Enseignement de l’informatique sur la base de projets

Exemples et ressources

Barbar Jobstmann & Jamila Sam

Faculté Informatique et Communication

Plan

- Un peu de contexte
- Un exemple concret de projet
  - But
  - Code fourni
  - Détails du projet:
    - Représentation d'images
    - Partie 1: Couleurs
    - Partie 2: Filtres/Convolution
    - Partie 3: Graphes et plus court chemin
- Autres exemples de projets
- Retour d'expérience

Un peu de contexte

- Enseignement de l'informatique aux étudiants de première année EPFL
- Cours «Informatique, Calcul et Communication»
  - Fondements théoriques: algorithmes, théorie de la complexité, traitement du signal, compression de données etc. (Barbara)
  - Programmation (moi):
    - En tant que discipline
    - Mise en pratique des concepts théoriques
- Cours «Introduction à la programmation» pour les étudiants en informatique et communication

Pourquoi des projets?

- Démonstration qu'avec un outillage simple en programmation , il est déjà possible de résoudre de «vrais problèmes»

  → renforce la motivation

- Travail collaboratif (groupes de 2) et de plus longue haleine que des exercices scolaires
- Illustration concrète de l'importance de toutes les phases de l'élaboration d'un programme informatique (conception, implémentation, tests)

Contraintes

- Durée limitée (2 à 4 semaines)
- Nombre limité de lignes de code (~300-400 lignes)
- Outillage basique: instructions itératives et conditionnelles, tableaux (et structures), fonctions

«mini-projets»

Plan

- Un peu de contexte
- Un exemple concret de projet
  - But
  - Code fourni
  - Détails du projet:
    - Représentation d'images
    - Partie 1: Couleurs
    - Partie 2: Filtres/Convolution
    - Partie 3: Graphes et plus court chemin
- Autres exemples de projets
- Retour d'expérience

But

- Afficher des images sans distortion sur différents média (téléphones, tablettes,...)
- Nécessite de redimensionner sur la base du contenu

Redimensionnement d'image

Seam Carving:
«Content-Aware Image Resizing»

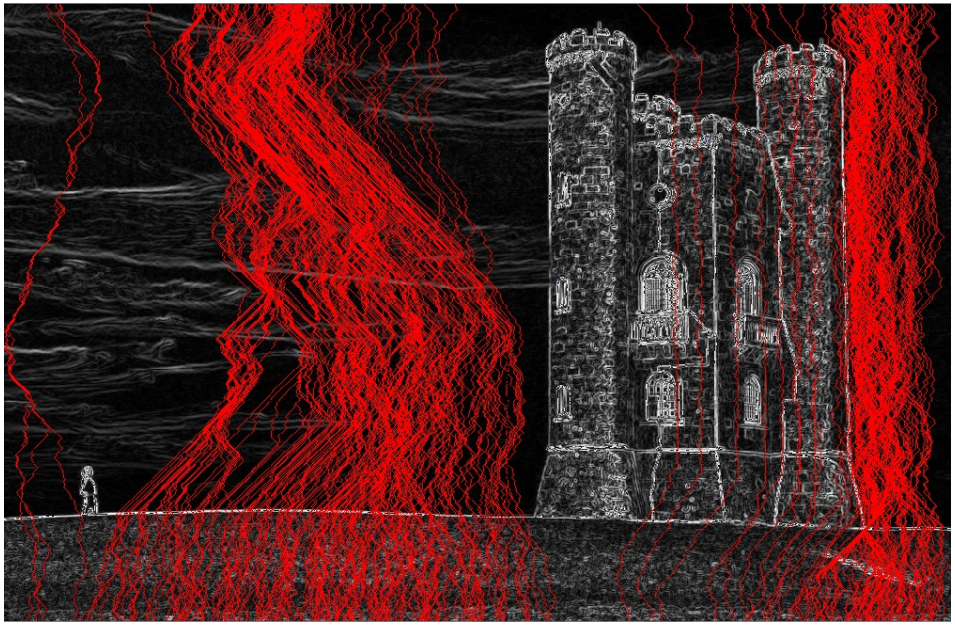
Demo: https://www.aryan.app/seam-carving/

Idée générale

Repéter jusqu'à la taille voulue:

1. Convertir en niveaux de gris
2. Lisser et détecter les contours
3. Calculer et supprimer un chemin («seam») de pixels

Exemple avec 300 chemins



Techniques sous-jacentes

Repéter jusqu’à la taille voulue

1. Convertir en niveau de gris
   - Représentation de la couleur
2. Lisser et détecter les contours
   - Application de filtres (convolution)
3. Calculer et supprimer un «seam»
   - Graphes et plus court chemin

Matériel fourni

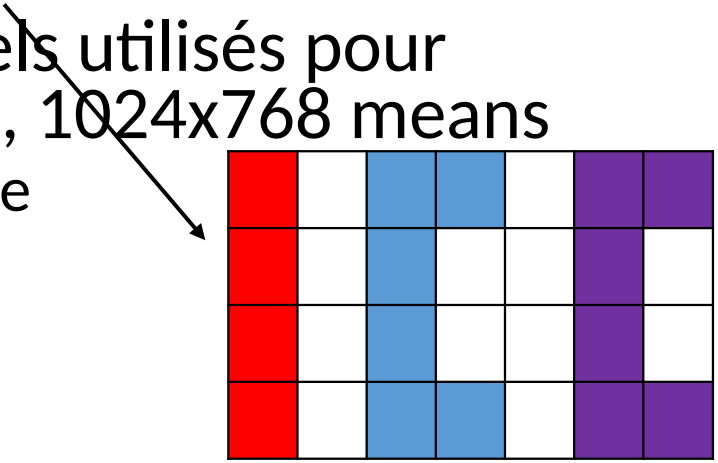
1. Une archive .zip incluant
   - Des fichiers en formats .jpg or .png:
     → images en entrée ou sortie pour tester
   - Quelques fichiers utilitaires
     → matériel pour lire et écrire des images
   - Du matériel de test

2. Un énoncé détaillé

https://iccsv.epfl.ch/mini-project/seam-fr.pdf

Représentation d'images

- Image digitale = grille de pixel (picture elements)
- Resolution = nombre de pixels utilisés pour représenter une image, e.g., 1024x768 means
  - 1024 pixels de gauche à droite
  - 768 pixels de haut en bas



- Dans ce tableau vous représenterez les images comme des tableaux à deux dimensions

```
typedef vector< vector <int> >    RGBImage;  // Color image
typedef vector< vector <double> > GrayImage; // Gray  image
```

Tâche 1: Couleurs

- Un pixel peut être simplement représenté par une couleur sous la forme d'une valeur en format RGB (Red-Green-Blue).
- Le modèle de couleur RGB est un modèle additif où des composantes rouge, verte et bleue s'additionnent pour former une large palette de couleurs possibles
  - Chaque valeur RGB est représentée par la **concaténation** de 3 octets, **un octet (8 bits) pour chaque composante couleur:**
    rrrrrrrrggggggggbbbbbbbb
  - Chaque composante couleur peut avoir une intensité entre 0 (min) et 2^8-1=255 (max) en décimal, ou de façon équivalente, entre 00 et ff en hexadécimal.
- **Votre tâche**: extraire les composantes rouge, verte, et bleue pour d'un pixel pour obtenir une valeur en niveau de gris.

Operateurs binaires

- **Shift gauche** (val << n): décale de n positions vers la gauche tous les bits de val, insère des zéros à droite:
  - **`0b010111 << 5`** gives **`0b010111`**`00000`
- **Shift droit** (val >> n): décale de n positions vers la droite tous les bits de n, insère des 0s ou 1s sur la gauche (selon le signe):
  - **`0b0101`**`11 >> 3` gives **`0b010`**
- **et binaire** (val1 & val2): combine les bits de val1 et val2 à chaque position **and** (gate).

```
   0b010011
 & 0b000101
 ----------
   0b000001
```

