UNIVERSITÉ DE NANTES
FACULTÉ DES
SCIENCES ET TECHNIQUES

ASD2 (X22I020)

2021–2022
L2 Informatique

projet **: Aerial Image Project**

réalisé par :

DIALLO Fatoumata
ASSOUD Ayham

Groupe: **484I**

Sommaire

Objectifs

Organisation

Diagramme de classe

description de la représentation mémoire

SDC Image
- Données
- Methodes
- complexités
- graphiques

SDC Analyst
- Données
- Methodes
- complexités

SDC FireSimulator
- Données
- Methodes
- complexités

Quelques images issues de la simulation

Conclusion

Objectifs

Notre travail consiste a créer une image, à partitionner cette image en zones de même couleur : zone de forêt en vert, zone de rivière en bleu, zone de cendres en noir et zone blanche de culture ou de sol.
Le feu se déclenche dans une zone de forêt, se propage tout en étant restreins dans cette zone, puis s'éteint en laissant place à du cendre selon le principe d'antériorité.

Organisation

Notre travail s'est organisé en trois phases.

-définir et implémenter :
. une SDC Image, la SDA étant déjà connue et définit.
. une SDC Analyst, la SDA étant déjà connue et définit.

-définition d'une SDA et une SDC FireSimulator.

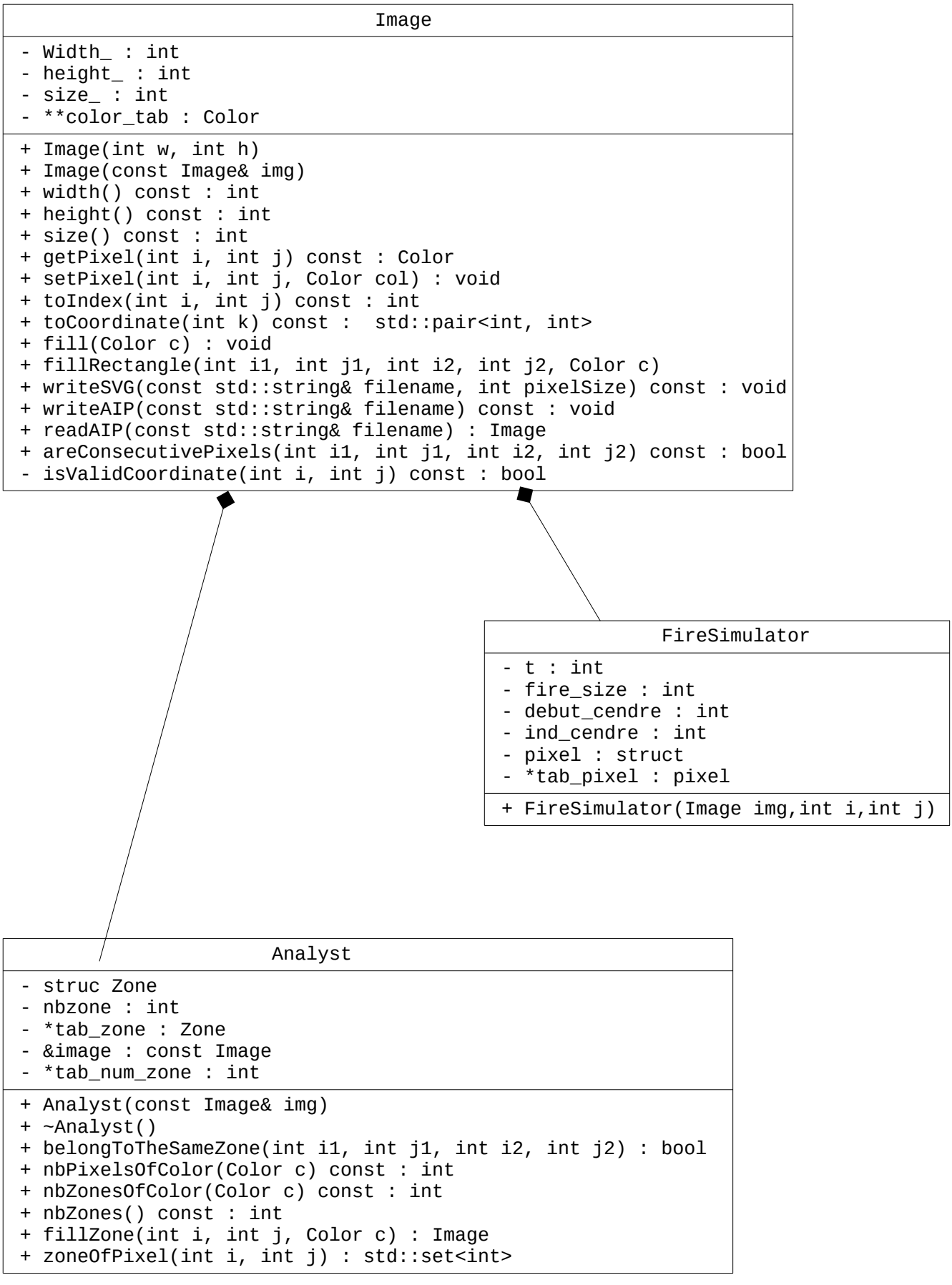


Image
- Width_ : int
- height_ : int
- size_ : int
- **color_tab : Color
+ Image(int w, int h)
+ Image(const Image& img)
+ width() const : int
+ height() const : int
+ size() const : int
+ getPixel(int i, int j) const : Color
+ setPixel(int i, int j, Color col) : void
+ toIndex(int i, int j) const : int
+ toCoordinate(int k) const : std::pair<int, int>
+ fill(Color c) : void
+ fillRectangle(int i1, int j1, int i2, int j2, Color c)
+ writeSVG(const std::string& filename, int pixelSize) const : void
+ writeAIP(const std::string& filename) const : void
+ readAIP(const std::string& filename) : Image
+ areConsecutivePixels(int i1, int j1, int i2, int j2) const : bool
- isValidCoordinate(int i, int j) const : bool
FireSimulator
- t : int
- fire_size : int
- debut_cendre : int
- ind_cendre : int
- pixel : struct
