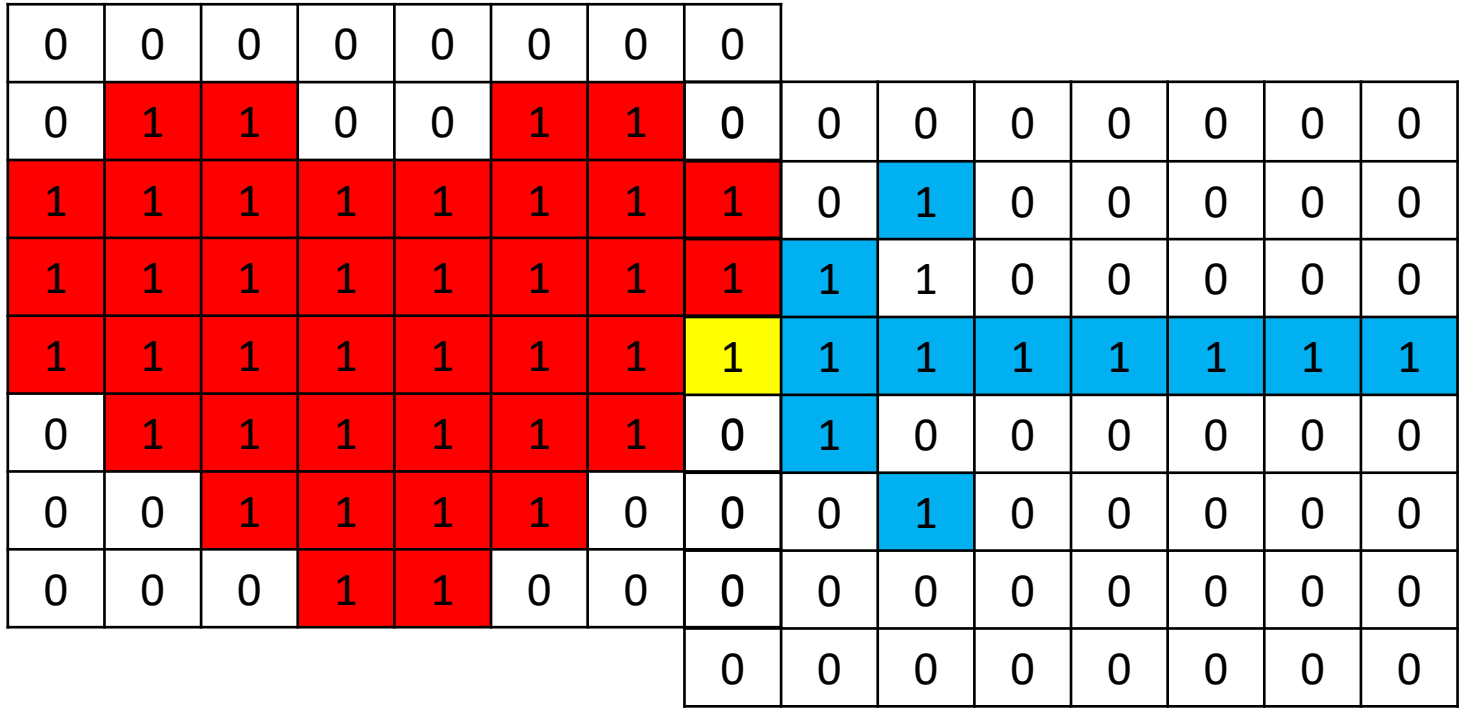
- On teste d’abord si les carrés englobants ont une intersection
  - Non, alors pas de collision !

0	0	0	0	0	0	0	0
0	1	1	0	0	1	1	0
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
0	1	1	1	1	1	1	0
0	0	1	1	1	1	0	0
0	0	0	1	1	0	0	0

0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	1	1	0	0	0	0	0
1	1	1	1	1	1	1	1
0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