Masquage

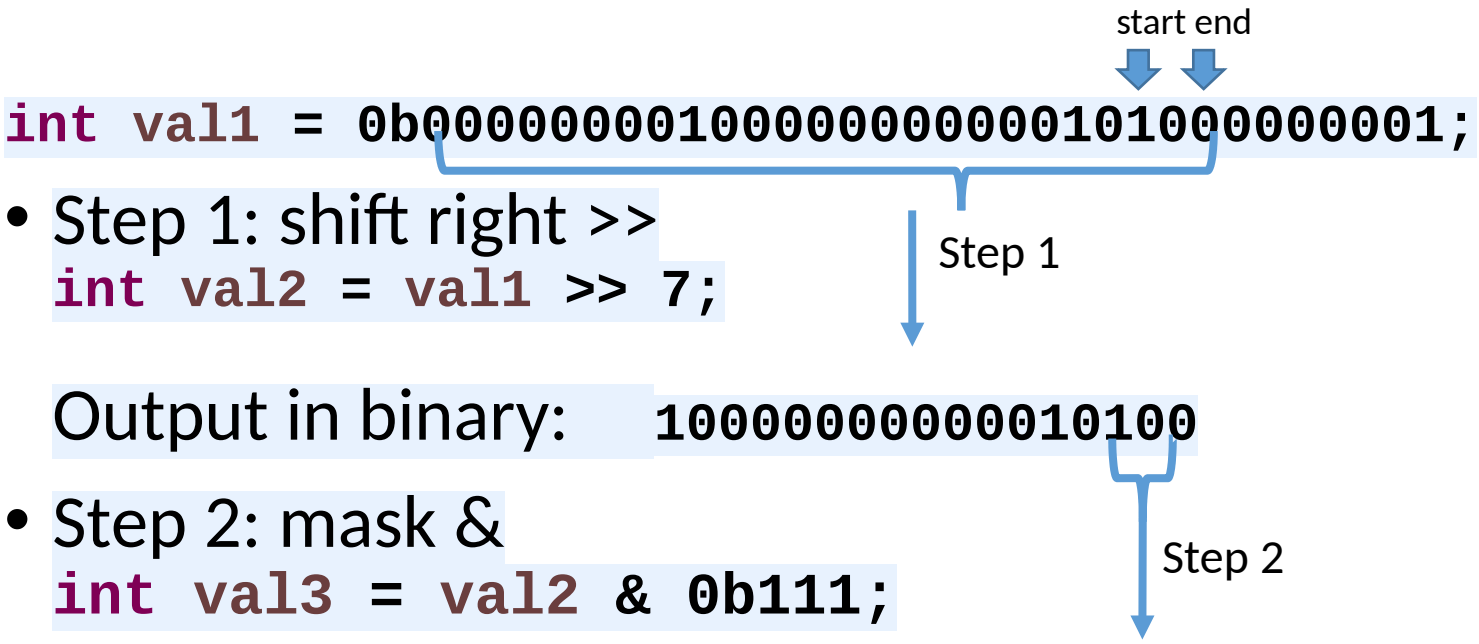
```
int val2 = 0b1000000000000101;
int val3 = val2 & 0b111;
```

0b1000000000000101
0b0000000000000111 Filled with leading zeros

0b0000000000000101 Bitwise And

Output in binary: **101** Without leading zeros

Extraction de bits



- Step 1: shift right >>

```
int val2 = val1 >> 7;
```

Output in binary: `1000000000010100`

- Step 2: mask &

```
int val3 = val2 & 0b111;
```

Output in binary: `100`

Fusion

```
int val1 = 0b1010;
int val2 = 0b10000001;
```

Goal: **0b101010000001**

- Step 1: shift left <<

```
int val1_sl = val1 << 8;
```

Output in binary: **101000000000**

- Step 2: add

```
int val3 = val1_sl + val2;
```

Output in binary: **101010000001**

Tâche 1: Fonctions à coder

- **Convertir un pixel**: for each color convert RGB-value to double value between 0.0 (min: intensity 0) and 1.0 (max: intensity 255)
  - `double get_red  (int rgb)`
  - `double get_green(int rgb)`
  - `double get_blue (int rgb)`
  - `double get_gray (int rgb)`
  - `int get_RGB(double red, double green, double blue)`
  - `int get_RGB(double gray)`
- **Convertir une image**
  - `GrayImage to_gray(const RGBImage  &cimage)`
  - `RGBImage  to_RGB (const GrayImage &gimage)`

Partie 2: Filtres

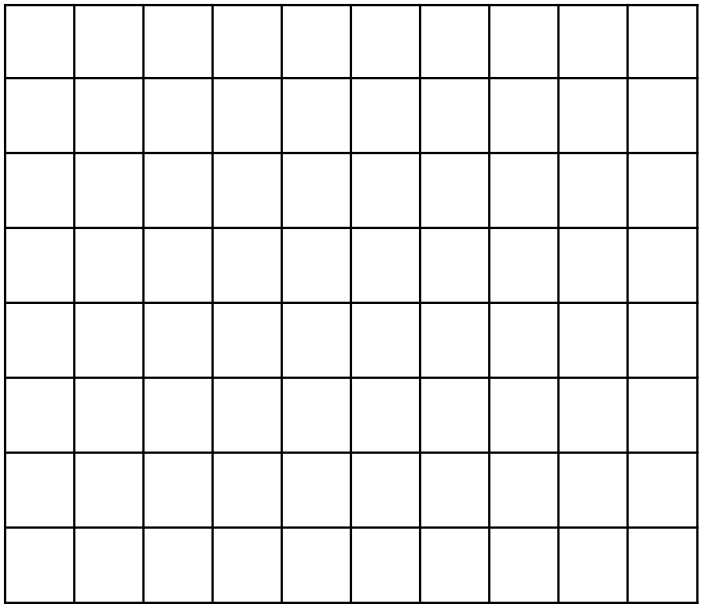
Original

Smoothed image

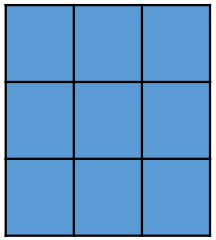
Edge detection (Sobel)

Idée générale

- Calculer une nouvelle valeur pixel sur la base de la valeur de ses voisin, e.g., lissage = moyenne des pixels voisins
- Opérateur mathématique : *convolution*



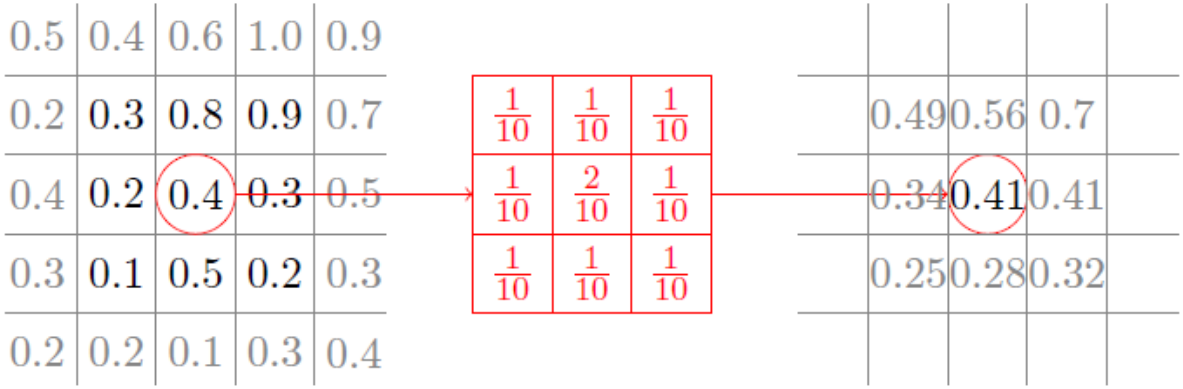
Image



Kernel

```
typedef vector< vector <double> > Kernel;
```

Exemple



$$0.3\frac{1}{10} + 0.8\frac{1}{10} + 0.9\frac{1}{10} + 0.2\frac{1}{10} + 0.4\frac{2}{10} + 0.3\frac{1}{10} + 0.1\frac{1}{10} + 0.5\frac{1}{10} + 0.2\frac{1}{10} = 0.41$$

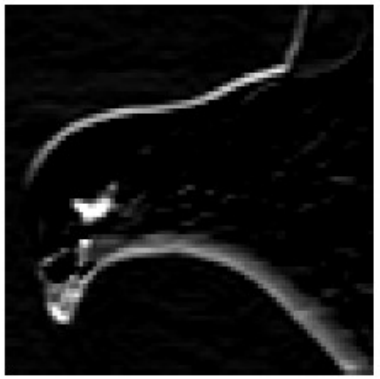
Kernels intéressants

$$[1] \qquad \begin{bmatrix} \frac{1}{10} & \frac{1}{10} & \frac{1}{10} \\ \frac{1}{10} & \frac{2}{10} & \frac{1}{10} \\ \frac{1}{10} & \frac{1}{10} & \frac{1}{10} \end{bmatrix} \qquad \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Identity

Smooth

SobelX:
to detect
vertical
edges



SobelY:
to detect
horizontal
edges

Détection de contour (dans ce projet) =
Norme Euclidienne de SobelX et SobelY

Tâche 2: fonctions à coder

- `void clamp(int& val, int max)`
- `GrayImage filter(const GrayImage &gray, const Kernel &kernel)`
- `GrayImage smooth(const GrayImage &gray)`
- `GrayImage sobelX(const GrayImage &gray)`
- `GrayImage sobelY(const GrayImage &gray)`
- `GrayImage sobel (const GrayImage &gray)`

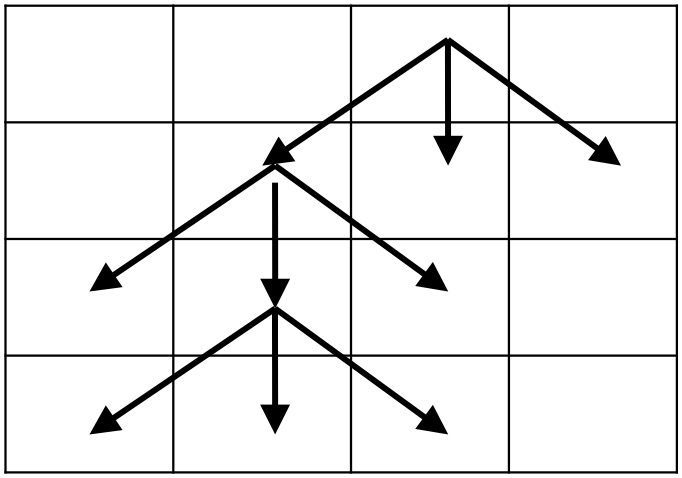
Partie 3: Graphe et plus court chemin

Étapes:

