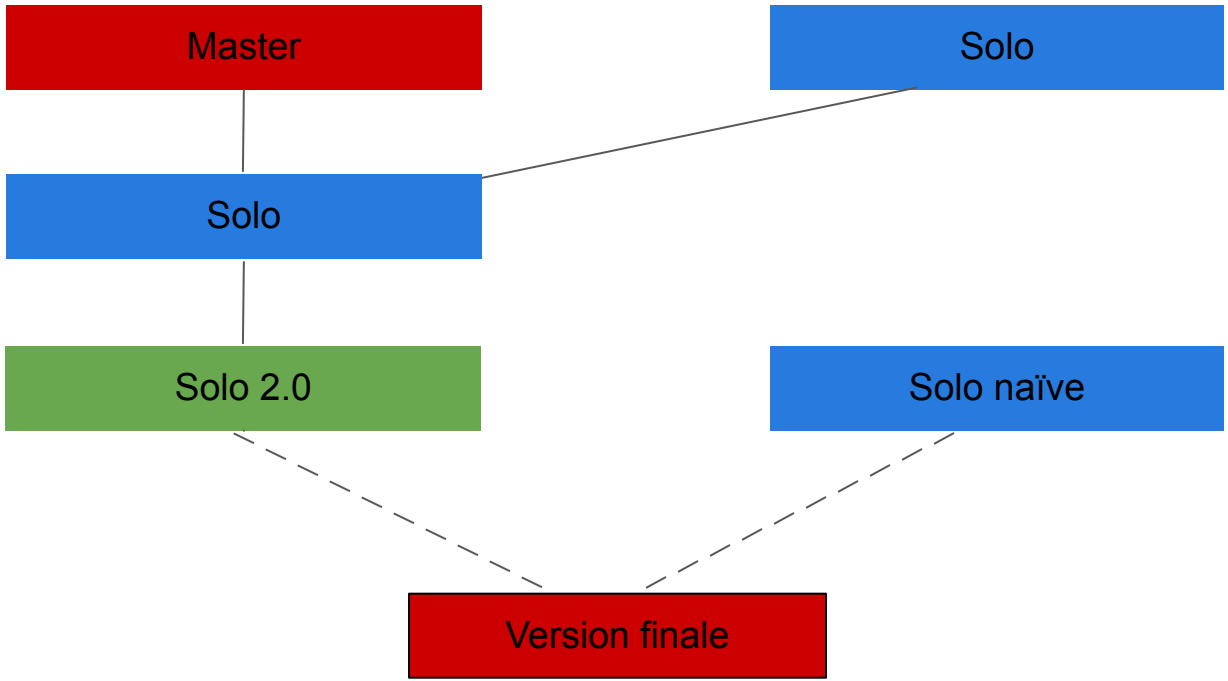
# Photo Slideshow

HashCode

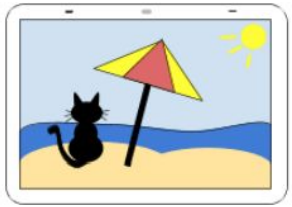
Groupe 3

# Historique



# Modélisation

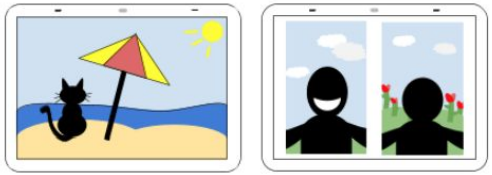
Picture.java



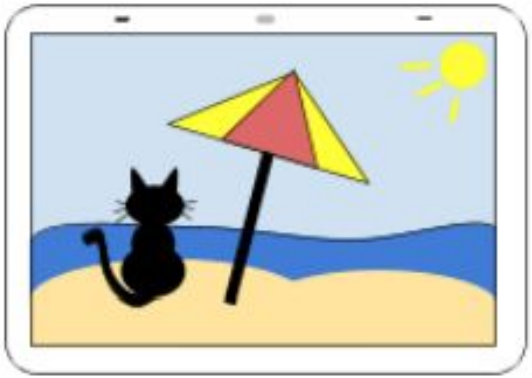
cat, beach sun

Main.java

Slide.java



# Picture



cat, beach sun

**Attributs:**

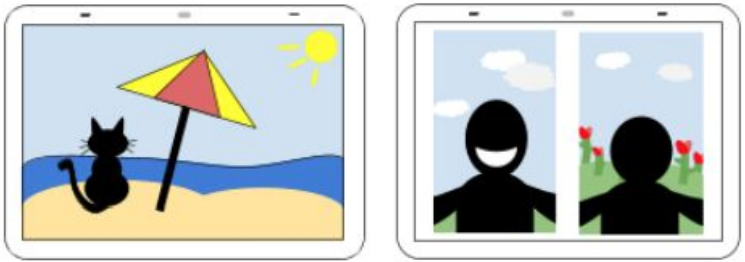
```
boolean isHorizontal;
int nb_tags;
HashSet<String> tags;
int id;
```

**Fonctions:**

```
void addTag(String tag)

// Obselète
int score_correspondance(Picture p)
```

# Slide



**Attributs:**

```
LinkedList<Picture> pictures;
```

**Fonctions:**

```
