Développement
en environnement de base de données

Projet 1

Table des matières

Énoncé

Vous devez concevoir et réaliser un logiciel simulant le jeu de la vie de Conway.

Objectif général

Ce premier laboratoire vise la réalisation d'un projet informatique visant la mise en pratique des éléments suivants :

- programmation orientée objets,
- développement modulaire,
- programmation en C++.

Objectifs spécifiques

Plus spécifiquement, ce projet vise ces considérations :

- conception orientée objets mettant de l'avant la notion d'encapsulation;
- développement favorisant la modularité et la réutilisabilité;
- utilisation de 3 patrons de conceptions :
  - singleton,
  - itérateur,
  - façade.
- programmation moderne en C++.

Il est important de comprendre que ce projet ne vise pas à évaluer votre capacité à produire simplement un logiciel simulant le jeu de la vie. Ce projet vise davantage votre capacité à produire un logiciel de qualité mettant en pratique les notions fondamentales de la programmation orienté objet considérant les contraintes du langage C++.

Présentation sommaire du jeu de la vie

Le jeu de la vie (*Game of life*) est un automate cellulaire imaginé par John Horton Conway. Les automates cellulaires sont des jeux à 0 joueur où le joueur peut déterminer les conditions initiales et observer l'évolution de la simulation.

L'univers

- 2d,
- discret,
- malgré qu'il soit théoriquement infini, il reste de dimension finie pour une simulation,
- il peut être borné ou circulaire.

Cellule de l'univers

- chaque case est appelée une cellule,
- possède l'un de ces deux états :
  - inactif mort 0 noir
  - actif vivant 1 blanc

Pour chaque évolution :

- chaque cellule est analysée selon :
    - son état,
    - le nombre de cellules actives parmi les 8 cellules voisines immédiates;
- la règle de Conway dit :
    - une cellule inactive :
        - devient active si elle possède 3 voisins actifs,
        - reste inactive sinon;
    - une cellule active :
        - reste active si elle possède 2 ou 3 voisins actifs,
        - devient inactive sinon;
- le traitement lié à la simulation ne doit pas modifier l'état courant de l'univers pendant le calcul de l'état suivant.

Initialisation de l'univers :

- aléatoirement (rarement intéressant)
- par la disposition de patrons connus
- par la disposition aléatoire de patrons connus