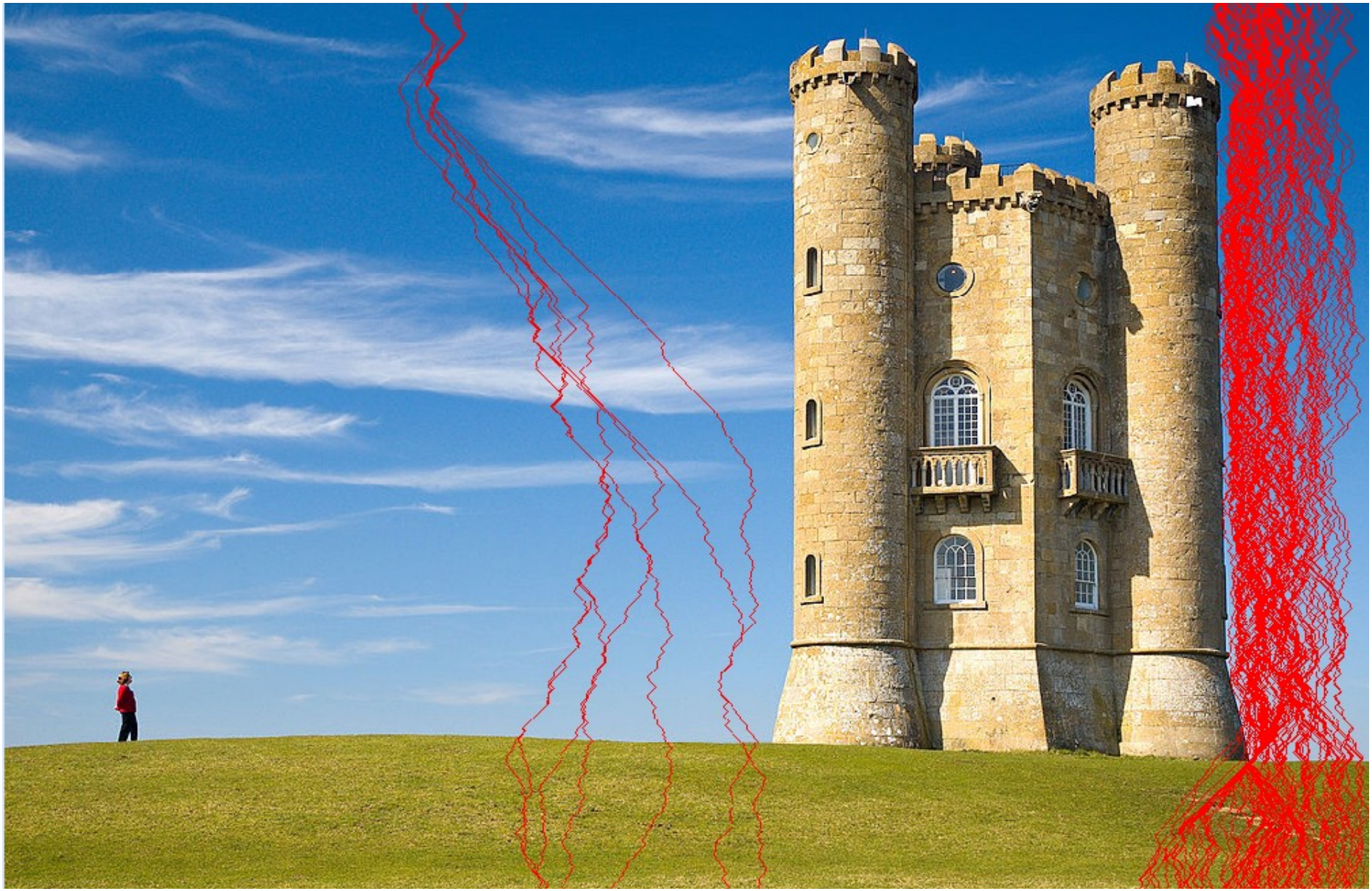
- Conversion d'une image en graphe
- Calcul du plus court chemin dans un graphe
- Conversion de chemin du graphe en «seam» (chemin de pixels) dans l'image
- Suppression itérative des «seam» jusqu'à obtenir la taille voulue.

Idée générale

- But: réduire la largeur d'une image d'un pixel
  - Trouver un chemin de pixels «peu pertinents» de haut en bas de l'image (peu ou pas de pixels de contour)
  - Les pixels du chemin doivent être "connectés" (d'une ligne à l'autre un seul pixel d'écart)



Exemple : écart de +/-1 entre chaque pixel

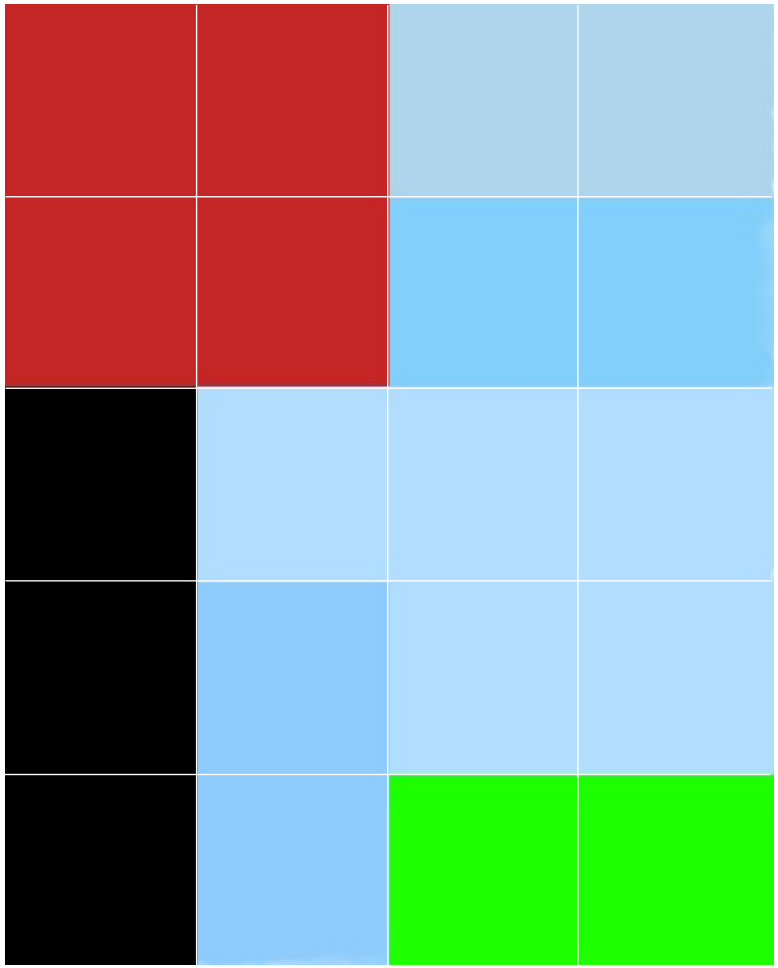


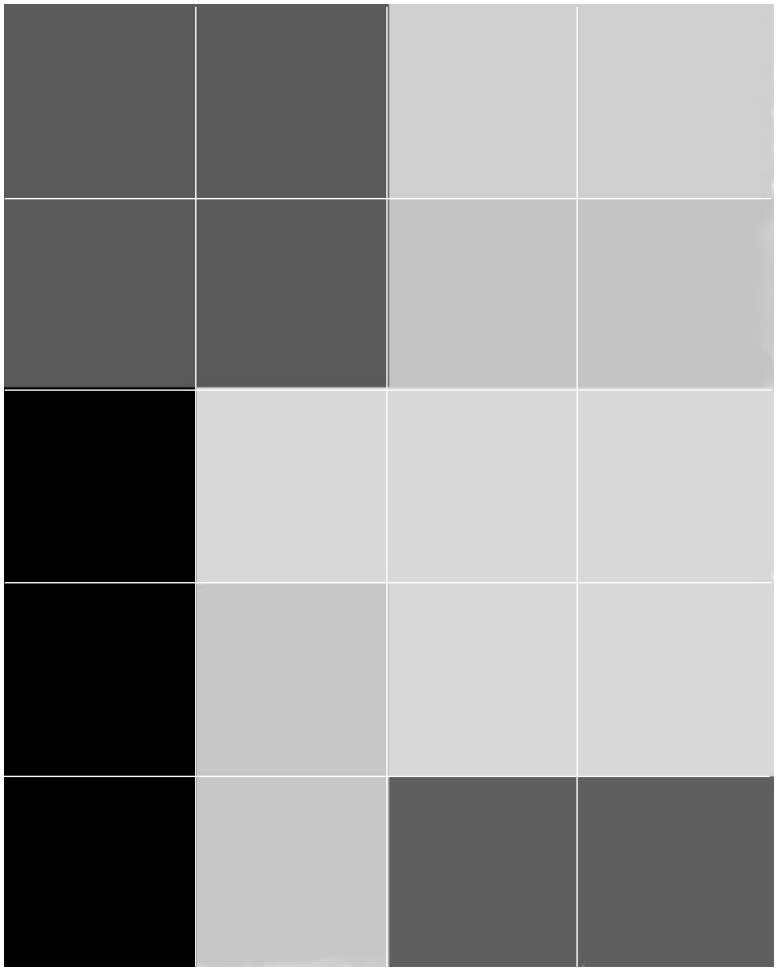
Exemple (après suppression des «seam»)

