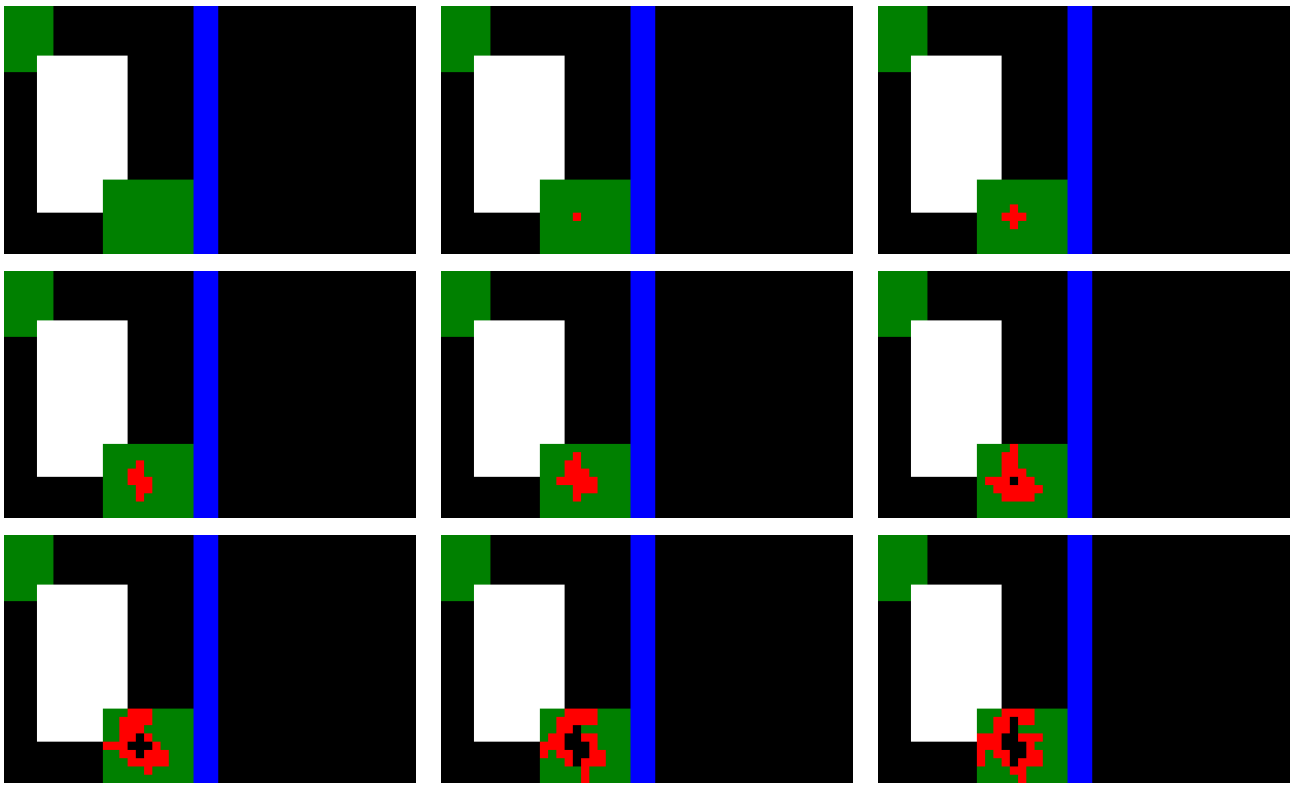
Aerial Image Project

1 Contexte

Les photos aériennes prises par des satellites ou des avions permettent d'étudier des phénomènes variés. En prenant des photos de la même zone à des moments différents, il est possible d'analyser l'évolution de ces phénomènes dans le temps. Par exemple, les images ci-dessous montrent l'évolution d'une forêt et la propagation d'un feu, les images étant rangées de gauche à droite et de haut en bas. En haut à gauche, on a l'image initiale au temps $t = 0$ comportant deux zones de forêt en vert, une rivière en bleu, trois zones de cendres en noir et une zone blanche de culture ou de sol. Dans l'image en haut au centre à $t = 1$, un feu se déclare dans une zone de forêt. Dans l'image en haut à droite à $t = 2$, le feu se propage. On voit sur les images suivantes qu'il continue à se propager dans la zone de forêt et seulement dans cette zone. Sur l'image du centre à droite à $t = 5$, le point du départ de feu s'éteint laissant place à des cendres. Sur l'image en bas à gauche à $t = 6$, on voit que le feu continue à se propager mais aussi que la zone de cendres s'étend. Les cendres remplacent le feu selon un principe d'antériorité, d'abord dans les endroits où le feu a duré le plus longtemps, c'est-à-dire là où il s'est déclaré le plus en amont dans le passé.