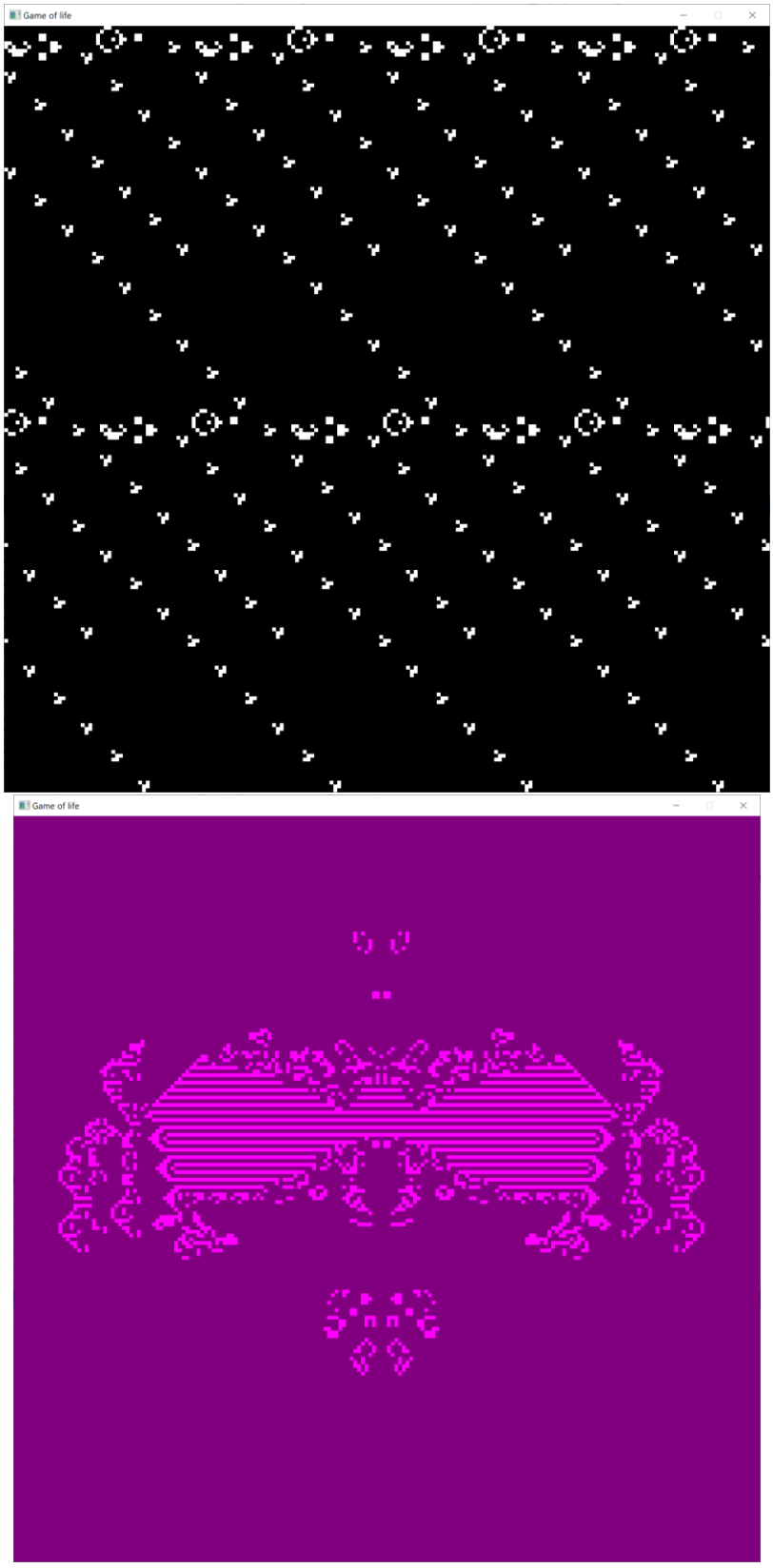
Présentation générale du projet

Le logiciel réalisé doit offrir :

- la lecture de fichiers RLE;
- un outil de visualisation de la simulation;
- une interface utilisateur permettant les quelques manipulations obligatoires des options :
    - alternance de la gestion des bordures : mortes ou cycliques;
    - alternance de la règle : B3/S23, B36/S23, B3678/S34678 et une autre de votre crue;
    - génération de conditions initiales selon :
        - aléatoire,
        - fichiers RLE;
    - changement de la palette de couleur;
    - pause / reprise;
    - vitesse de la simulation;
- un ajout personnel.

Interface utilisateur

L'interface utilisateur est très simple. On ne voit que le résultat de la simulation dans une console.



Exemples de l’interface usager
Deux simulations en cours

Les commandes à réaliser sont :

- Barre d'espacement : pause et reprise de la simulation
- 1 à 9 : vitesse de la simulation (x1, x2, x3, …, x9)
- R ou r : bascule entre les différentes règles :
    - B3/S23
    - B36/S23
    - B3678/S34678
    - une règle de votre crue
- B ou b : bascule entre les deux modes liés à la gestion des effets de bord :
    - bordures mortes
    - bordures cycliques
- P ou p : bascule entre les différentes couleurs pour les cellules actives :
    - blanc intense
    - rouge intense
    - vert intense
    - bleu intense
    - jaune intense
    - magenta intense
    - cyan intense
- O ou o : bascule le mode de couleur pour les cellules inactives :
    - noir,
    - même couleur que les cellules actives mais en version foncée;
- A, S, D, F, G, H (insensible à la casse) : génération aléatoire selon les pourcentages suivants :
    - A ou a : 1%
    - S ou s : 5%
    - D ou d : 10%
    - F ou f : 15%
    - G ou g : 25%
    - H ou h : 50%
- Z, X, C (insensible à la casse) : réinitialisation de l'univers selon la liste des fichiers donnés :
    - Z ou z : fichier précédent dans la liste
    - X ou x : même fichier dans la liste
    - C ou c : fichier suivant dans la liste

Fichiers RLE

Vous avez à votre disposition un ensemble de plusieurs fichiers sélectionnés venant du site ConwayLife. Ces fichiers décrivent des patrons intéressants à simuler. Vous devez être en mesure de lire ces fichiers et d'initialiser l'univers en disposant le patron lu au centre de l'univers.

Le fichier RLE est un fichier texte structuré en trois parties distinctes. La première partie est optionnelle et possède *n* lignes (où $n \geq 0$). Elle sert à décrire le patron. La deuxième partie est obligatoire et ne constitue qu'une seule ligne donnant des informations techniques sur le patron (notamment sa taille). La troisième partie est obligatoire et représente le patron lui-même réparti sur *p* ligne (où $p > 0$).

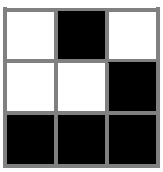
Voici une description détaillée du format RLE à lire :

1. Section en-tête (où seules ces expressions sont possibles) : *n* lignes (où $n \geq 0$)
    - #C [commentaires]
    - #c [commentaires]
    - #N [nom du patron]
    - #O [auteur]
    - #R [x] [y] x et y représentent les coordonnées de décalage recommandée
    - #r [nom de la règle sous la forme B3/S23 ou 3/23]
2. Ligne de transition : 1 ligne
    - Une ligne offrant cette syntaxe : x = [width], y = [height], rule = [abc]
    - La troisième partie (*rule*) est optionnelle alors que les 2 premières sont obligatoires.
3. Données du patron sous format RLE (*Run Length Encoded*) : *p* lignes (où $p > 0$)
    - [*nombre*]étiquette
        - où le *nombre* est optionnel
            - s'il est présent, il indique le nombre d'occurrence de l'étiquette
            - s'il est absent, l'étiquette n'est présent qu'une seule fois
        - l'étiquette ne peut être que l'un de ces trois caractères :
            - b = cellule inactive
            - o = cellule active
            - $ = saut de ligne
    - ! = fin du fichier