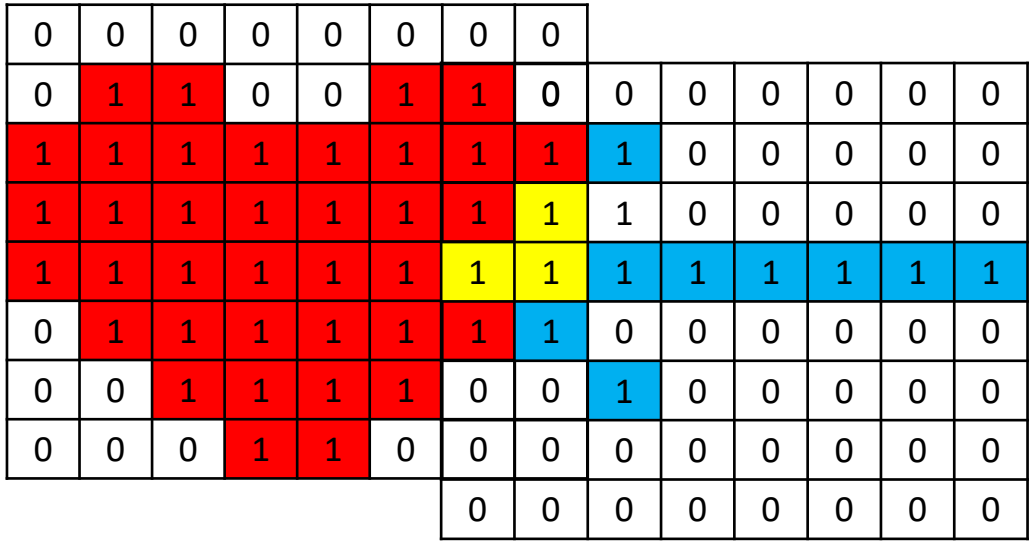
Test de collision

- On teste d'abord si les carrés englobants ont une intersection
  - Oui, alors il reste à vérifier si deux 1 se chevauchent



Test de collision

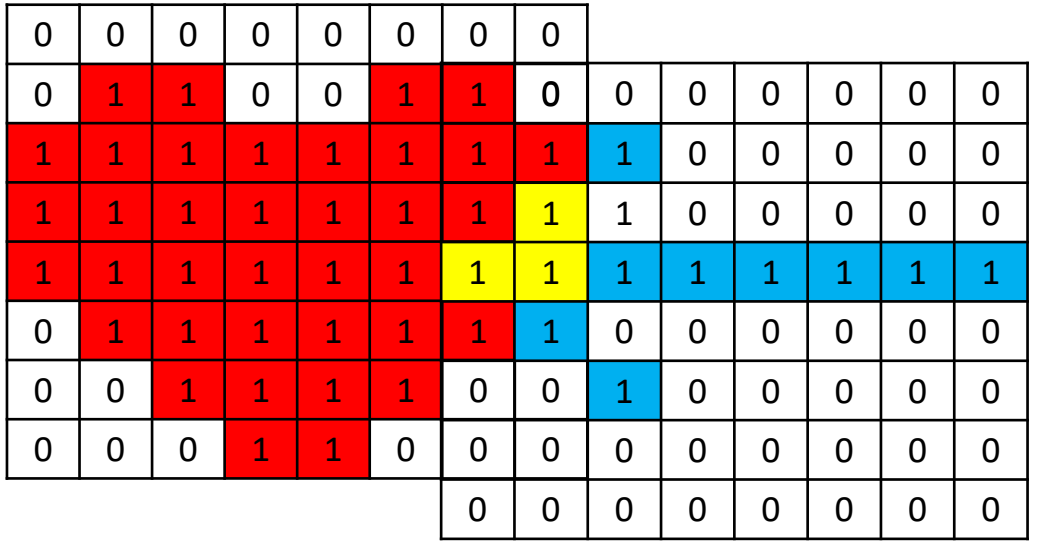
- Pour tester si le recouvrement partiel de deux masques conduit à superposer deux 1, il faut
  - Parcourir les lignes concernées (7 lignes ici) :
    - Lignes 1-7 pour le cœur
    - Lignes 0-6 pour la flèche
  - Tester le recouvrement des lignes du masque avec un AND
    - X_1 & X_2 est vrai si au moins un bit est à 1 à la même position dans les deux entiers



Test de collision

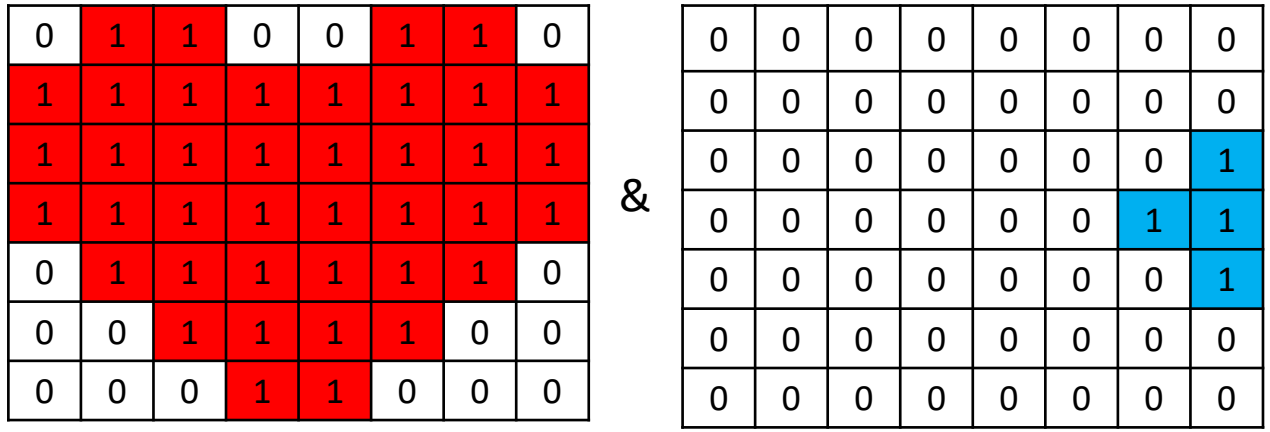
- Attention, il faut décaler (avec un *shift*) l'un des deux masques
- Par exemple: dans la collision ci-contre, on peut :
  - décaler la flèche de 6 bits

