Développement
en environnement de base de données

Projet 1 – Version préliminaire

Table des matières

Énoncé

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

Objectif général

Ce premier laboratoire vise la réalisation d'un projet complet visant la 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++.

Le 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 sera de dimension finie pour la simulation,
- il peut être borné ou circulaire.

Cellule de l'univers

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

Pour chaque évolution :

- toutes les cellules sont analysée selon :
  - leur état,
  - le nombre de cellule active parmi les 8 voisins immédiats;
- la règle de Conway dit :

    - une cellule inactive : devient active si elle possède 3 voisins actifs
    - une cellule active : reste active si elle possède 2 ou 3 voisins actifs
- 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 aléatoire de patrons connus
- par la disposition de patrons connus

Variantes du jeu de la vie

Il existe plusieurs variantes possibles mais on retiendra ces quatre possibilités principales.

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
- une cellule survit (Survive) si elle a 2 ou 3 voisins actifs.

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.

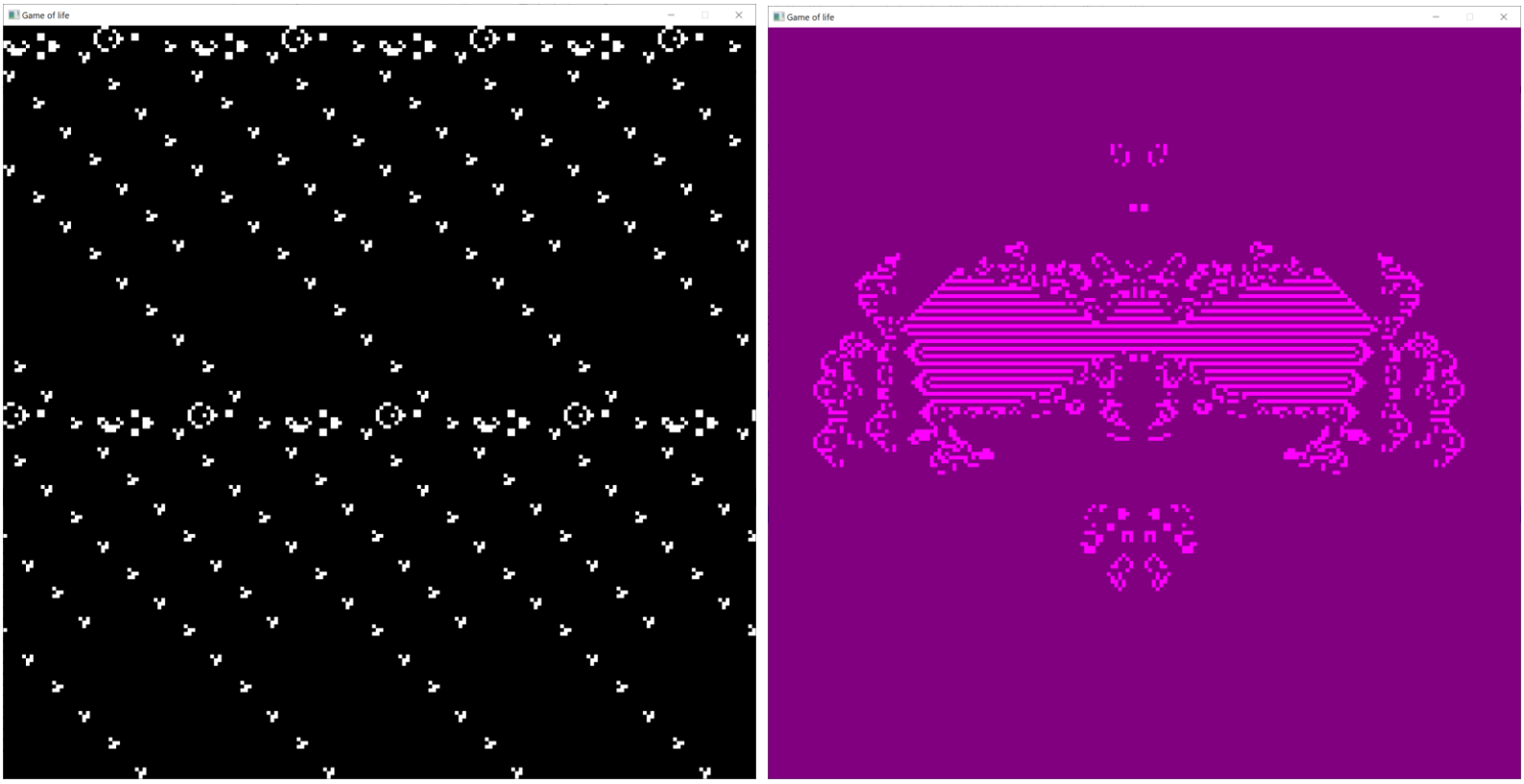
Présentation générale du projet

Le logiciel réalisé doit offrir :

- la lecture de fichier RLE;
- un outil de visualisation de la simulation;
- une interface utilisateur permettant les quelques manipulations obligatoires des options :
    - univers aux bordures mortes ou cycliques,
    - règles : B3/S23, B36/S23, B3678/S34678 ou 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.