Exemple: avec des écarts de +/- 20

Exemple: avec des écart de +/- 20 (suppression)

Exemple: Originale/ Gris



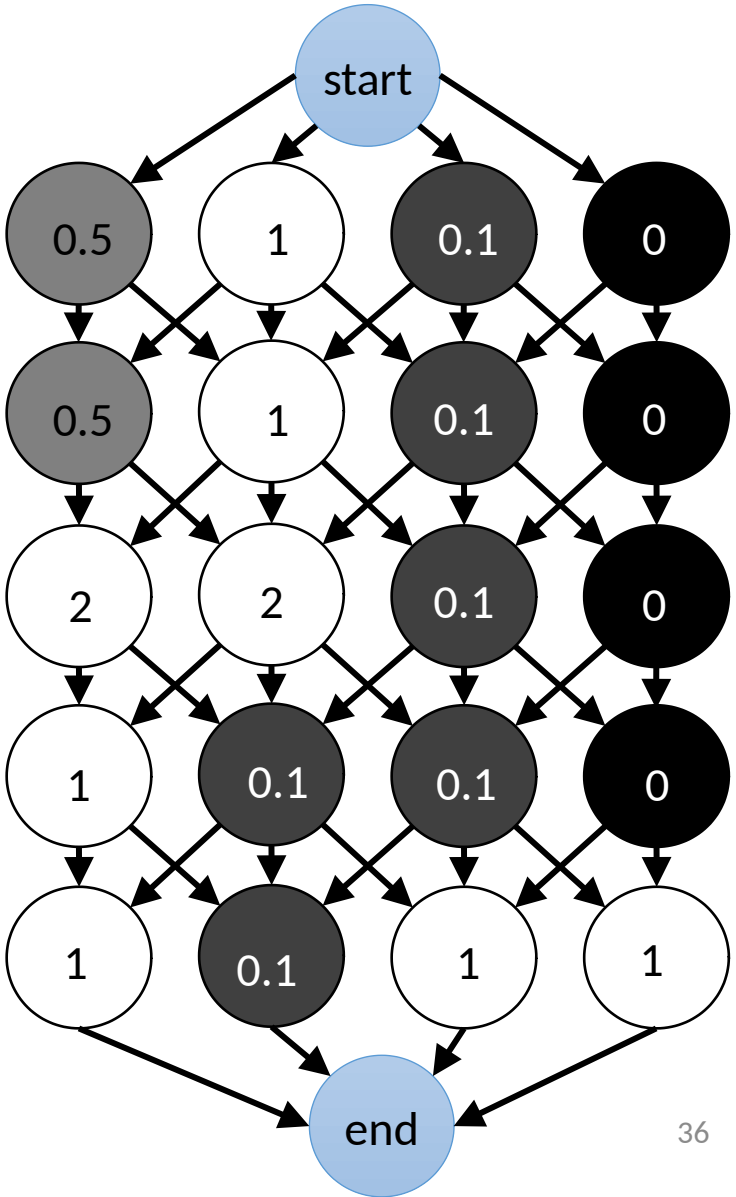


Exemple: Approximation de Sobel (simplification)