Ainsi, le fichier suivant :

```
#C Un planneur.
x = 3, y = 3
bo$2bo$3o!
```

Donne le patron suivant :



Il est attendu que vous fassiez une gestion des erreurs de conformité du fichier d'entrée. Minimalement, vous devez gérer ces cas :

- Même si pour cette application on ignore toutes les lignes d'en-tête, elles doivent débuter par # et respecter les 6 caractères valides pour la 2e position (CcNORr).
- L'en-tête peut posséder entre *0* et *N* lignes.
- La ligne de transition doit être valide et obligatoirement présente. Les tailles *x* et *y* doivent être positive.
- La séquence doit être valide et ne jamais déborder la taille du patron indiqué par *x* et *y*.
- Le caractère de fin de séquence ! est attendu.
- Le contenu du fichier se trouvant après le caractère ! est ignoré.

Vous devez ajouter une condition dans la procédure de lecture : si un patron est plus grand que la taille de l'écran, la lecture du fichier termine immédiatement sans charger le patron. Cette stratégie est essentielle car certains fichiers sont gigantesques et pourraient nécessiter plusieurs Go en RAM.

Conception orientée objet

Réalisé ce projet est relativement facile pour des finissants. L'objectif n'est pas de réaliser simplement le projet mais de faire une bonne analyse afin de produire un logiciel de qualité.

L'un des critères les plus importants pour ce projet n'est pas la performance de la simulation (quoiqu'elle reste significative) ou l'ajout de fonctionnalités superflues mais plutôt la modularité et la qualité du logiciel, même pour votre ajout personnel.

Schéma de classes

Pour ce faire, il est attendu que vous produisiez un diagramme de classe UML que vous mettrez à même votre projet Visual Studio dans un dossier nommé doc. Ce diagramme de classes doit illustrer toutes les classes que vous faites et les relations qu'elles ont les unes avec les autres. Il n'est pas nécessaire d'indiquer les détails des attributs et méthodes mais de mettre l'emphase sur un bon découpage et les relations (incluant la cardinalité).

Patrons de conception

Afin de vous assurer de bien pratiquer certaines notions liées au paradigme orienté objet, on vous demande de mettre en pratique ces 3 patrons de conceptions.

Singleton

Vous n'avez pas à réaliser un singleton mais plutôt à utiliser adéquatement celui de la librairie Console.

Itérateur

Vous n'avez pas à créer un nouvel itérateur mais plutôt à les utiliser.

Façade

On vous impose de créer une façade pour gérer le voisinage d'un état. Ainsi, cette classe doit prendre en charge toutes les technicalités liées à la gestion du voisinage d'une cellule. C'est à vous de déterminer les détails de cette classe pour qu'elle soit pertinente.

Modèle/vue/contrôleur (optionnel)

Optionnellement, vous pouvez utiliser le patron MVC que vous connaissez bien.

Contraintes de réalisation

Programmation C++

Il est attendu que vous mettiez de l'avant les concepts vus en classe :

- garde fou (#ifndef … #endif)
- un duo de fichier par classe (sans exception)
- utilisation de la norme de codage donnée dans les exercices
- vous devez déclarer les constructeurs par défaut et le destructeur; en plus, vous devez déterminer si vous faites votre implémentation, l'implémentation par défaut (= default) ou si vous voulez retirer ces fonctions (= delete).
- vous devez utiliser les variables, références et pointeurs adéquatement
- vous devez déterminer les versions const (variables, références, pointeurs et fonctions)
- vous utilisez les variables et fonctions statiques judicieusement
- vous devez pousser les concepts de l'encapsulation de l'orienté objet (sans utiliser l'héritage et le polymorphisme)
- utilisation des classes et fonctions de la libraire standard

Un indice : il est possible de réaliser ce projet sans aucune allocation dynamique explicite de votre part.

Commentaires

Chaque fichiers *.h doit posséder cet en-tête adéquatement rempli :

```
// Contexte de réalisation (exemple : institution | no de cours)
//
// Courte description (une ligne)
//
// Description détaillée (optionnelle)
//
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
//
// Date de création :
// Auteurs :
//    - principal
//    - autres (si requis)
//

#ifndef ...
```

Vous n'avez pas à documenter tout votre code. C'est-à-dire qu'on se fie beaucoup sur l'auto-documentation que vous faites de vos types, fonctions, variables, etc. L'ajout de commentaires supplémentaires n'est requis seulement lorsqu'il y a des subtilités que vous jugez pertinente de partager.

Si vous désirez ajouter des commentaires, sachez qu'on retrouve généralement seulement les concepts et les exemples dans les *.h. Les commentaires dans les *.cpp représentent davantage des détails techniques.