HashSet<String> compactTags()
int compare(Slide s2)
```

# Main

**Attributs:**

```
static LinkedList<Picture> all_pictures;

static LinkedList<Slide> verticals;
static LinkedList<Slide> horizontals;
static LinkedList<Slide> all_slides;
```

**Fonctions:**

```
static void matching_slides(Slide current)
static void advance_method()
static void naive_method()
```

# Parser/Writer

**[a_example.txt]**

---

4
H 3 cat beach sun
V 2 selfie smile
V 2 garden selfie
H 2 garden cat

**[a_example.sol]**

---

3
0
3
1 2

**Parser**

- Creation liste d'images

**Main**

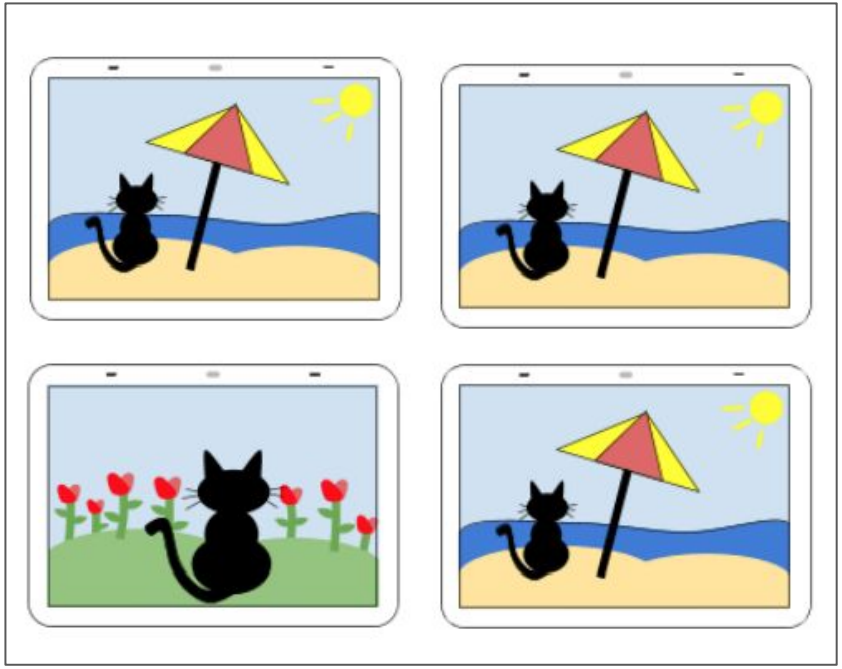
- Creation liste de slides
- Calcul d'une solution