0.5	1	0.1	0
0.5	1	0.1	0
2	2	0.1	0
1	0.1	0.1	0
1	0.1	1	1

Exemple: de l’image au graphe

0.5	1	0.1	0
0.5	1	0.1	0
2	2	0.1	0
1	0.1	0.1	0
1	0.1	1	1



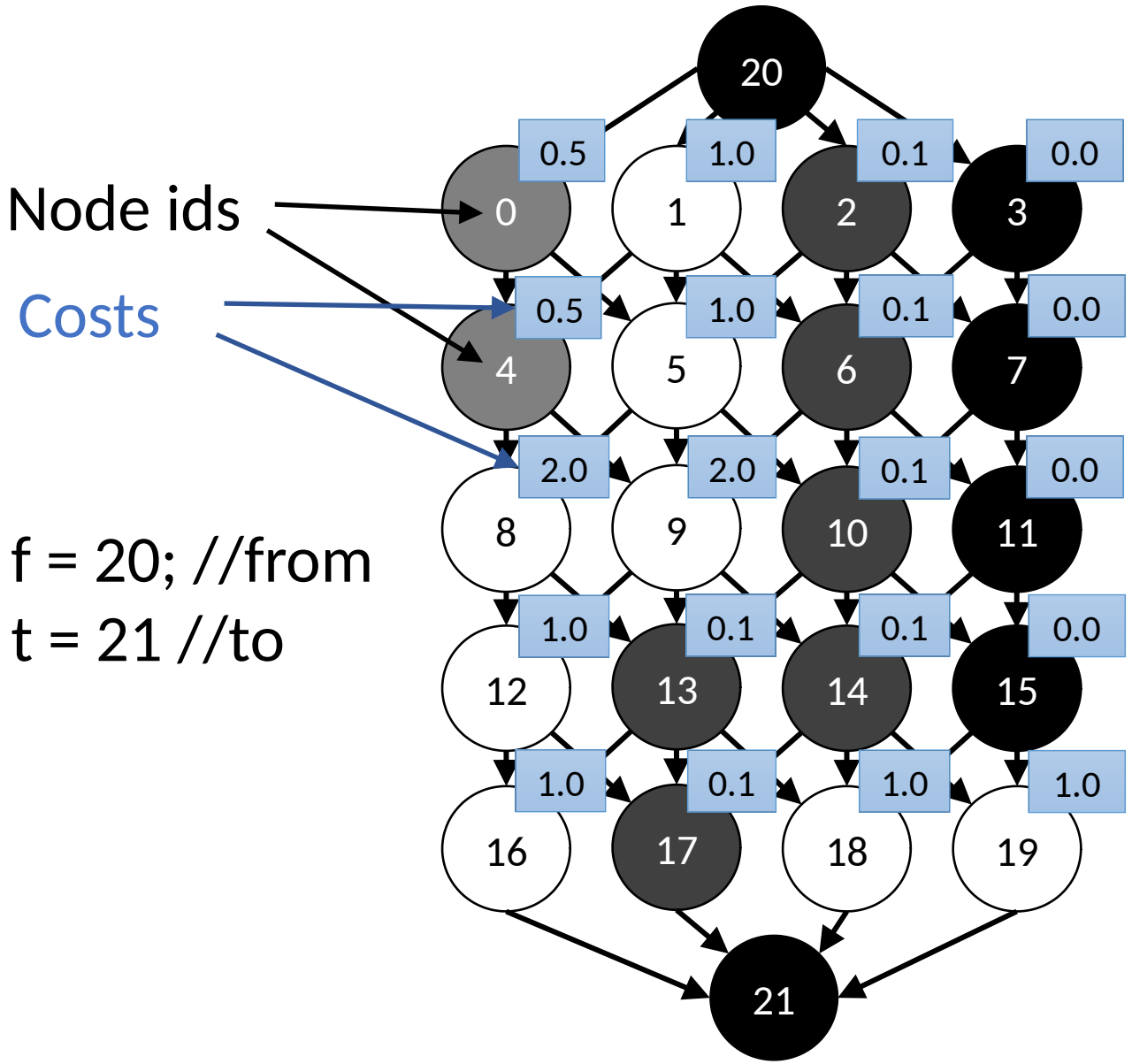
Plus court chemin

Étant donné

- un graphe avec des coûts,
- un noeud “from” (f) et un moeud “to” (t),

But: calculer le plus court chemin (celui de moindre coût) de f à t

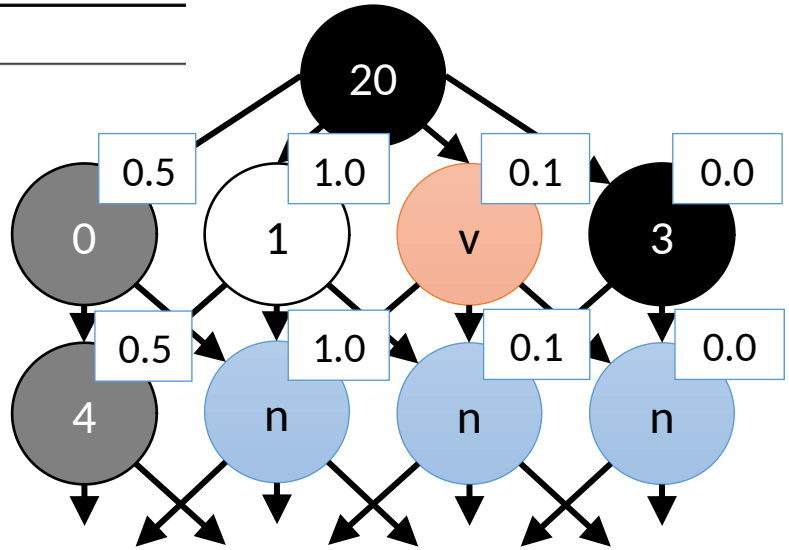
Graphe avec coûts



Algorithme

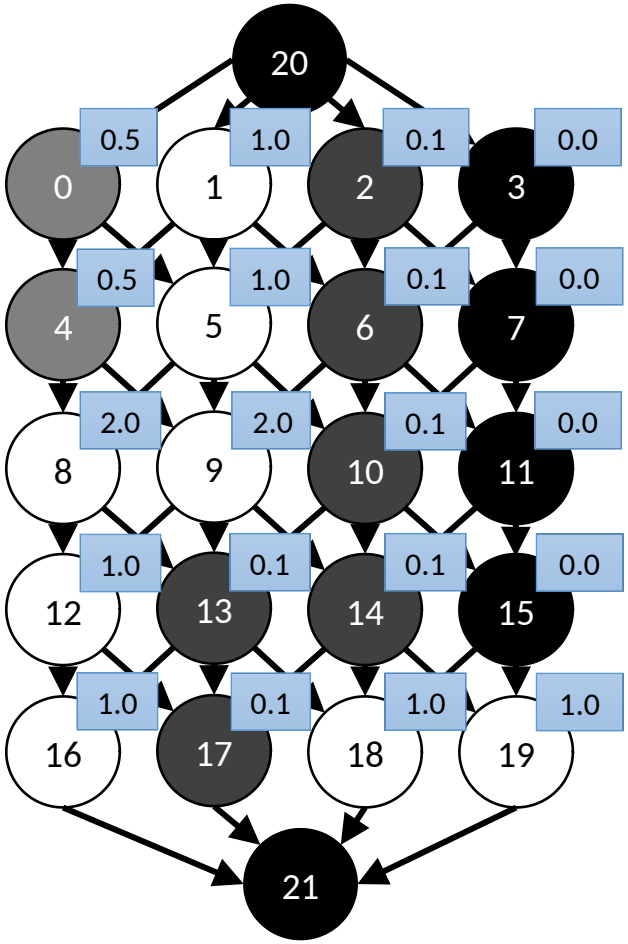
Algorithm 1 Algorithme de Dijkstra modifié

```
for all v ∈ vertices do
    distance(v) ← ∞
    bestPredecessor(v) ← null
end for
distance(from) ← cost(from)
modified ← true
while modified do
    modified ← false
    for all v ∈ vertices do
        for all n ∈ successors(v) do
            if distance(n) > distance(v) + cost(n) then
                distance(n) ← distance(v) + cost(n)
                bestPredecessor(n) ← v
                modified ← true
            end if
        end for
    end for
end while
```

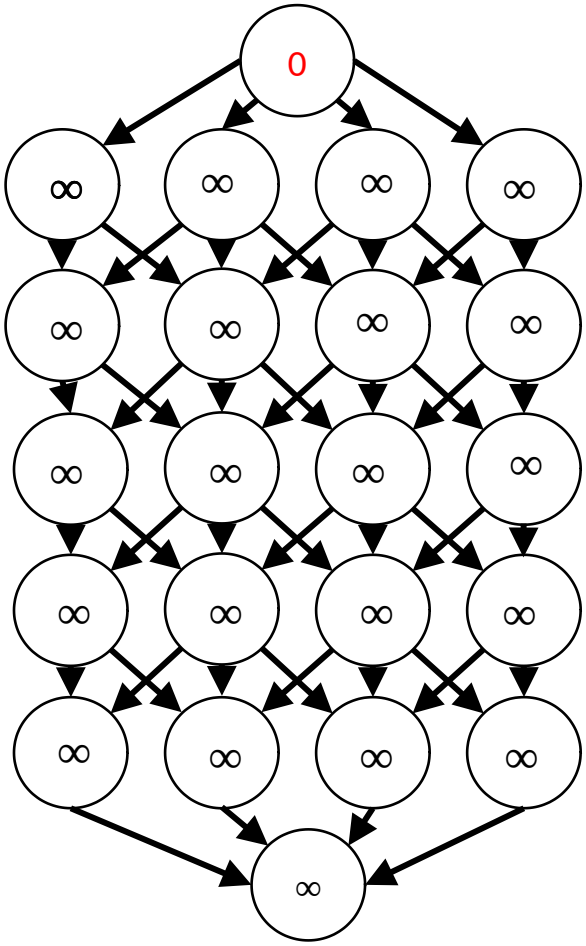


Itération 0

Input graph: from=20, to=21

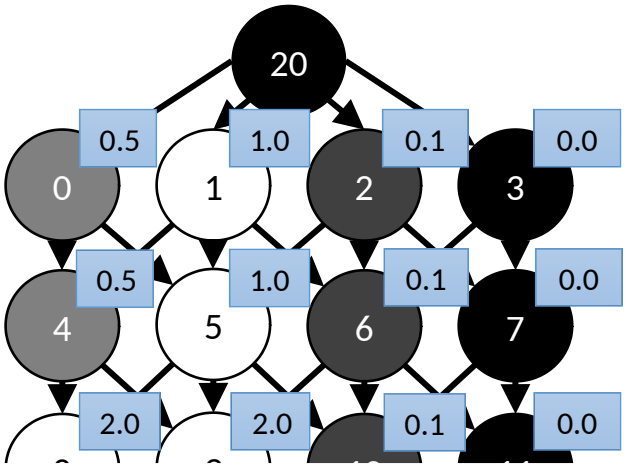


distance from state 20 in 0 steps

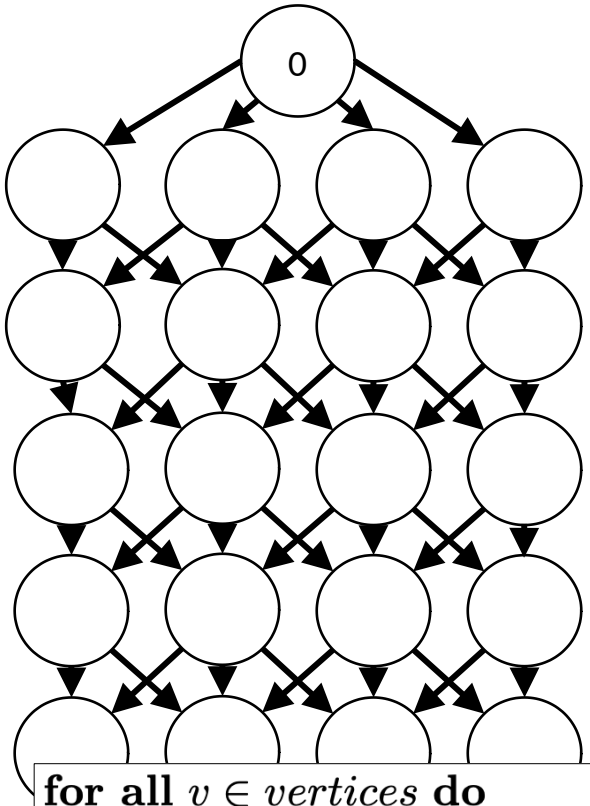


Itération 1

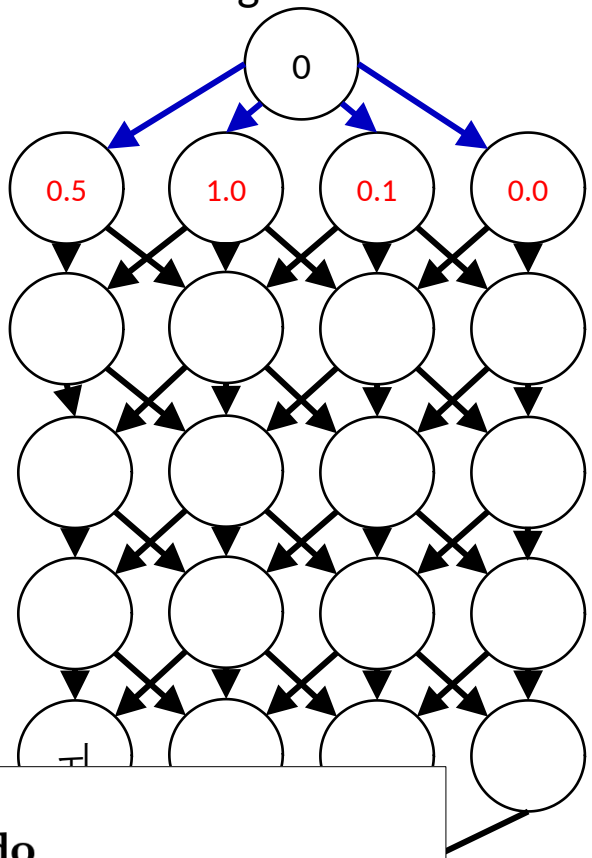
Input graph



distances in iteration 0



distances in iteration 1
blue edges = bestPredec.