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0



Test de collision

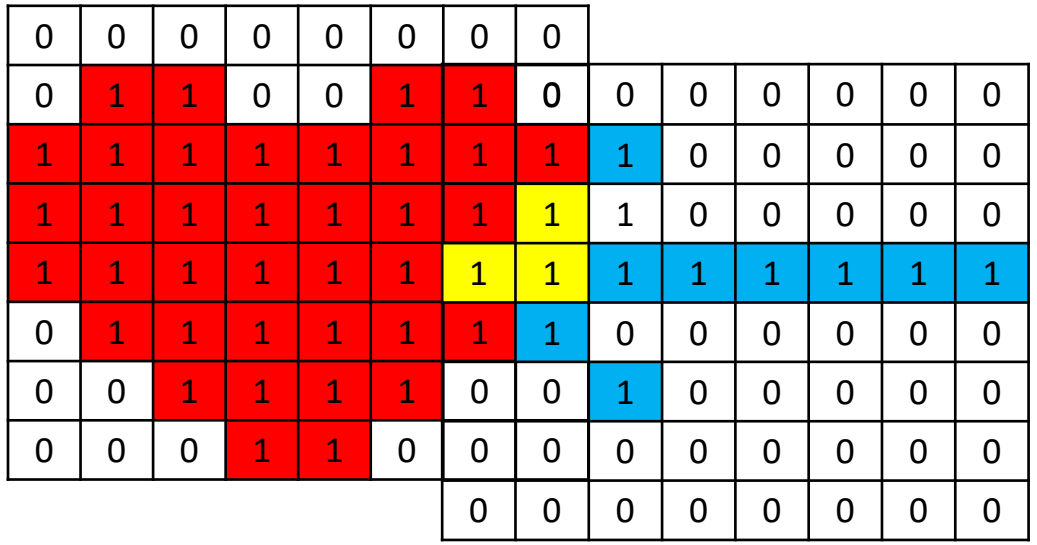
```
for (int i = 1; i < 7; i++) {
  bool collision =
    heart_mask[i] & (arrow_mask[i – 1] >> 6);
  if (collision)
    return TRUE;
}
return FALSE;
```

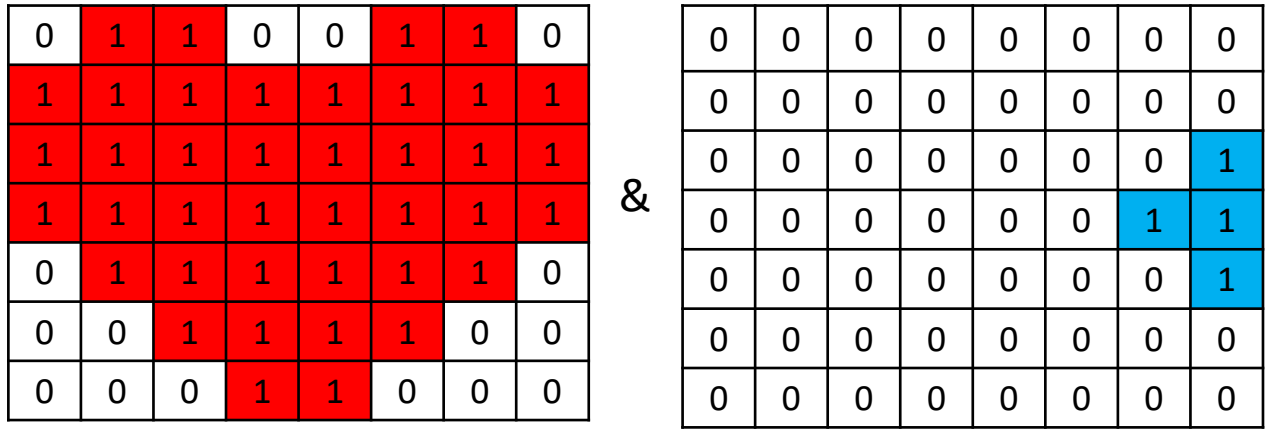


Test de collision

- Dans le cas présent (si faisait tous les tours de boucle) :