**Writer**

- Création du fichier solution
- Écriture des slides

# 1ère stratégie : Fonctionnement

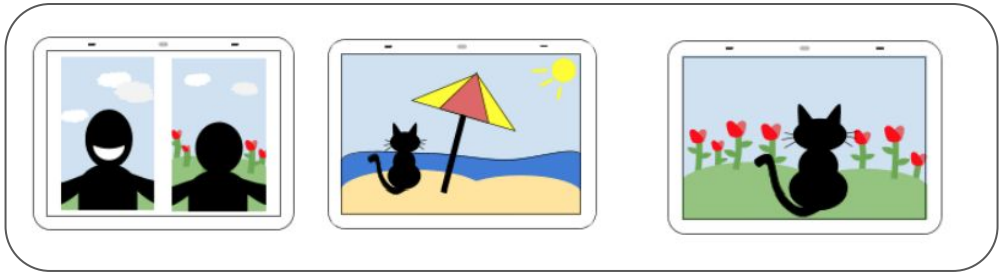
Slide Horizontale



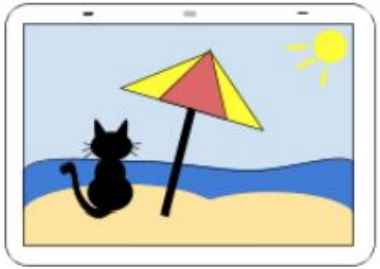
Slide Verticale

# 1ère stratégie : Fonctionnement

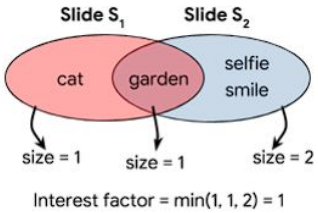
Liste All_Slide



1 Slide



Match la slide avec une autre slide




met le résultat du match à la suite de la liste

Résultat

Refait la même chose avec la dernière slide ajoutée

# Regrouper les photos verticales en fonction de leurs tags en commun

```
if(!p.isHorizontal){
    Picture tmp = null;
    int n = -1;
    for(Picture p2 : all_pictures){
        if(p.id != p2.id && !p2.isHorizontal && !taken.contains(p2)
        && Picture.score_score_correspondance(p, p2) > n){
            tmp = p2;
            n = Picture.score_score_correspondance(p, p2);
        }
    }
    if (tmp != null) {
        verticals.add(new Slide(p, tmp));
        taken.add(p);
        taken.add(tmp);
    }
```

Exemple avec le test C:

Photos ayant le moins de tags en commun: 1504 points

Photos ayant le plus de tags en commun: 1550 points

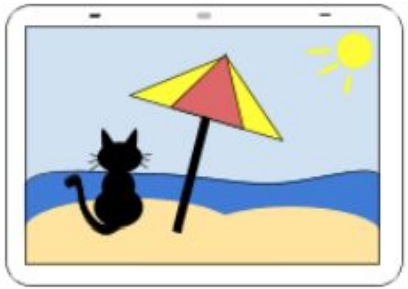
# Trier les slides par nombre de tags

```java
public static Comparator<Slide> comp = new Comparator<Slide>() {
    @Override
    public int compare(Slide o1, Slide o2) {
        return Integer.compare(o1.compactTags().size(), o2.compactTags().size());
    }
};
```

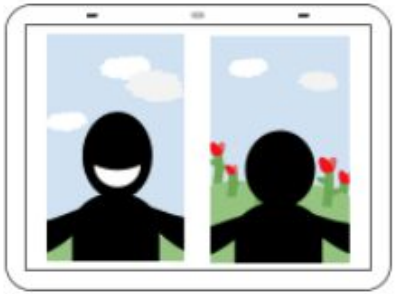
| | Liste non-triée | Liste triée |
|---|---|---|
| Exemple A | 2 points | 2 points |
| Exemple B | 12 points | 9 points |
| Exemple C | 1 508 points | 1 550 points |
| Exemple D | 195 091 points | 213 271 points |
| Exemple E | 112 468 points | 117 447 points |