Surtout, ne pas mettre des commentaires creux :

```
if (mSpeed == 0.0) {     // si la vitesse est 0   | Commentaire absolument
      ...                //                       | inutile! À proscrire. Il est
}                        //                       | mieux de ne rien mettre.
```

Autres contraintes

Vous devez respecter ceci :

- Vous devez travailler en équipe de 3 à 4 (4 équipes de 4 étudiants et 2 équipes de 3 étudiants). La charge de travail doit correspondre à la taille de l'équipe.
- Aucun étudiant ne peut travailler seul!
- Il est très important que tous les membres de l'équipe travaillent équitablement, car l'évaluation finale tient compte de plusieurs critères, dont la répartition de la tâche de travail.
- Vous devez utiliser le langage C++ ainsi que Visual Studio sous Windows.
- Vous devez faire une conception orientée objets soignée de votre projet.
- L'utilisation des patrons de conception présentée est obligatoire.
- Sans y être contraint, vous pouvez utiliser la petite librairie fournie pour faciliter l'utilisation de la console.
- La date de remise est non négociable.

Ajout personnel

Vous devez faire un ajout personnel au projet. Attention, cet ajout vaut pour 20% de la note finale.

Les points sont attribués selon ces trois critères :

- pertinence;
- niveau de difficulté;
- qualité de la réalisation.

Il est fortement encouragé de discuter avec l'enseignant de vos idées.

Variantes possibles du jeu de la vie

En fait, le jeu de la vie est l'une des variantes possibles des automates cellulaires. Il est donc facile d'imaginer une multitude d'autres variantes. On présente ici quatre exemples.

Modification de la règle

La règle de Conway est symbolisée par B3/S23 qui se traduit par :

- une cellule nait (Born) si elle a 3 voisins actifs, d'où B3
- une cellule survit (Survive) si elle a 2 ou 3 voisins actifs, d'où S23.

On comprend qu'il existe plusieurs règles possibles où toutes les variantes de naissance et de survie peuvent être explorées. Les règles suivantes sont connues et intéressantes :

- *Game Of Life* : B3/S23
- *Highlife* : B36/S23
- *Day & Night* : B3678/S34678

Nombre d'états

Un automate cellulaire peut avoir plus de deux états possibles. Par exemple, il serait envisageable d'avoir trois états et de déterminer la règle pour chaque état et ses voisins.

Voisinages divers

Il est possible d'envisager plusieurs configurations de voisinage et selon diverses pondérations :

- les 4 voisins orthogonaux,
- les 8 voisins immédiats,
- les 8 voisins immédiats pondérés (+1 pour les voisins orthogonaux et $+\sqrt{2}/2$ pour les voisins diagonaux,
- les 24 voisins immédiats (avec ou sans pondérations),
- …

Topologie de l'univers

L'univers peut posséder plusieurs variantes :

- *n* dimensions (n > 0),
- connectivité des voisins de forme diverses (rectangulaire, hexagonal, …)
- sur une forme (par exemple, sur une sphère ou un tore),
- aux bordures mortes ou cycliques.

Références pertinentes pour ce laboratoire

- Automate cellulaire
  - Cellular automaton
  - Game of life (voir ConwayLife pour toute référence et ce lien pour les fichiers RLE)
  - Highlife
  - Day & Night
  - Un simulateur intéressant
- Patron de conception
  - Singleton
  - Itérateur
  - Façade
  - Modèle-Vue-Contrôleur

Rapport

Vous devez produire un rapport correspondant à ceci :

1. Vous devez indiquer clairement qui sont les étudiants ayant travaillé sur le projet.
2. Vous devez répondre à ces questions :
    1. Lecture de fichier :
        a. Quelle(s) est(sont) la(les) classe(s) qui s'occupe(nt) de faire la gestion du fichier d'entré.
        b. Êtes-vous capable de lire un fichier RLE?
        c. Êtes-vous capable de gérer les erreurs de fichiers mentionné?
    2. Où se trouve dans votre code l'endroit où vous spécifiez le dossier où se trouve les fichiers RLE?
3. Veuillez présenter votre ajout personnel :
    1. Quels étaient les intentions initiales, qu'est-ce que fonctionnent bien et qu'est-ce qui fonctionnent moins bien.
    2. Selon vous, quelle note (sur 10) méritez-vous pour ces trois critères :
        a. pertinence
        où 0 est insignifiant
        b. difficulté
        où 0 est trivial
        c. qualité de l'implémentation
        où 0 correspond à un travail dont vous ne parlerez pas en entrevu

Le fichier texte doit être déposé dans le filtre *doc* de votre solution Visual Studio.

Stratégie d'évaluation

L'évaluation se fera en 2 parties. D'abord, l'enseignant évaluera le projet remis et assignera une note de groupe pour le travail. Ensuite, chaque équipe devra remettre un fichier Excel dans lequel sera soigneusement reportée une cote représentant la participation de chaque étudiant dans la réalisation du projet. Cette évaluation est faite en équipe et un consensus doit être trouvé.

Une pondération appliquée sur ces deux évaluations permettra d'assigner les notes finales individuelles.

Ce projet est long et difficile. Il est conçu pour être réalisé en équipe. L'objectif est que chacun prenne sa place et que chacun laisse de la place aux autres.