- *tab_pixel : pixel
+ FireSimulator(Image img,int i,int j)
Analyst
- struc Zone
- nbzone : int
- *tab_zone : Zone
- &image : const Image
- *tab_num_zone : int
+ Analyst(const Image& img)
+ ~Analyst()
+ belongToTheSameZone(int i1, int j1, int i2, int j2) : bool
+ nbPixelsOfColor(Color c) const : int
+ nbZonesOfColor(Color c) const : int
+ nbZones() const : int
+ fillZone(int i, int j, Color c) : Image
+ zoneOfPixel(int i, int j) : std::set<int>

description de la représentation mémoire

SDC Image

Une image rectangulaire correspond à une matrice de couleurs. Une Image a comme données : la largeur **width_** , la hauteur **height_** , **size_** nombre de pixels qui est égal à la multiplication de la longeur et largeur , ****color_tab** un tableau de couleurs à deux dimensions de taille size_ pour représenter la matrice. Une Image a comme paramètre w (la largeur) et h (la hauteur).
i est la variable en ordonnée et **j** la variable en abscisse.
L'image est initialisée avec une couleur noir.

Méthodes :

int width() const
Renvoie la l'argeur.

int height() const
Renvoie la la hauteur.

int size() const
renvoie le nombre de pixels

Color getPixel(int i, int j) const
renvoie la couleur du pixels dont les coordonnés sont i et j

void setPixel(int i, int j, Color col)
colore le pixel (i,j) avec la couleur col

int toIndex(int i, int j) const
renvoie l'index dont la valeur est: **index=(i-1)*width()+(j-1)**

std::pair<int, int> toCoordinate(int k) const
Associe un identifiant aux coordonnées correspondantes
Etant donné le résultat p, p.first est la ligne et p.second est la colonne . **p.first=k/width()+1 ; p.second=k%width()+1**

void fill(Color c)
colore avec la couleur c

void fillRectangle(int i1, int j1, int i2, int j2, Color c)
Remplit un rectangle avec la couleur donnée c. (i1, j1) est le coin supérieur gauche (i2, j2) est le coin inférieur droit . Nous avons appelée la méthode : **setPixel(i, j, c)** en parcourant les pixels du rectangles avec deux boucles .