# Commencer par une slide horizontale

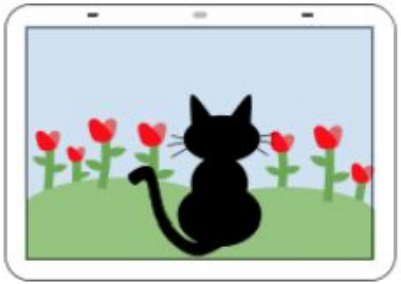
```
Slide first = horizontals.isEmpty() ? all_slides.getFirst() : horizontals.getFirst();
```



cat, beach sun



garden, selfie, smile

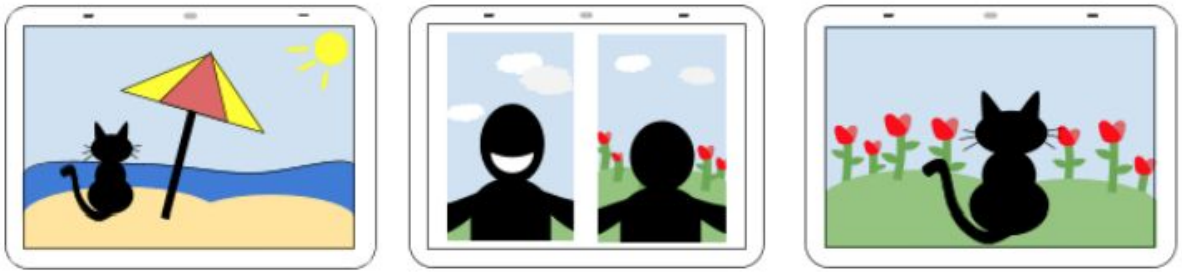


garden, cat

# 2ème stratégie : efficace

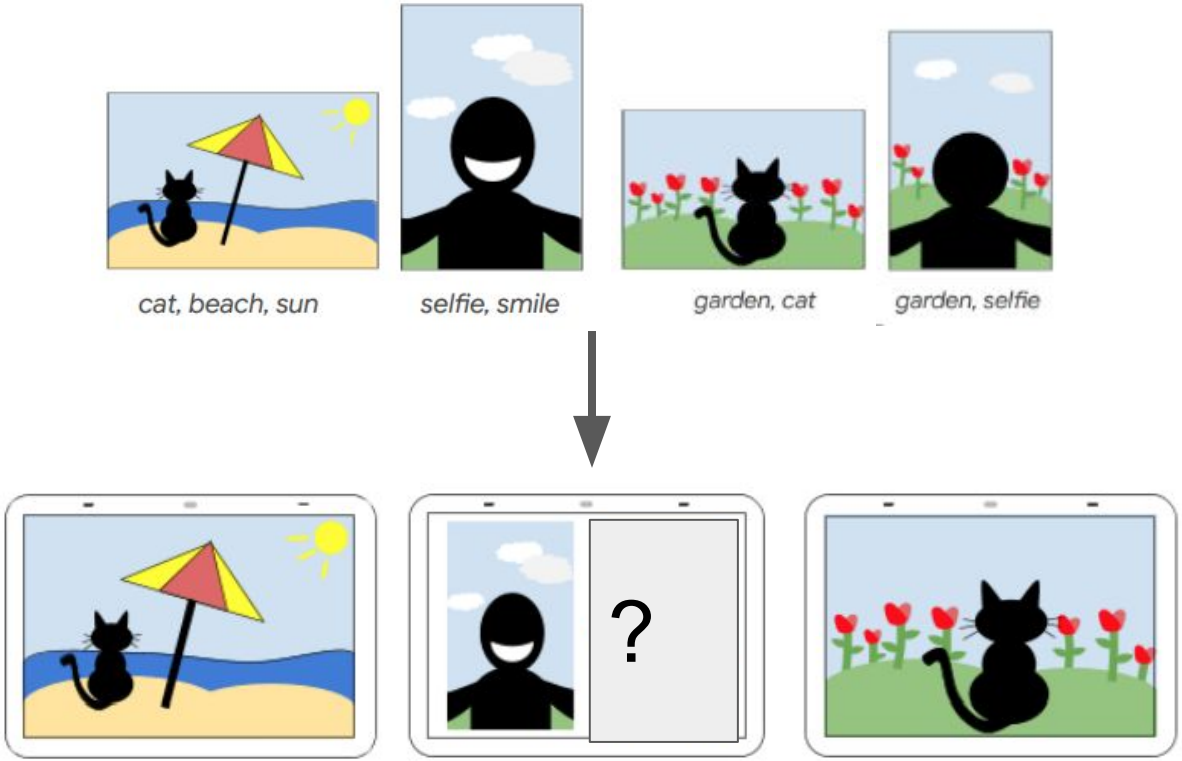
# 2ème stratégie : efficace


L’idée :