Ainsi, trois critères sont évalués :

- **participation** (présence en classe, participation active, laisse participer les autres, pas toujours en train d'être sur Facebook ou sur son téléphone, concentré sur le projet, pas en train de faire des travaux pour d'autres cours, …)
- **réalisation** (répartition du travail réalisé : conception, modélisation, rédaction de script, documentation, …)
- **impact** (débrouillardise, initiative, amène des solutions pertinentes, motivation d'équipe, …)

Remise

Vous devez créer un fichier de format `zip` dans lequel vous insérez :

- votre solution Visual Studio **bien nettoyée** `solution.zip`
- votre schéma de conception `schema.pdf`
- votre rapport `rapport.txt`
- le fichier Excel rempli sur la participation active des membres du groupe `evaluation.xlsx`

Vous devez remettre votre projet une seule fois sur Lea après avoir nommé votre fich `void (*action)();` ier :

`NomPrenomEtudiant1_NomPrenomEtudiant2[_NomEtudiantN].zip`

Nettoyage d'une solution de Visual Studio

Visual Studio est un logiciel professionnel destiné à l'industrie. Il possède une multitude de stratégies afin d'optimiser son travail et réduire certaines tâches redondantes (la compilation par exemple). Pour cette raison Visual C++ génère beaucoup de fichiers non essentiels à votre solution. Lors d'une remise, il devient gênant et inutile de remettre tous ces documents qui peuvent souvent faire exploser la taille du projet.

Pour cet exemple, supposons le projet `ProjetABC` se trouvant dans la solution `ProjetABC`. Ainsi :

- le dossier de la solution est : `c:\dev\ProjetABC`
- le dossier du projet est : `c:\dev\ProjetABC\ProjetABC`

Voici une procédure simple pour vous permettre de faire une remise complète et de taille minimum :

1. Nettoyer la solution : menu `Build / Clean Solution`

2. Directement dans les dossiers du projet, effacer les dossiers et tous les fichiers contenus dans les dossiers Debug – Release – x64 :
    - `c:\dev\ProjetABC\ProjetABC\Debug`
    - `c:\dev\ProjetABC\ProjetABC\Release`
    - `c:\dev\ProjetABC\ProjetABC\x64`
3. Directement dans les dossiers de la solution, effacer les dossiers et tous les fichiers contenus dans les dossiers .vs – Debug – Release – x64 :
    - `.vs` (attention, ce dossier est invisible par défaut, il faut spécifier l'affichage des items cachés) **c'est aussi le dossier le plus important à effacer**.
    - `c:\dev\ProjetABC\Debug`
    - `c:\dev\ProjetABC\Release`
    - `c:\dev\ProjetABC\x64`
4. Dans les dossiers du projet, effacer tout ce qui a comme extension :

• `*.aps`	• **`*.sdf`**	• `*.sbr`
• `*.builds`	• `*.aps`	• **`*.sdf`**
• `*.cachefile`	• `*.builds`	• `*.scc`
• **`*.ilk`**	• `*.cachefile`	• `*.svclog`
• `*.meta`	• **`*.ilk`**	• `*.tlb`
• **`*.ncb`**	• `*.meta`	• `*.tli`
• **`*.obj`**	• **`*.ncb`**	• `*.tlh`
• `*.opendb`	• **`*.obj`**	• `*.tmp`
• `*.opensdf`	• `*.opendb`	• `*.tmp_proj`
• **`*.pch`**	• `*.opensdf`	• `*.log`
• **`*.pdb`**	• **`*.pch`**	• **`*.VC.db`**
• `*.pgc`	• **`*.pdb`**	• **`*.VC.VC.opendb`**
• `*.pgd`	• `*.pgc`	• `*.vspscc`
• `*.pidb`	• `*.pgd`	• `*.vssscc`
• `*.rsp`	• `*.pidb`	
• `*.sbr`	• `*.rsp`	

les extensions mises en caractères gras indiquent les fichiers les plus pertinents à retirer à cause de leur nombre et de leur volume potentiel

Il est important de ne pas effacer ces fichiers : `*.sln`, `*.vcxproj`, `*.vcxproj.filters`.

À partir de ce point, votre projet est nettoyé complètement et il est minimal. Vous pouvez compresser votre solution (le dossier `c:\dev\ProjetABC\`) en un seul fichier pour l'archiver ou l'envoyer.

Un dernier conseil, avant de vous mettre à tout effacer, travailler sur une copie de votre projet!

Outils et exemples

Librairie Console

La petite librairie `Console` offre un ensemble de classes facilitant l'utilisation de la console pour réaliser une application graphique rudimentaire.

Plusieurs prises en charge sont réalisées sans toutefois être trop invasives et restreindre le développeur. La console en soi est limitée, mais il est tout de même possible de faire une application fonctionnelle et suffisante.