void writeSVG(const std::string& filename, int pixelSize) const
Génère une image SVG avec un nom de fichier sans l'extension
Le nom de fichier de cette image est filename.svg .Chaque pixel est représenté par un carré de côté pixelSize Lève une exception **std::runtime_error** si une erreur se produit

void writeAIP(const std::string& filename) const

Enregistre ceci dans un fichier texte en utilisant un format spécifique. largeur (whidth)et hauteur (height) de ceci sur la première ligne suivi des lignes de this de sorte que chaque chiffre correspond à la couleur du pixel correspondant(dans une double boucle grace à l'instruction : **file << getPixel(i,j).toInt()**). Le fichier de sortie est nommé filename.aip lève une exception **std::runtime_error** si une erreur se produit.

Image readAIP(const std::string& filename)

Crée une image à partir d'un fichier AIP, Lève une exception std::runtime_error si une erreur se produit .On créer une image img avec logeur et largeur et on attribut les couleurs des pixels avec l'instruction : img.setPixel(i, j, Color::makeColor(color-'0')) dans une double boucle .

bool areConsecutivePixels(int i1, int j1, int i2, int j2) const

Renvoie vrai si (i1, j1) et (i2, j2) sont deux éléments consécutifs.Donc si la valeur absolue de de la différence entre i1 et i2 =1 et j1=j2 ; sinon si la valeur absolue de de la différence entre j1 et j2 =1 et i1=i2.

bool isValidCoordinate(int i, int j) const

test si i et j sont des coordonnés de pixel de l'objet courant

Image makeRandomImage(int w, int h)

Génère une image de largeur w et hauteur h telle que chaque pixel soit sélectionné au hasard

Complexité

Méthodes	**Complexités**
Image(int w, int h)	$\Theta(n^2)$
int width() const	$\Theta(1)$
int height() const	$\Theta(1)$
int size() const	$\Theta(1)$
Color getPixel(int i, int j) const	$\Theta(1)$
void setPixel(int i, int j, Color col)	$\Theta(1)$
Image(const Image& img)	$\Theta(1)$
int toIndex(int i, int j) const	$\Theta(1)$
std::pair<int, int> toCoordinate(int k) const	$\Theta(1)$
void fill(Color c)	$\Theta(n)$
void fillRectangle(int i1, int j1, int i2, int j2, Color c)	$\Theta(n^2)$
void writeSVG(const std::string& filename, int pixelSize) const	$\Theta(n^2)$
void writeAIP(const std::string& filename) const	$\Theta(n^2)$
Image readAIP(const std::string& filename)	$\Theta(n^2)$

bool areConsecutivePixels(int i1, int j1, int i2, int j2) const	Θ(1)
bool isValidCoordinate(int i, int j) const	Θ(1)

Image initiale

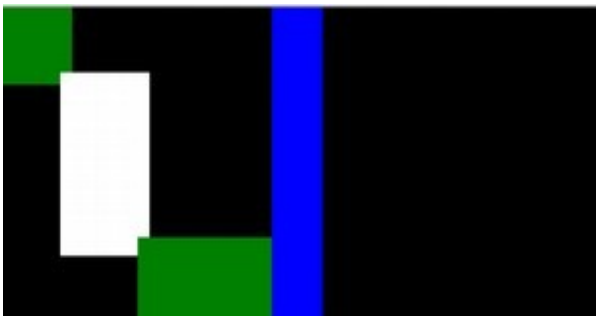


image avec les différentes zones

7 zones : 2 zones vertes de forêt,
une zone blanche de culture ou de sol,
une zone bleu de rivière ,
3 zones noirs de cendre .