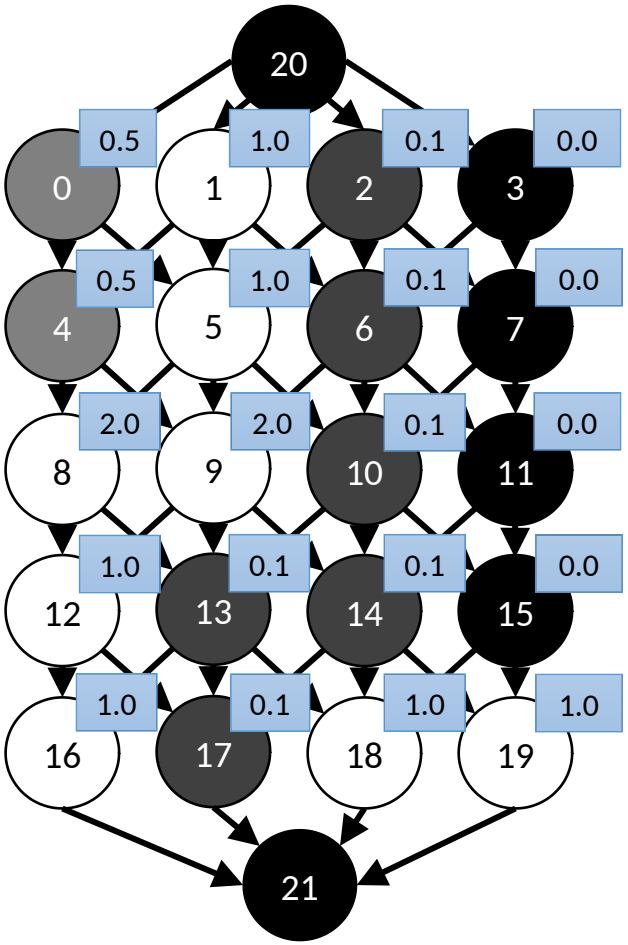
Update:
For all nodes v,

- go through all successors n of v and check if a path going through v is shorter (less expensive) than distance of n from previous iteration
- if yes, update distance of n and set bestPredecessor of n to v.

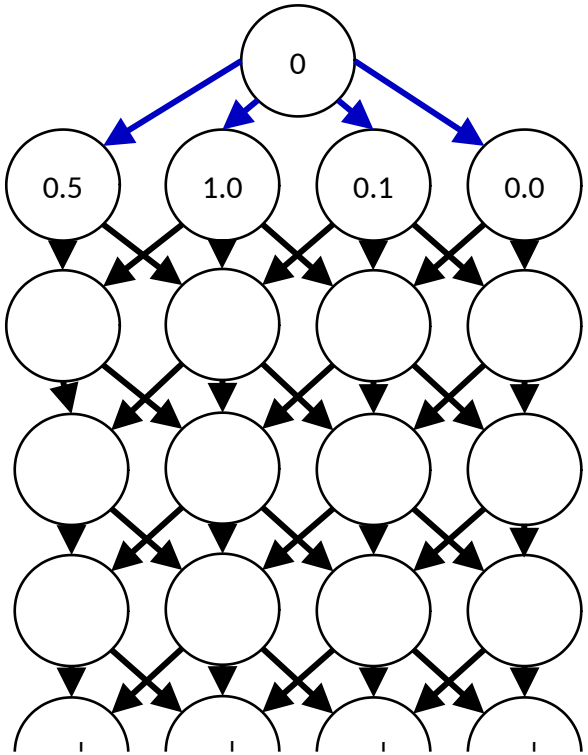
```
for all v ∈ vertices do
    for all n ∈ successors(v) do
        if distance(n) > distance(v) + cost(n) then
            distance(n) ← distance(v) + cost(n)
            bestPredecessor(n) ← v
```

Itération 2

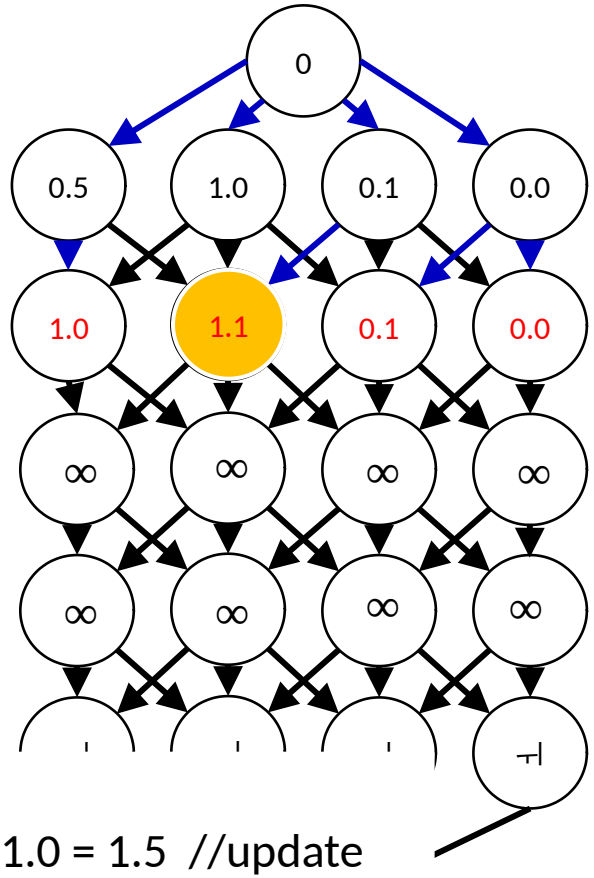
Input graph



distances in iteration 1



distances in iteration 2

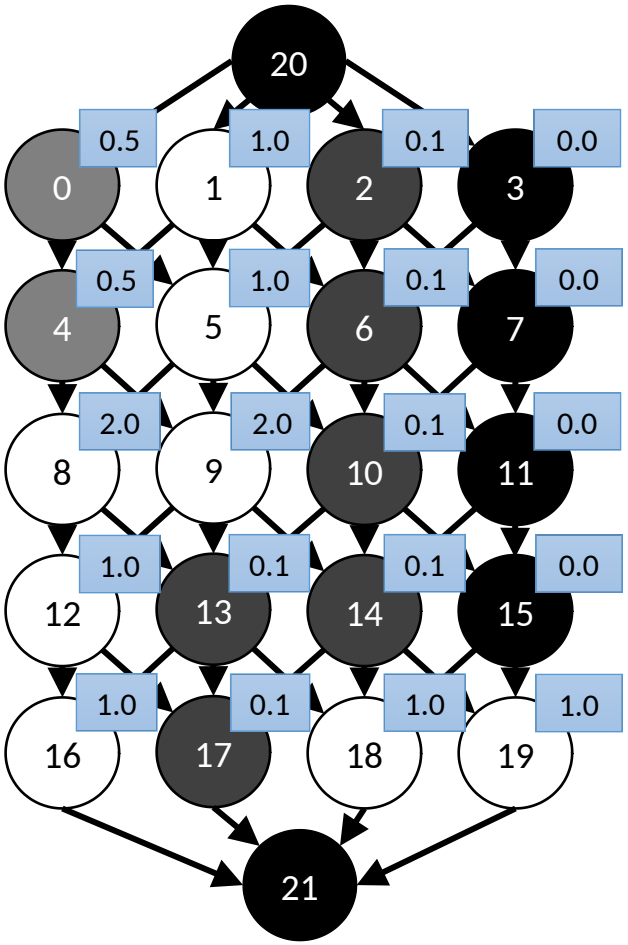


Node 5 is check 3x:

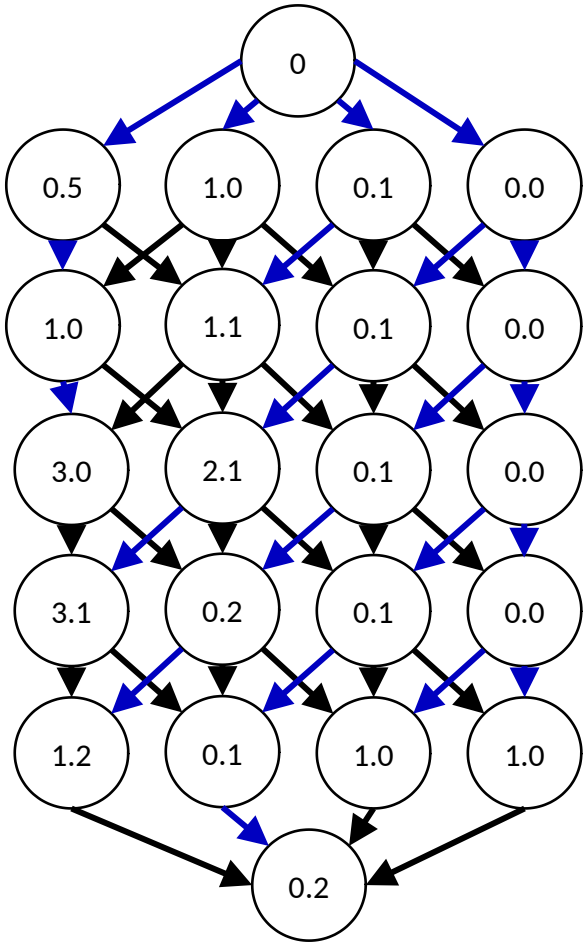
1. As successor of node 0: 0.5+1.0 = 1.5 //update
2. As successor of node 1: 1.0+1.0 = 2.0 //no update
3. As successor of node 2: 0.1+1.0 = 1.1 //update