Les classes importantes sont les suivantes :

```
class Console;           // la console - singleton
class ConsoleContext;    // le contexte de la console – les paramètres

class ConsoleReader;     // classe gérant les entrées de la console
class ConsoleKeyEvent;   // classe représentant un évènement au clavier
class ConsoleKeyFilter;  // classe permettant de filtrer certains évènements
                         // il existe plusieurs filtres dérivant de celui-ci

class ConsoleWriter;     // classe gérant la sortie de la console (écran)
class ConsoleImage;      // classe utilitaire facilitant l'affichage
class ConsoleColor;      // classe utilitaire gérant les couleurs
```

Ce premier exemple met en place un exemple simple et présente quelques concepts fondamentaux.

```
#include <Console/Console.h>
#include <Console/ConsoleKeyFilterUp.h>
#include <Console/ConsoleKeyFilterModifiers.h>

int main()
{
	// Le contexte doit être défini AVANT d'appeler le getInstance pour la
	// première fois. Il est impossible de modifier le contexte après la
	// première instanciation de la classe.
	ConsoleContext consoleContext(100, 50, "Game of life",
	                              10, 10, L"Consolas");
	Console::defineContext(consoleContext);

	// Création de 2 références utilitaires.
	ConsoleKeyReader & reader{ Console::getInstance().keyReader() };
	ConsoleWriter & writer{ Console::getInstance().writer() };

	// Installation de deux filtres pour la lecture des touches au clavier
	reader.installFilter(new ConsoleKeyFilterUp);
	reader.installFilter(new ConsoleKeyFilterModifiers);

	// Boucle éternelle
	ConsoleKeyReader::KeyEvents keyEvents;
	while (true) {
		// Effectue la lecture au clavier
		reader.read(keyEvents);
		// Si une touche a été appuyée
		if (keyEvents.size() > 0) {
			// Affiche une image aléatoire
			writer.randomize();
		}
	}

	returo 0;
}
```

Ce deuxième exemple présente un exemple un peu plus étayé faisant la lecture au clavier et un affichage plus sophistiqué.

```
#include <Console/Console.h>
#include <Console/ConsoleKeyFilterUp.h>
#include <Console/ConsoleKeyFilterModifiers.h>

// Structure utilitaire pour réunir plusieurs informations
struct Theme
{
  ConsoleColor  backgroundColor, textColor,  outlineColor;
  char          backgroundChar , textChar,   outlineChar;
};

// Initialise une image
void setupImage(std::string const & imageName, Theme const & theme)
{
  // Va chercher l'image déjà créée par son nom : imageName
  ConsoleImage & image{ Console::getInstance().writer().image(imageName) };
  // Dessine un cadre et un fond
  image.drawRect(0, 0, image.width(), image.height(),
         theme.outlineChar, theme.outlineColor,
         theme.backgroundChar, theme.backgroundColor);
}

// Dessine du texte centré
void drawCenteredText(  std::string const & imageName,
                        std::string const & message,
                        Theme const & theme)
{
  // Va chercher l'image déjà créée par son nom : imageName
  ConsoleImage & image{ Console::getInstance().writer().image(imageName) };
  // Dessine sur cette image le texte centré
  image.drawText(
    (image.width() - message.length()) / 2,  // position x
    image.height() / 2,                      // position y
    message,                                 // le message
    theme.textColor);                        // la couleur
}

// Affiche l'image finale à l'écran
void showImage(std::string const & imageName,
               std::string const & message,
               Theme const & theme)
{
  // Dessine l'image demandée dans l'image de sortie
  Console::getInstance().writer().image(imageName) =
                    Console::getInstance().writer().image("Output");

  // Écrit le texte dans l'image de sortie
  drawCenteredText("Output", message, theme);

  // Dessine l'image de sortie à l'écran
  Console::getInstance().writer().write("Output");
}
```

```
int main()
{
  // Le contexte doit être défini AVANT d'appeler le getInstance pour la
  // première fois. Il est impossible de modifier le contexte après la
  // première instanciation de la classe.
  ConsoleContext consoleContext(100, 50, "Snake etc...",
              10, 10, L"Consolas");
  Console::defineContext(consoleContext);

  // Création de 2 références utilitaires.
  ConsoleKeyReader & reader{ Console::getInstance().keyReader() };
  ConsoleWriter & writer{ Console::getInstance().writer() };

  // Installation de deux filtres pour la lecture des touches au clavier
  reader.installFilter(new ConsoleKeyFilterUp);
  reader.installFilter(new ConsoleKeyFilterModifiers);

  // Détermine le caractère de remplissagestd :
  char fillChar{ char(219) };
  // Détermine les thèmes utilisées
  Theme deepBlueTheme{
         ConsoleColor(ConsoleColor::tb, ConsoleColor::bb),
         ConsoleColor(ConsoleColor::tW, ConsoleColor::bb),
         ConsoleColor(ConsoleColor::tB, ConsoleColor::bB),
         fillChar,
         fillChar,
         fillChar
       };
  Theme mustardTheme{
         ConsoleColor(ConsoleColor::ty, ConsoleColor::by),
         ConsoleColor(ConsoleColor::tW, ConsoleColor::by),
         ConsoleColor(ConsoleColor::tY, ConsoleColor::by),
         fillChar,
         fillChar,
         'X'
  };

  // Création de deux images de référence et de l'image de sortie
  writer.createImage("DeepBlue");
  writer.createImage("Mustard");
  writer.createImage("Output");

  // Prépare les deux images de référence
  setupImage("DeepBlue", deepBlueTheme);
  setupImage("Mustard", mustardTheme);
```

```
  // Boucle éternelle
  ConsoleKeyReader::KeyEvents keyEvents;
  while (true) {
    // Effectue la lecture au clavier
    reader.read(keyEvents);
    for (auto key : keyEvents) {
      switch (key.keyA()) {
        case 'a' :
             showImage("DeepBlue",
                    "DeepBlue",
                    deepBlueTheme);
          break;
        case 'd' :
          showImage("Mustard",
                 "Mustard",
                    mustardTheme);
          break;
      }
    }
  }

  return 0;
}
```