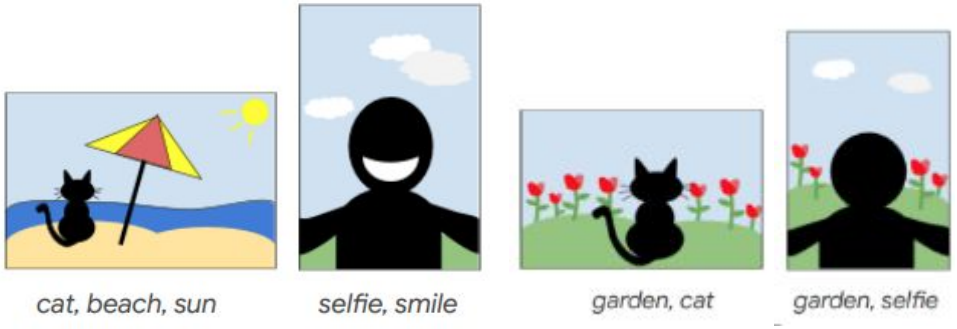
# 2ème stratégie : efficace

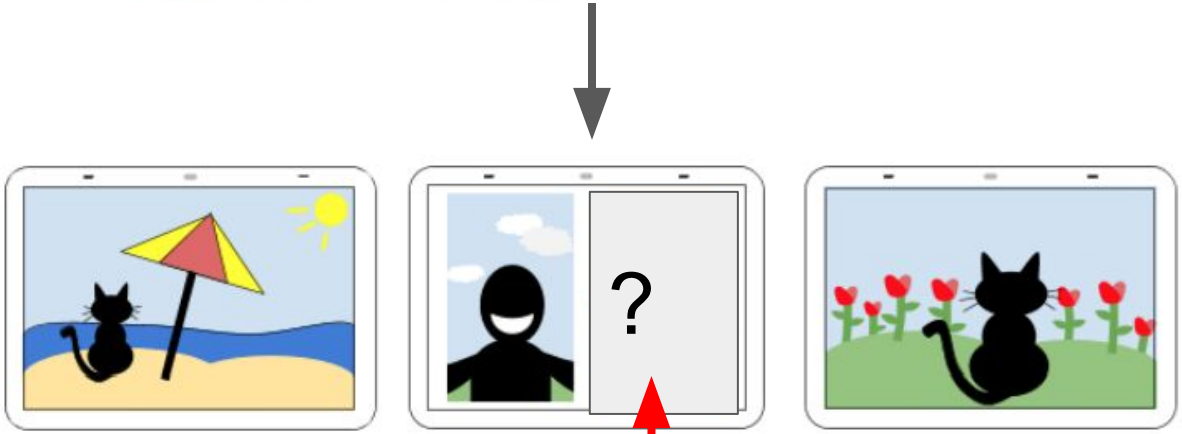
Mais si on avait



# 2ème stratégie : efficace

Mais si on avait

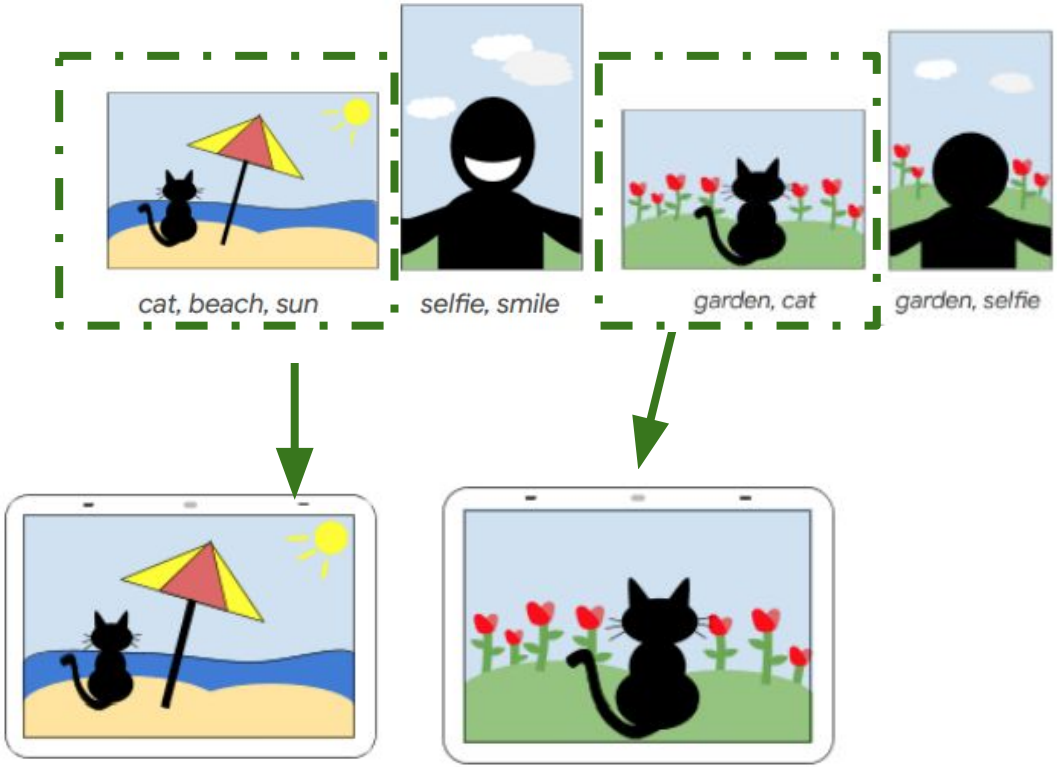



=> recherche d'un partenaire pour cette slide

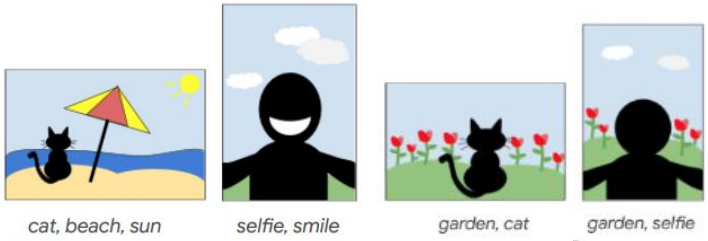