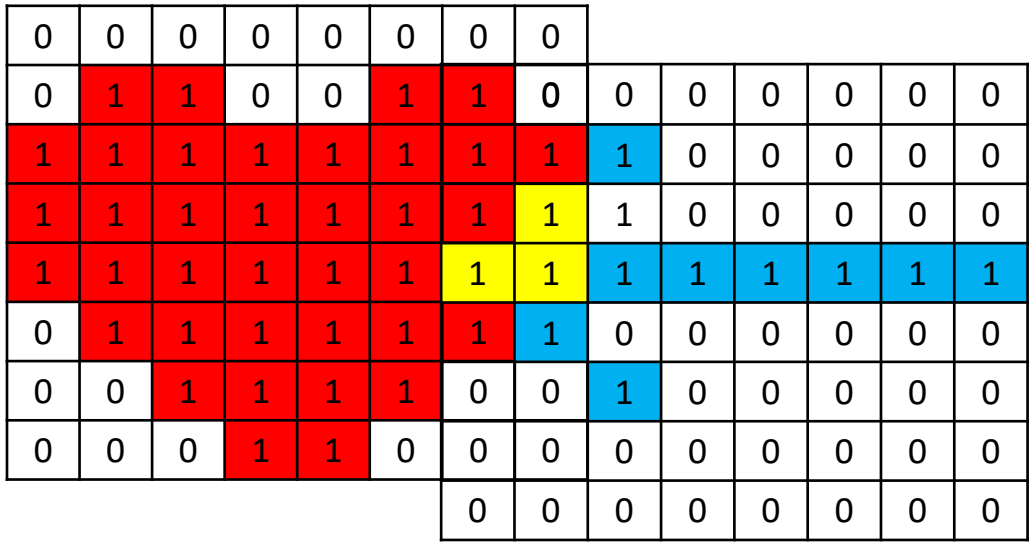
  01100110 & 00000000
  11111111 & 00000000
  11111111 & 00000001
  11111111 & 00000011
  01111110 & 00000001
  00111100 & 00000000
  00011000 & 00000000



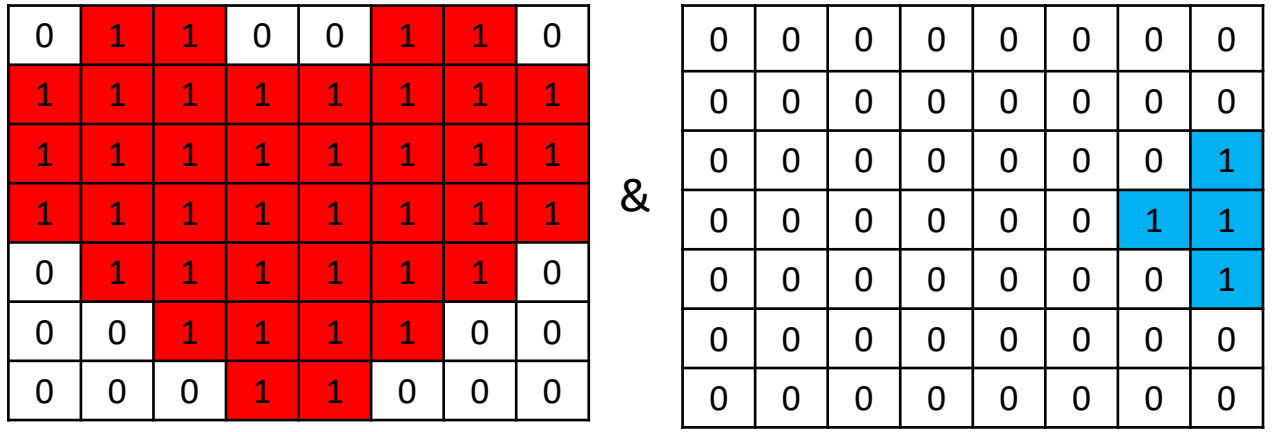


Test de collision

- Évidemment, on s'arrête lors de la troisième itération car le résultat du AND est différent de zéro



01100110 & 00000000
11111111 & 00000000
11111111 & 00000001



Test de collision

- Avec des masques stockés sous forme de tableaux de uint64_t, en un cycle on teste la collision de 64 pixels avec 64 autres !