Lecture dans un fichier

Il existe plusieurs façons de lire les données contenues dans un fichier. L'une des stratégies possibles consiste à créer un flux de données (entrant), d'en faire la lecture ligne par ligne et, finalement, d'analyser chacune des lignes.

Voici un exemple de cette stratégie.

```
#include <fstream>    // requis pour ifstream
#include <string>     // requis pour string et getline
#include <array>      // requis pour array
#include <algorithm>  // requis pour for_each

// Compte le nombre d'occurrence pour chaque
// caratère de la table ascii dans un fichier
bool charOccurrenceInFile( std::string const & fileName,
                           std::array<size_t, 256> & charInFile)
{
   // Réinitialise le tableau d'occurrence à 0
   std::fill(charInFile.begin(), charInFile.end(), 0);

   // Ouvre le flux de données entrant en lecture seule : le fichier
   std::ifstream iStream(fileName, std::ios::in);

   // Si le fichier est ouvert
   if (iStream.is_open()) {
      // Tant que toutes les lignes ne sont pas lues
      while (!iStream.eof()) {
         // Chaine de caractères qui contiendra la ligne courante
         std::string line;
         // Lecture de la ligne courante
         //  -> le flux entrant passe à la ligne suivante
         std::getline(iStream, line);
         // Pour chaque caractère de la ligne,
         // on incrémente le tableau d'occurrence
         std::for_each(line.begin(), line.end(),
                   [&charInFile](char c) { ++charInFile[(unsigned char)c]; });
      }
      return true;
   }

   return false;
}

// Exemple d'appel de fonction
std::array<size_t, 256> charInFile;
bool success{ charOccurrenceInFile("c:\\myfile.txt", charInFile) };
```

Regex avec std::regex

Les expressions régulières sont supportées à même la librairie standard depuis C++11. L'usage des regex est un vaste sujet et on suppose ici que vous avez déjà des connaissances de base pour leur usage. Voici deux exemples d'utilisation.

Exemple validant si une chaîne de caractère est conforme à une expression régulière.

```
#include <list>     // requis pour la liste
#include <string>   // requis pour la chaîne de caractères
#include <regex>    // requis pour l'expression régulière

// Transfert les chaînes de caractères d'une liste dans une autre
// liste. Le transfert est conditionnel à la conformité de la
// chaîne de caractères à une expression régulière.
void copyMatch(  std::list<std::string> const & from,
                 std::list<std::string> & to,
                 std::string const & regexMatch )
{
   // Vide la liste de résultats
   to.clear();
   // Construit le regex
   std::regex matchRegex(regexMatch);
   // Pour toutes les chaînes de caractères sources
   for (auto const & str : from) {
      // Si la chaîne de caractères est conforme
      if (std::regex_match(str, matchRegex)) {
         // Ajoute une copie
         to.push_back(str);
      }
   }
}

// Exemple d'appel de fonction
std::list<std::string> quote;
quote.push_back("Je pense, donc je suis. [Descartes]");
quote.push_back("Connais-toi toi-même. [Socrate]");
quote.push_back("Ce que je sais, c'est que je ne sais rien. [Socrate]");
quote.push_back("Si Dieu n'existait pas, il faudrait l'inventer. [Voltaire]");
quote.push_back("Deviens ce que tu es. [Nietzsche]");
quote.push_back("Pourquoi y a t il quelque chose plutôt que rien?. [Leibniz]");

std::list<std::string> result;
copyMatch(quote, result, ".*\\[Socrate\\]");
```

Exemple faisant l'extraction (la capture) d'une information à partir d'une expression régulière.

```
// Fonction faisant l'extraction d'une sous chaîne
// de caractères suivant la capture d'une expression
// régulière.
std::string extractStr( std::string const & source,
                        std::string const & regexCapture)
{
   // Construit le regex
   std::regex captureRegex(regexCapture);
   // Construit la classe de capture
   std::smatch captureMatch;
   // S'il y a une concordance avec le regex
   if (std::regex_match(source, captureMatch, captureRegex)) {
      // S'il y a bien un seul "match"
      // Le premier résultat est la chaîne de caractères originale
      // Le deuxième résultat est celui la première capture
      // ...
      if (captureMatch.size() == 2) {
         // Retourne le résultat de la capture
         return captureMatch[1].str();
      }
   }
   // Retourne une chaîne vide
   return std::string();
}

// Exemple d'appel de fonction
std::string quote(extractStr(
          "Je pense, donc je suis. [Descartes]",
          "(.*) \\[Descartes\\]"));
```

Accès aux ressources système (fichiers et dossier) avec std::filesystem

L'accès aux fichiers système est supportées à même la librairie standard depuis C++17. Visual Studio 2017 ne supporte que partiellement la norme C++17. Toutefois, il est possible d'utiliser la version expérimentale de cette librairie.