# 2ème stratégie : efficace

À la place,

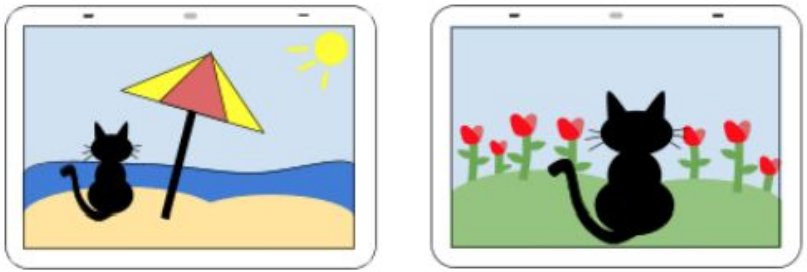


# 2ème stratégie : efficace

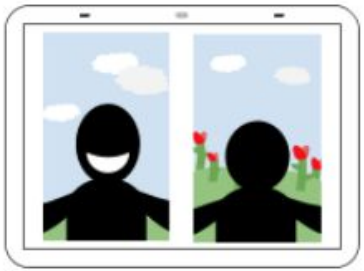

ou



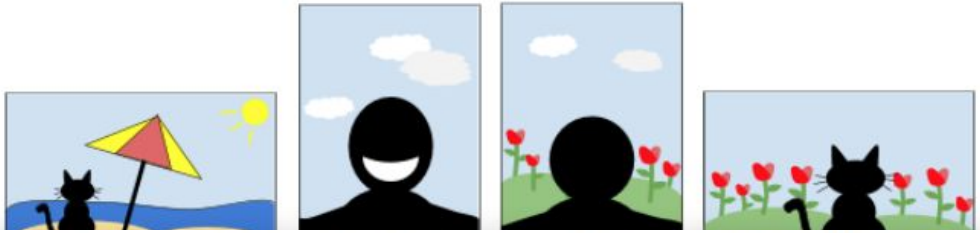

horizontales



verticales

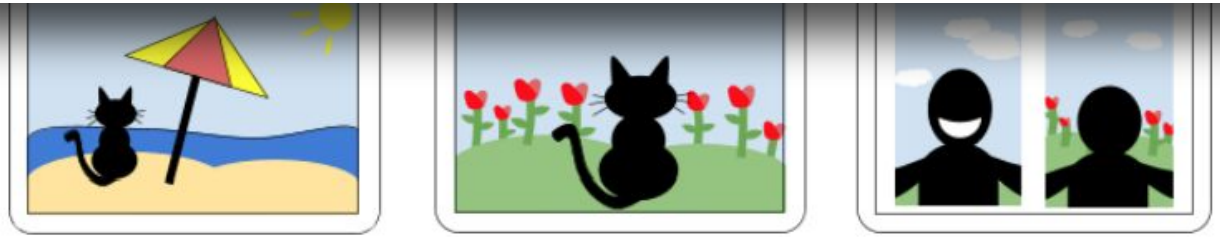


# 2ème stratégie : efficace

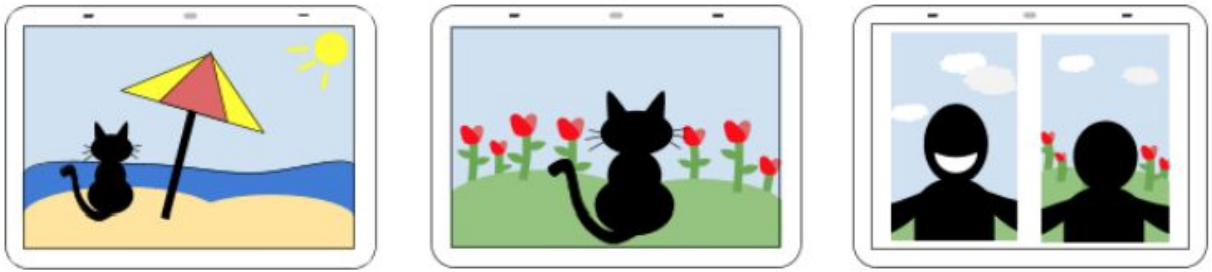


For two subsequent slides $S_i$ and $S_{i+1}$, the interest factor is the minimum (the smallest number of the three) of:

- the number of common tags between $S_i$ and $S_{i+1}$
- the number of tags in $S_i$ but not in $S_{i+1}$
- the number of tags in $S_{i+1}$ but not in $S_i$.