SDC Analyst

L'analyse des images vise principalement à partitionner une image en zones de même couleur.
Analyst a pour donnés un type zone représenté par sa couleur et le numero de la zone (entier),un entier **nbzone** (le nombre de zone),un pointeur vers un tableau de zone **Zone* tab_zone ,** la copie de l'image , un entier **k** ,un pointeur vers un tableau d'entier **int* tab_num_zone.**

Méthodes

Analyst(const Image& img): image(img)

on initialise le nombre de zone **nbzone** à 0. on créer le tableau de zone , on compare la couleur de chaque pixel au pixel consécutif d'en haut et de gauche s'ils ont la même couleur alors c'est deux pixels ci appartiennent à la même zone
On fait attention au conditions : les pixel de coordonné i=1 n'ont pas de voisins haut, ceux de j=1 n'ont pas de voisin gauche. Ainsi on affecte à chaque pixel son numéro de zone.

bool belongToTheSameZone(int i1, int j1, int i2, int j2)

cette fonction retourne vrai si le pixel de coordonné (i1,j1) et le pixel de coordonné (i2,j2) appartiennent à la même zone donc ont le même numéro de zone.

int nbZones() const
renvoie le nombre de zone

int nbZonesOfColor(Color c) const
renvoie le nombre de zone de couleur c . On créer une liste vide d'entier **tab_num_zone** et dans cette liste,on compare pour chaque pixel si la couleur correspond à celle en paramètre et si son numéro ne se trouve pas dans le tableau on l'insère,on fait ceci jusqu'à avoir tous les numéro de zone de couleur c ; Et au final on

retourne la taille du tableau .

std::set<int> zoneOfPixel(int i, int j)
renvoie tous les indexes de la zone où on a le pixel (i,j).En effet, on stock le numéro de zone du pixel (i,j),on stock dans une liste **pixel_indexes** tous les pixels qui ont le même numéro que notre pixel en paramètre, et on retourne cette liste.

Image fillZone(int i, int j, Color c)
Crée une nouvelle image en remplissant la zone de pixel (i, j) dans l'entrée
image avec une couleur donnée.

Complexité

Méthodes	**Complexités**
Analyst(const Image& img): image(img)	$\Theta(n^2)$
bool belongToTheSameZone(int i1, int j1, int i2, int j2)	$\Theta(1)$
int nbZones() const	$\Theta(1)$
int nbZonesOfColor(Color c) const	$\Theta(n^2)$
std::set<int> zoneOfPixel(int i, int j)	$\Theta(n^2)$
Image fillZone(int i, int j, Color c)	$\Theta(1)$