Voici un exemple de code qui vous permet de retrouver tous les fichiers inclus dans un dossier. Le namespace filesystem vous donne accès aux fonctions et classes utilitaires de la librairie. La classe std ::path vous donne accès à plusieurs informations sur une entrée de votre répertoire comme un fichier (chemin d'accès, nom du fichier, nom de l'extension, ...).

```
//#include <filesystem>             // lorsque la librairie sera complètement
                                    // supportée on mettra cette ligne...
#include <experimental/filesystem>  // ...mais pour l'instant on doit mettre
                                    // celle-ci
namespace efs = std::experimental::filesystem;
                                    // ceci déclare un alias de namespace, très
                                    // pratique et permet d'éviter d'écrire un
                                    // long namespace à chaque fois
#include <list>                     // requis pour la liste
#include <string>                   // requis pour la chaîne de caractères

// On crée une structure permettant de stocker plusieurs
// informations relatives à un fichier.
struct file_info
{
   file_info(  std::string const & c,
               std::string const & r,
               std::string const & p,
               std::string const & f,
               std::string const & e)
      :  complete(c),
         root(r),
         path(p),
         filename(f),
         extension(e) {
   }
   std::string complete;
   std::string root;
   std::string path;
   std::string filename;
   std::string extension;
};
```

```
// Fonctionnant retournant une liste d'information sur les fichiers inclus
// dans le dossier passé en paramètre. Le parcours est non récursif.
std::list<file_info> getFileInfo(std::string const & pathName)
{
   // crée la liste à retourner
   std::list<file_info> info;
   // parcours tout le dossier (non-récursivement)
   for (auto const & pathIterator : efs::directory_iterator(pathName)) {
        // valide si l'entrée est bien un fichier
        if (efs::is_regular_file(pathIterator)) {
           // crée une référence pour simplifier l'écriture qui suit
           efs::path const & filePath{ pathIterator.path() };
           // insère un nouvel item dans la liste
           info.push_back(file_info(
                filePath.string(),                  // complete
                filePath.root_name().string(),      // root
                filePath.parent_path().string(),    // path
                filePath.stem().string(),           // filename
                filePath.extension().string()));    // extension
      }
   }
   return info;
}

// Exemple d'appel de fonction
std::list<file_info> fileInfoInDir(getFileInfo("c:\\myDir\\"));
```

On remarque que la notion d'itérateur est encore utilisée : la classe directory_iterator est un itérateur.

Critères d'évaluation

- Fonctionnalité et robustesse 40
  - Simulation fonctionnelle 16
    - simulation 8
    - initialise aléatoirement 1
    - initialise par patron centré 2
    - supporte adéquatement le changement de règle 2
    - supporte adéquatement la gestion des effets de bord 3
  - Lecture du fichier 12
    - Capable de lire le fichier RLE 4
    - Gestion des erreurs :
      - validation des lignes d'entête 1
      - l'entête peut posséder entre 0 et n lignes 1
      - présence obligatoire de la ligne de transition 2
      - les tailles x et y sont positives 1
      - la séquence ne déborde pas la taille du patron (x et y) 1
      - le caractères de fin de séquence est validé 1
      - le contenu du fichier est ignoré après le caractère de fin de séq.1
  - Interaction usager 12
    - pause et continue 1
    - détermination de la vitesse (1 à 9) 1
    - initialisation aléatoire (6 modes) 1
    - initialisation par un fichier RLE 1
    - parcours des fichiers : précédent, courant et suivant 1
    - changement de la couleur des cellules actives (7 couleurs) 2
    - changement de la couleur des cellules inactives (2 modes) 2
    - gestion des effets de bord (2 modes) 1
    - changement de la règle (4 règles) 2
- Conception et qualité de la solution 30
  - Choix approprié de la structure de données fondamentale 5
  - Découpage orienté objet approprié pour les trois modules principaux : 15
    - Simulation 5
    - Lecture du fichier RLE 5
    - Interface usager 5
  - Patrons de conception imposés 10
    - utilisation appropriée du singleton (Console) 1
    - utilisation appropriée des itérateurs 3
    - façade :
      - pertinence 2
      - qualité 2
      - utilisation 2
- Ajout personnel 20
  - pertinence 6

- niveau de difficulté 6
- qualité de la réalisation 8

- Respect des contraintes 5
  - Un duo de fichiers par classe 1
  - Entête appropriée pour chaque *.h 1
  - Norme de codage 0.5
  - Choix des noms (types, fonctions, variables, ...) 1
  - Commentaires 0.5
  - Lisibilité et clarté du code 1
- Rapport 5
  - Diagramme de classe 1.5
  - 1 1.0
  - 2.1.a 0.5
  - 2.1.b 0.5
  - 3.1 1.0
  - 3.2 0.5