Ces images sont issues d'une simulation informatique. En particulier, l'image initiale a été créée artificiellement et le feu se propage de manière aléatoire. L'objectif du projet est de produire des simulations similaires. Cela pose des questions algorithmiques intéressantes. Comment partitionner une image en zones de même couleur ? Comment mesurer la taille des forêts ? Comment restreindre un feu à se propager dans une seule zone ? Comment éteindre un feu selon le principe d'antériorité décrit précédemment ? Comment reboiser partiellement ou totalement une zone brûlée ?

2 Travail

Vous devrez commencer par récupérer le fichier AIPetu.zip sur madoc. Le travail est organisé en trois phases successives décrites ci-après.

2.1 Les images

Une image rectangulaire correspond à une matrice de couleurs. La SDA Image est connue et définie par les signatures des méthodes publiques de la classe correspondante. Le travail consiste à définir et implémenter une SDC Image. Une matrice peut être codée par un tableau à deux dimensions ou un tableau à une seule dimension dans lequel les lignes sont mises bout à bout. Le choix du codage est libre mais également conditionné par la complexité temporelle des algorithmes des méthodes que vous devrez définir.

Dans le fichier Image.h qui est fourni, les signatures des méthodes sont mises en commentaires. Il faudra bien sûr les décommenter au moment de les implémenter dans le fichier Image.cpp. Dans le fichier Image.cpp, la génération d'une image au format SVG est fournie. Le code s'appuie sur les méthodes publiques de la classe. Il pourra donc être directement réutilisé. Enfin, vous trouverez des images au format AIP (format non standard défini pour ce projet) dans le répertoire images. Ces images sont celles de la première page de ce sujet. Vous pourrez les lire au moyen de la méthode readAIP et les transformer en images SVG au moyen de la méthode writeSVG de manière à vérifier que votre code est correct.

Vous devrez suivre une méthode de développement raisonnable : réfléchir avant de coder ; coder un peu et tester beaucoup ; mettre des assertions pour vérifier les préconditions ; commenter les parties non triviales du code, les attributs, etc.

2.2 Analyse des images

L'analyse des images vise principalement à partitionner une image en zones de même couleur. Par exemple, la première image de la simulation est composée de 7 zones ou parties. La SDA Analyst est connue et définie par les signatures des méthodes publiques de la classe correspondante. Le travail consiste à définir et implémenter une SDC Analyst : représentation mémoire (attributs), codage des méthodes et complexité temporelle des algorithmes. Un objectif est d'avoir les meilleures complexités possibles.

Les signatures des méthodes publiques ne devront pas être modifiées. Cela permettra d'évaluer votre projet sur des jeux de test choisis par les professeurs. Vous devrez vérifier que la commande suivante ne génère pas d'erreur de compilation :

```
g++ Color.cpp Image.cpp Analyst.cpp testeval.cpp -o testeval
```

Vous pourrez utiliser le programme exécutable pour tester en donnant les noms des fichiers sur la ligne de commande, par exemple :

```
./testeval img1.aip img2.aip img3.aip
```

Ce programme permet de mesurer le temps d'exécution. Un bonus à la performance sera accordé aux codes (corrects) les plus rapides.

2.3 Simulation du feu

La simulation d'un feu prend en entrée une image et les coordonnées d'un point ou pixel dans une zone de forêt. Le travail demandé est de définir une SDA et une SDC FireSimulator. On peut établir un cahier des charges (à compléter) :

1. Le temps s'écoule $t = 0, 1, 2 \ldots$ L'image donnée en entrée correspond au temps $t = 0$.
2. Une méthode permet de faire évoluer la situation du temps courant t au temps suivant $t + 1$. De $t = 0$ à $t = 1$, elle crée simplement un départ de feu en choisissant un pixel aléatoirement dans la zone de forêt. Par la suite, elle propage le feu dans la zone puis elle éteint ce feu selon le principe d'antériorité décrit précédemment.
3. Une méthode permet de faire évoluer la situation sur n périodes de temps.
4. On peut générer l'image (un objet Image) qui correspond à la situation à chaque temps t.

La propagation du feu se fait dans la zone de forêt autour des pixels de couleur rouge. Le nombre maximum de nouveau pixels rouges peut être tiré aléatoirement entre 1 et une proportion de la taille de la zone de forêt restante. Le même principe pourra s'appliquer à l'extinction du feu.

3 Consignes

Les consignes sont des impératifs à respecter.

3.1 Méthode de travail

Le travail se fait par binômes. Vous devrez inscrire votre binôme au plus le **28 février à 12h** dans le fichier dédié sur madoc. Si votre binôme n'est pas inscrit dans ce fichier à cette date, votre note finale sera 0. En cas de problème pour trouver un binôme, il faudra vous identifier auprès de votre chargé de TP le plus tôt possible, et ce avant le 28 février.