SDC FireSimulator

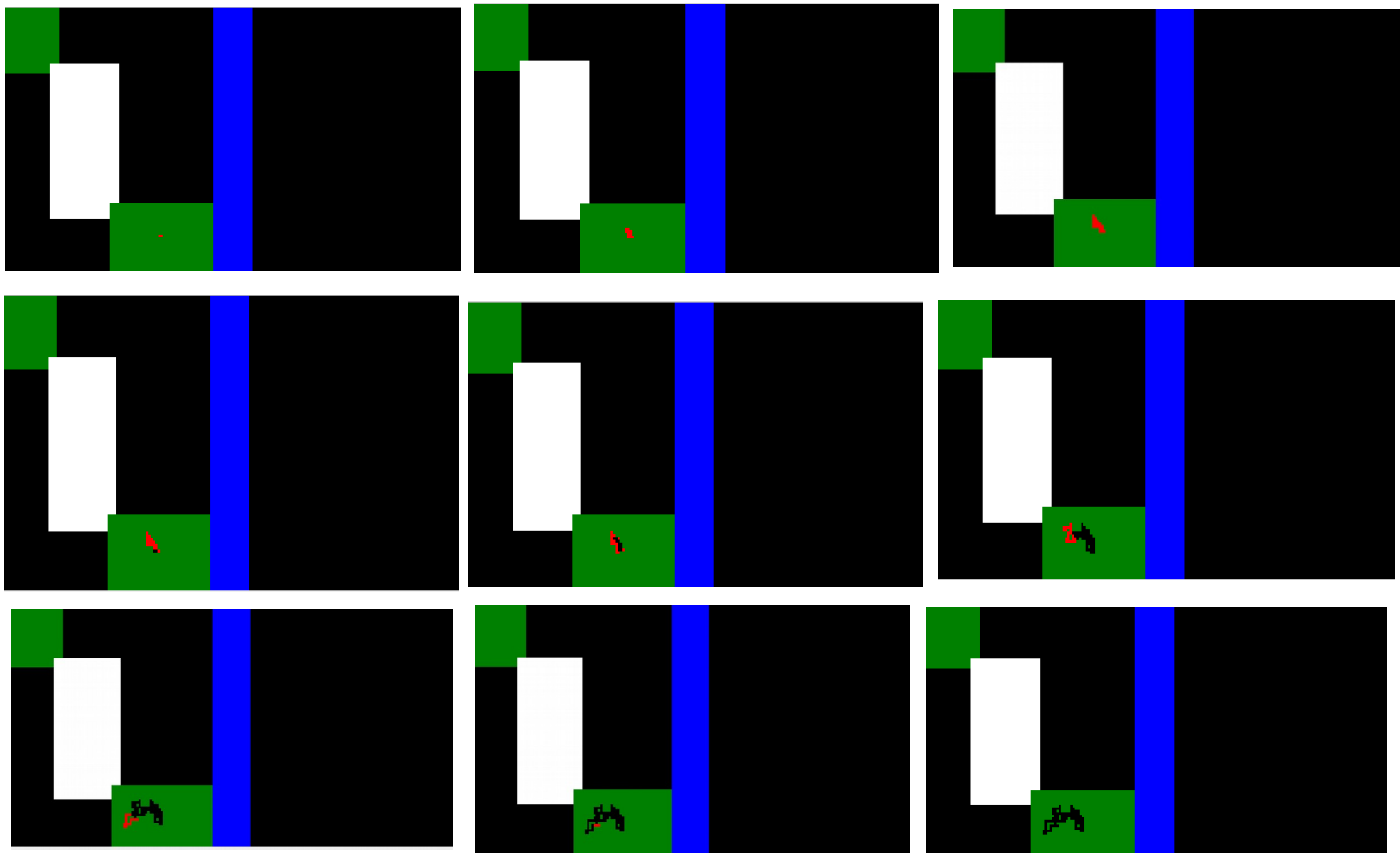
La simulation d'un feu prend en entrée une image(**img**) et les coordonnées d'un point ou pixel dans une zone de forêt(**i,j**) .

l'image en entrée correspond à t = 0,
FireSimulator a pour donnés:
fire_size est un nombre aléatoire qui représente le nombre de pixels de feu qui vont être rajoutés
debut_cendre est un nombre aléatoire qui désigne le début de propagation du feu
tab_pixel est un tableau de pixel ,ce tableau stock au fur et à mesure les pixels du feu et il va permettre la propagation du cendre .
t : le temps
Le feu se déclenche à partir d'un pixel ensuite il peut se propagé de 4 façon : soit le pixel du haut,du bat ,gauche ou encore droit .
Le feu se propage et arrivé un moment le cendre commence à se propagé suivant le principe d'antériorité.

Complexité

Méthodes	**Complexités**
FireSimulator(Image img,int i, int j) ~FireSimulator()	$\Theta(n)$ $\Theta(1)$

Quelques images issues de la simulation



On peut observé dans cette simulation que le feu se déclenche dans une zone de forêt à la première image
De t = 0 à t = 1, se propage. Le cendre commence à se propagé à l'image du milieu gauche.
Les deux continuent à se propager jusqu'à ce que le feu s'éteigne laissant place au cendre .

Conclusion

Ce projet s'est révélé très enrichissant dans la mesure où il nous a apprit a partitionner une image en zone de couleur , à pouvoir visionner ceci ainsi que la simulation du feu.