Interface utilisateur

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



Exemple de l'interface usager : simulation en cours

Les commandes à réaliser sont :

- Barre d'espacement : pause et reprise
- 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 aux bordures :
    - 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. Vous devez être en mesure de lire ces fichiers de façon à initialiser votre univers.

De plus, lors de l'initialisation, vous devez centrer le patron dans l'univers.

Le fichier RLE est constitué ainsi :

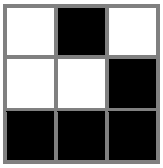
- Section en-tête
    - #C [commentaires]
    - #c [commentaires]
    - #N [nom du patron]

    - #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]
- Ligne de transition :
    - Une ligne offrant cette syntaxe : x = [width], y = [height], rule = [abc]
    - La troisième partie est optionnelle alors que les 2 premières sont obligatoires.
- Données du patron sous format RLE (*Run Length Encoded*):
    - b = cellule inactive
    - o = cellule active
    - nombre = le nombre de cellule du type qui suit (un type sans nombre indique qu'il n'y a qu'une cellule de ce type)
    - $ = fin de ligne
    - ! = fin du fichier

Ainsi, le fichier suivant :

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

Donne le patron suivant :



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 (en terme de rapidité) ou l'ajout de fonctionnalités superflues mais plutôt la modularité et la qualité du logiciel.

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 classe 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).

Afin de vous assurer de bien pratiquer certaines notions liées au paradigme orienté objet, on vous demande de mettre en pratique 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 le voisinage d'un état. Ainsi, cette classe prendra 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.

Programmation C++

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

- garde de page (#ifndef …)
- 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 et 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 (ne pas utiliser d'héritage et de polymorphisme)
- utilisation des classes et fonctions de la libraire standard
- …

Un indice : aucune allocation dynamique explicite n'est requise de votre part pour ce projet.

Autres contraintes

Vous devez respecter ces contraintes:

- 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++;
    - 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.

Outils fournis

Librairie Console

La petite librairie `Console` offre un ensemble de classes facilitant l'utilisation de la console pour réaliser une application similaire à un jeu vidéo.

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>

void 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();
		}
	}
}
```

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;
	ConsoleColor textColor;
	ConsoleColor outlineColor;
	char backgroundChar;
	char textChar;
	char 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 : centré
		image.height() / 2,                     // position y : centré
		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().push(imageName, "Output");

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

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

void 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 remplissage
```

```
	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;
			}
		}
	}
}
```

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 ...
```

Tous les noms que vous donnez doivent servir d'auto-documentation. Il est attendu que vous fassiez une documentation minimum pour chaque fonction. Ajoutez des commentaires lorsque nécessaire seulement pour ce projet.

On retrouve les concepts et exemples dans les *.h. Les commentaires dans les *.cpp représentent davantage des détails techniques.

Surtout, ne pas mettre de commentaires creux :

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

Rapport

Vous devez produire un rapport correspondant à ceci :

- Fichier texte déposé dans le dossier doc dans votre solution.
- Vous devez indiquer clairement qui sont les étudiants ayant travaillé sur le projet.
- Vous devez répondre à ces questions :
    - À venir…
- Concernant votre ajout personnel :
    - Donnez une description.
    - Quels étaient les intentions initiales, qu'est-ce que fonctionnent bien et qu'est-ce qui fonctionnent moins bien.
    - Selon vous, quelle note (sur 10) méritez-vous pour ces trois critères :
        - pertinence
          où 0 est insignifiant
        - difficulté
          où 0 est trivial

- qualité
  où 0 correspond à un travail dont vous ne parlerez pas en entrevu

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`

Références

- Automate cellulaire
  - Cellular automaton
  - Game of life (voir ConwayLife pour toute référence)
  - Highlife
  - Day & Night
- Patron de conception
  - Singleton
  - Itérateur
  - Façade