Il est conseillé de créer un dépôt git pour travailler de manière collaborative.

Comme dit ci-avant, vous devrez suivre une méthode de développement raisonnable : réfléchir avant de coder ; définir des SDA claires ; choisir des SDC pour avoir les meilleures complexités possibles des algorithmes ; coder un peu et tester beaucoup ; mettre des assertions pour vérifier les préconditions ; commenter les parties non triviales du code, les attributs, etc.

3.2 Rendu du projet

Le rendu du projet se fera au plus tard le **vendredi 29 avril à 18h** par l'un des membres du binôme dans le dépôt de son groupe de TP sur madoc. Aucun retard ne sera autorisé.

Vous devrez déposer une archive au format zip comportant un rapport au format pdf et les programmes sources. Pensez à faire un make clean avant de créer l'archive.

Le rapport sera constitué des 4 parties suivantes (et seulement de ces parties) :

1. La description de la représentation mémoire de la SDC Image (explications, graphiques) et un tableau recensant les complexités temporelles (en ordre de grandeur) des méthodes publiques ;
2. La description de la représentation mémoire de la SDC Analyst et un tableau recensant les complexités temporelles (en ordre de grandeur) des méthodes publiques ;
3. La SDA FireSimulator écrite en pseudo-code, la description de la représentation mémoire de la SDC FireSimulator et un tableau recensant les complexités temporelles (en ordre de grandeur) des opérations de la SDA.
4. Quelques résultats de simulation montrant des images de la propagation d'un feu comme cela apparaît dans la première page de ce sujet.

3.3 Absences aux TP

Le projet sera encadré pendant quatre séances de TP. Un appel sera fait à chaque séance. Il sera toléré au plus une absence non justifiée par personne. Si vous ne justifiez pas au moins deux absences, votre note finale sera 0.

3.4 Exigences et notation

Les exigences pour une SDC sont les suivantes :

- Au niveau A, la description de la représentation en mémoire est très claire et bien expliquée au moyen de graphiques. Les complexités des algorithmes sont optimales.
- Au niveau B, la description est celle du niveau A et les complexités sont bonnes sans être optimales.
- Au niveau C, la description n'est pas claire mais les complexités sont bonnes.
- Au niveau D, la description est absente ou incohérente ou les complexités sont mauvaises par rapport à ce qui est attendu.

Les exigences pour une SDA sont les suivantes :

- Au niveau A, le rôle de chaque opération ou méthode est bien expliqué. Les signatures sont définies et constituent une bonne interface pour un utilisateur. Les préconditions sont données. On peut utiliser la SDA facilement pour écrire de nouveaux algorithmes.
- Au niveau B, la SDA est globalement bonne mais les explications ne sont pas toujours limpides et l'interface peut être améliorée. La SDA peut quand même être utilisée pour écrire de nouveaux algorithmes.
- Au niveau C, il manque des préconditions, les rôles ne sont pas toujours expliqués clairement et l'interface peut être améliorée.
- Au niveau D, la SDA est inutilisable pour écrire de nouveaux algorithmes. Un utilisateur ne peut pas la comprendre.

Les exigences pour le code C++ sont les suivantes :

- Au niveau A, le code est clair, bien commenté, bien indenté. Les classes standards de C++ sont mises en œuvre à bon escient. La compilation ne génère pas de *warnings*. Les méthodes sont élémentaires, le code est bien découpé. Les résultats d'exécution sont corrects, i.e. les algorithmes sont corrects et bien codés.
- Au niveau B, le code est globalement d'un bon niveau mais il y a des faiblesses par rapport au niveau A. Par exemple, certaines méthodes ne sont pas élémentaires. Les résultats d'exécution sont corrects, i.e. les algorithmes sont corrects et bien codés.
- Au niveau C, le code compile mais il est mal commenté ou indenté (résultat d'un développement à la va-vite). Il n'y a pas de plantage à l'exécution mais certains résultats sont incorrects, i.e. certains algorithmes sont faux ou mal codés.
- Au niveau D, le code ne compile pas ou il plante à l'exécution. Le code n'est pas commenté. Il manque des parties du code.

Enfin, un barème est donné ci-dessous à titre indicatif :

- Entre 17 et 20 : niveau au moins B+ en moyenne, niveau A à B dans toutes les catégories ;
- Entre 13.5 et 16.5 : niveau B en moyenne, niveau au moins C+ dans toutes les catégories ;
- Entre 10 et 13 : niveau C+ en moyenne, solution en partie fonctionnelle ;
- Moins de 10 : niveau C ou D en moyenne, projet insuffisant comportant de nombreux manques.