Iteration finale

Input graph

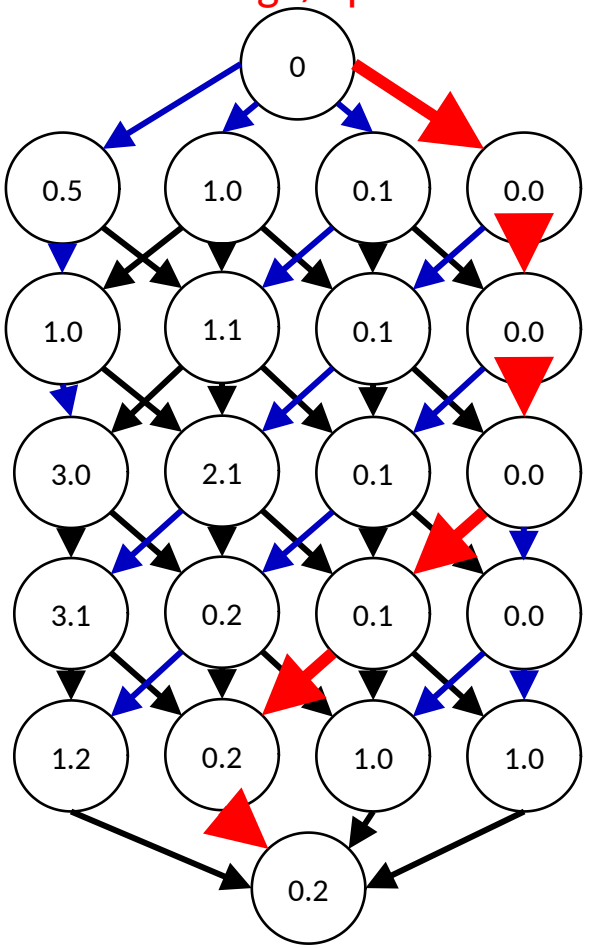


distances in iteration 6



distances in iteration 7

No change/update: DONE



Shortest path from 20 to 21: 20->3->7->11->14->17->21

Tâche 3: fonctions à coder

- `Graph create_graph(const GrayImage &gray)`
- `Path shortest_path(Graph &graph,`
  `size_t from,`
  `size_t to)`
- `Path find_seam(const GrayImage &gray)`

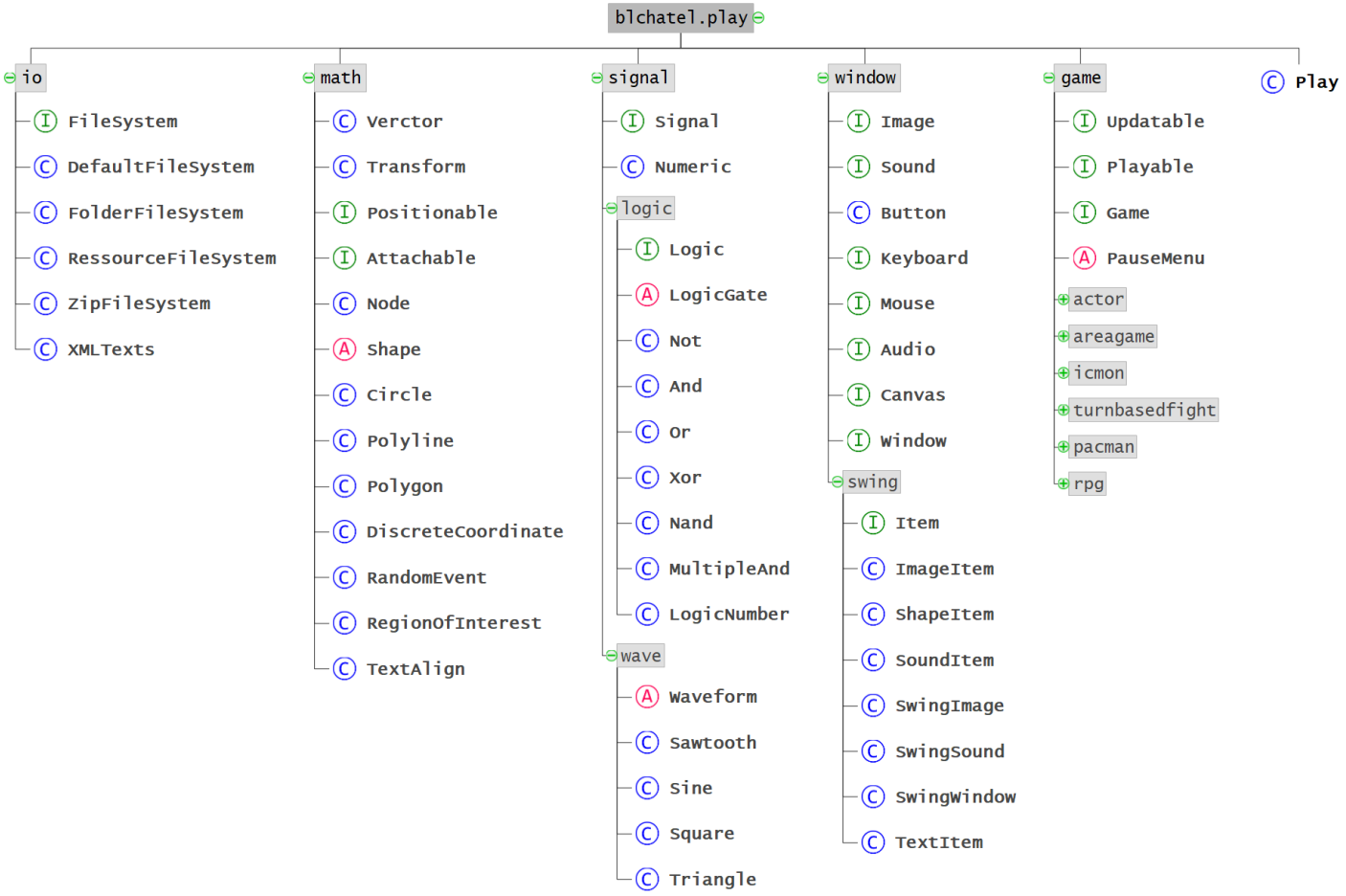
Puis possibilités d'extensions plus libres:

- Appliquer l'algorithme horizontalement
- Elargissement d'images
- ...

Autres exemples de projets

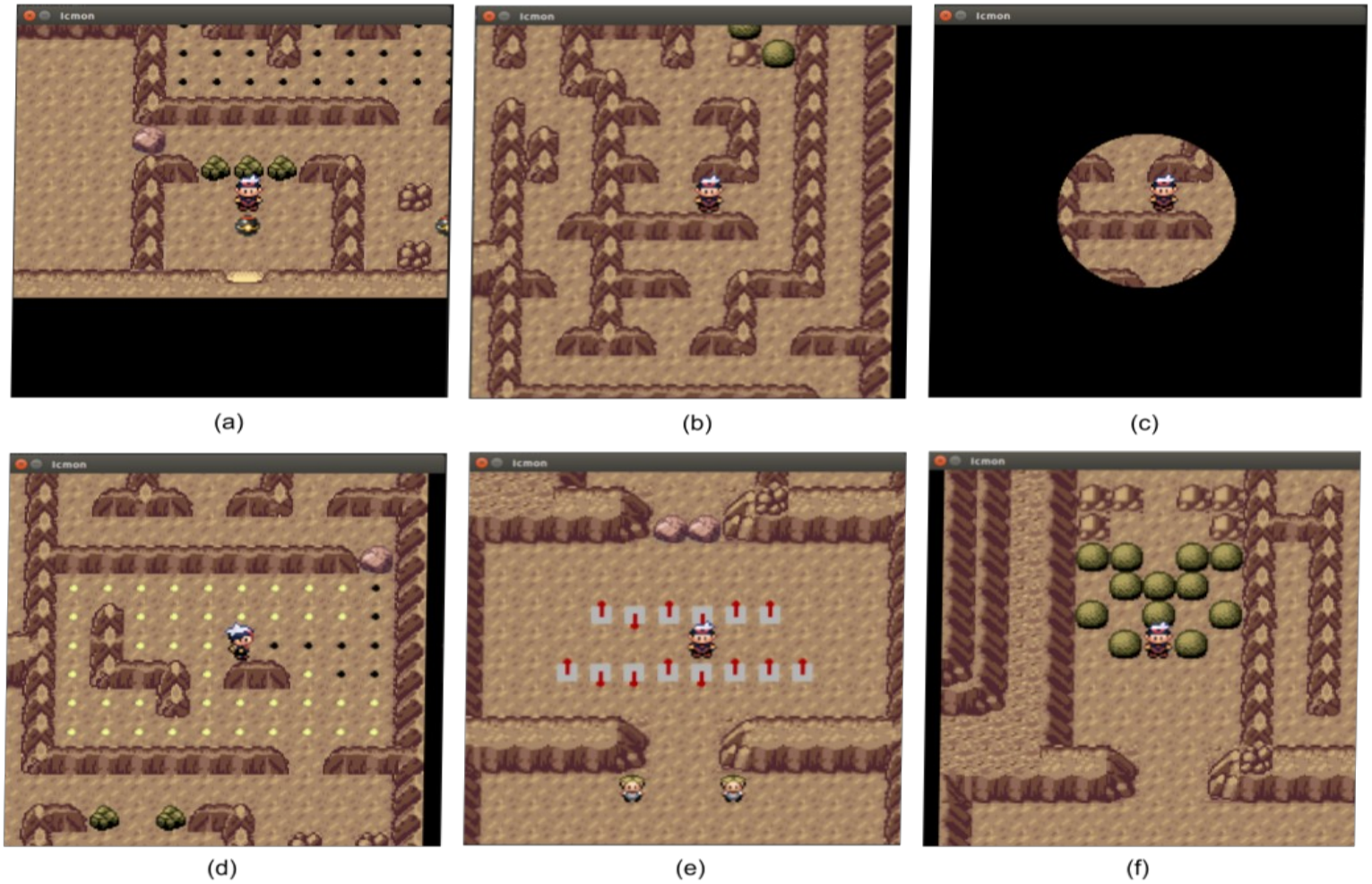
- Algorithme «Page Rank»
- Systèmes de recommandation
- Stéganographie
- «Où est Charlie ?»
- Reconnaissance d'écriture manuscrite
- Techniques de crypto-analyse (analyse de fréquences)
- Génération de QR-Codes
- Reconnaissances d'empreintes digitales