# 2ème stratégie : efficace

min(1,2,1) = 1 min(1,3,1) = 1 ⇒ 2

# 2ème stratégie : efficace

| | | |
|---|---|---|
| [a_example.txt] | Points : 2 | 0.07s user 0.03s system |
| [b_lovely_landscapes.txt] | Points : 12 | 4.53s user 0.28s system |
| [c_memorable_moments.txt] | Points : 152  // Trop peu | 0.21s user 0.04s system |
| [d_pet_pictures.txt] | Points : 195 091 | 2.47s user 0.22s system |
| [e_shiny_selfies.txt] | Points : 112 468 | 3.51s user 0.43s system |

Mais un temps plus que correct!

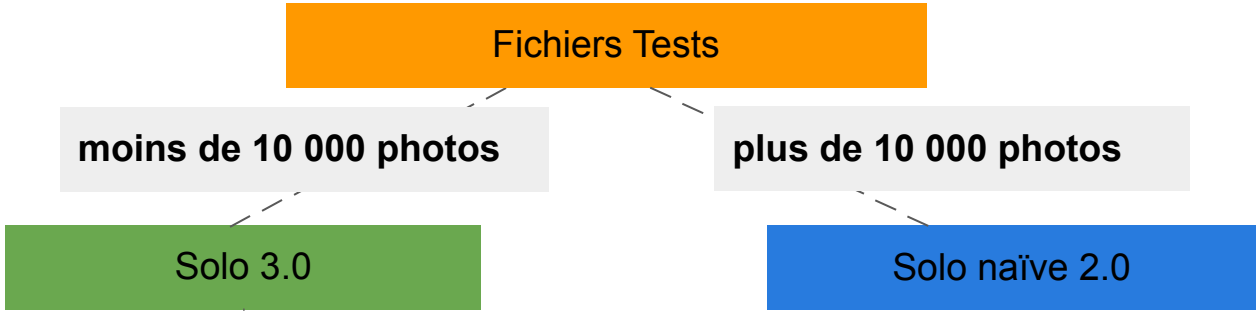
# 2ème stratégie : efficace

| | | |
|---|---|---|
| [a_example.txt] | Points : 2 | 0.07s user 0.03s system |
| [b_lovely_landscapes.txt] | Points : 12 | 4.53s user 0.28s system |
| [c_memorable_moments.txt] | Points : 152 // Trop peu | 0.21s user 0.04s system |
| [d_pet_pictures.txt] | Points : 195 091 | 2.47s user 0.22s system |
| [e_shiny_selfies.txt] | Points : 112 468 | 3.51s user 0.43s system |

⇒ Avec tri de toutes les slides par le nombre de tags d'une slide :

| | | | |
|---|---|---|---|
| [a_example.txt] | Points : 1 | ← - 1 | 0.07s user 0.03s system |
| [b_lovely_landscapes.txt] | Points : 9 | ← - 3 | 4.89s user 0.32s system |
| [c_memorable_moments.txt] | Points : 175 | ← + 23 | 0.25s user 0.04s system |
| [d_pet_pictures.txt] | Points : 213 271 | ← + 18 180 | 2.84s user 0.29s system |
| [e_shiny_selfies.txt] | Points : 117 447 | ← + 4 979 | 3.59s user 0.45s system |

# Stratégie finale : mélange des deux stratégies



# Stratégie finale : mélange des deux stratégies

| Fichiers tests | Nb de photos | Points | Temps |
|---|---|---|---|
| [a_example.txt] | 4 | 2 | 0.07s user 0.03s system |
| [b_lovely_landscapes.txt] | 80000 | 9 | 4.82s user 0.26s system |
| [c_memorable_moments.txt] | 1000 | 1 550 | 0.20s user 0.04s system |
| [d_pet_pictures.txt] | 90000 | 213 271 | 3.26s user 0.28s system |
| [e_shiny_selfies.txt] | 80000 | 117 447 | 3.87s user 0.32s system |