Boîte à outil (Jeux, Java)

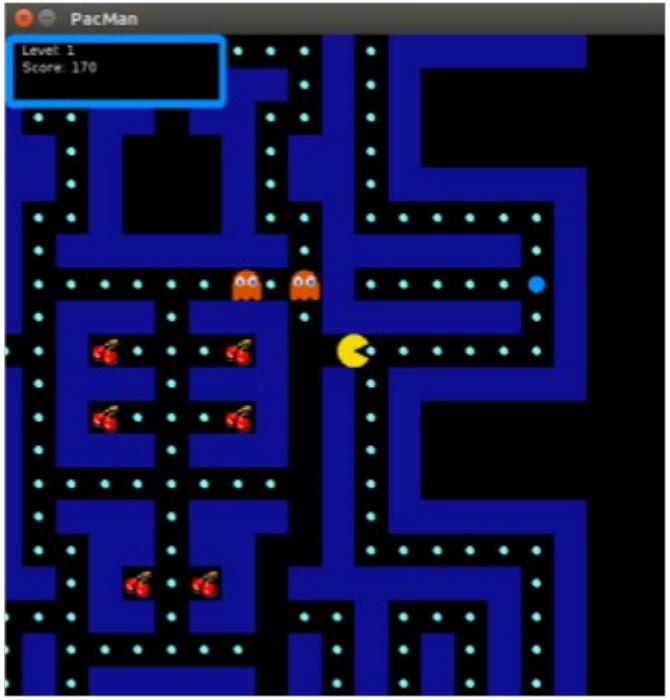


Enjeu : illustrer l'intérêt de la programmation orientée-objet

Exemple de projet dérivé

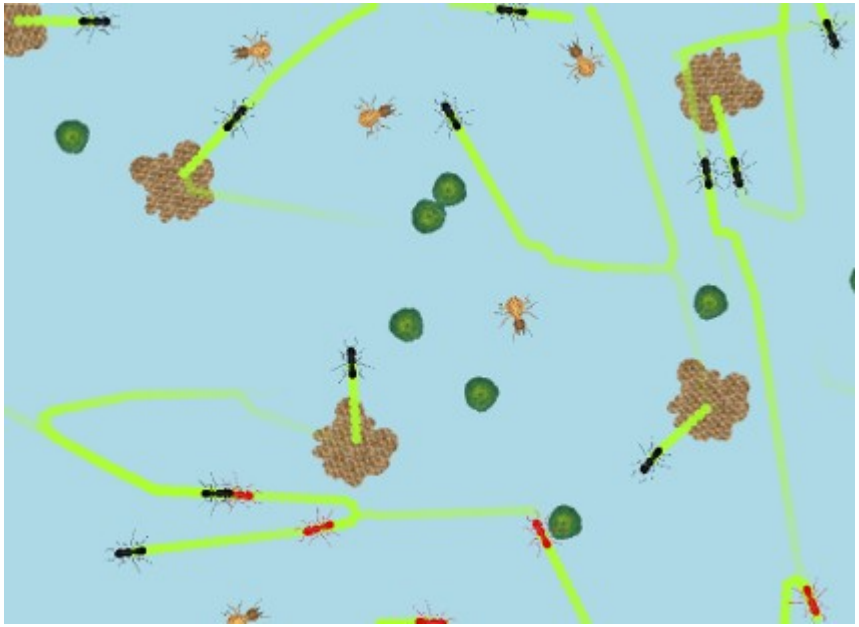


Exemple de projet dérivé



En version MOOC

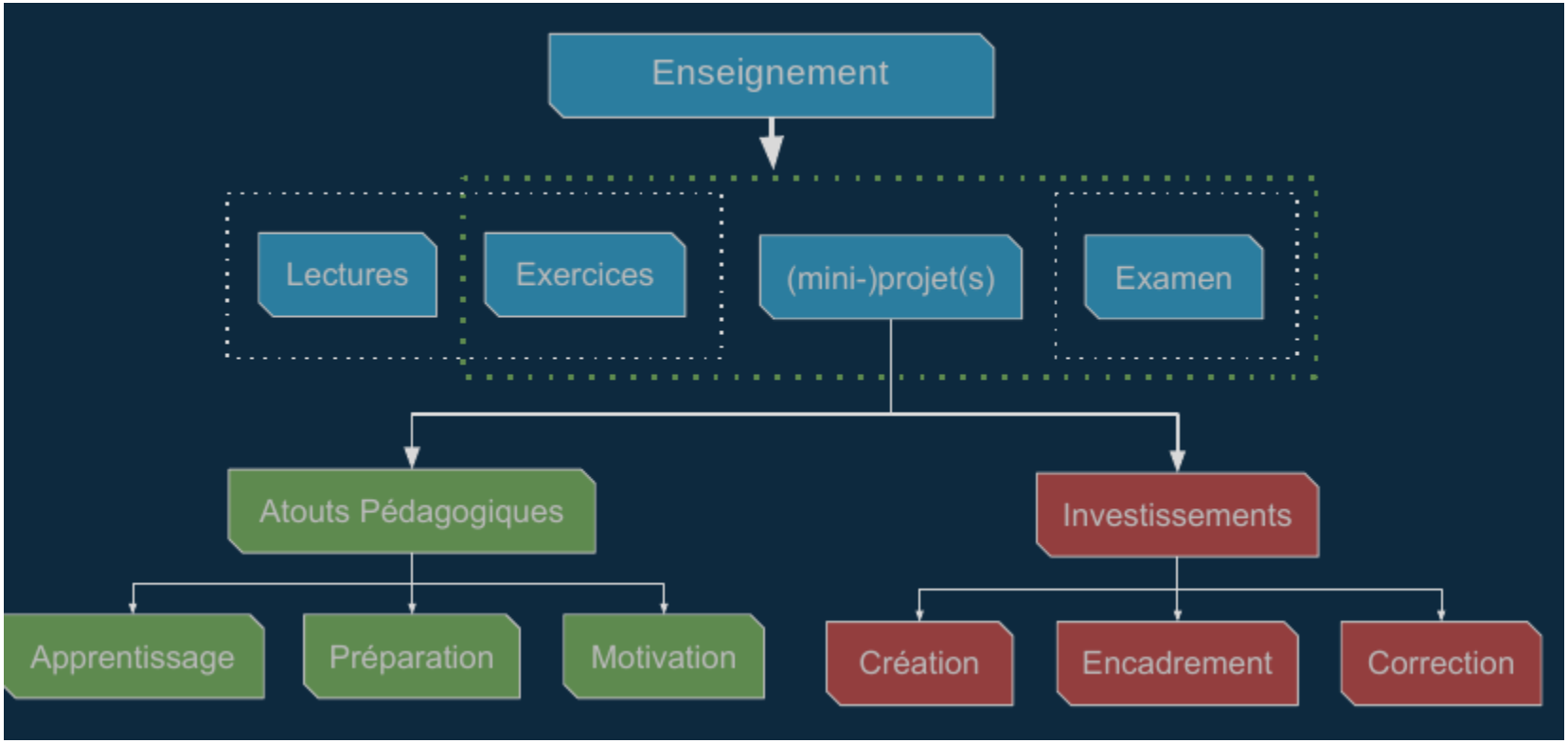
https://www.coursera.org/learn/projet-programmation-java/



Correction automatique

→ Format plus contraint

Retour d’expérience



Publication d’un catalogue?

- Intérêt plus large ? (notamment pour l’enseignement au niveau gymnasial)
- Sous quel format ? (MOOC, livre numérique ?)
- Contraintes ?(réutilisation, droits d’auteur etc.)

Questions?