# LO21 - Printemps 2021 - Projet Cellulut

Dans ce projet, il s'agit de concevoir et développer l'application CELLULUT, destinée à simuler des automates cellulaires à 2 dimensions. Avant de commencer, nous vous invitons à découvrir le mondes très riche des automates cellulaires en consultant, par exemple, la page WIKIPÉDIA correspondante et les documents mis à disposition.

## 1 Description des concepts principaux

Dans ce document, sont présentées les spécifications du fonctionnement de l'application demandée. Ces spécifications vous laissent volontairement des choix de conception, tant fonctionnels que conceptuels et technologiques. Quels que soient les choix et adaptations que vous ferez, vous prendrez garde de les exposer et de les justifier dans le rapport rendu avec le projet. Il peut manquer des spécifications. Dans ce cas, faites un choix en l'exposant clairement dans votre rapport.

### 1.1 Différents types d'automates cellulaires

De façon générale, un automate cellulaire est caractérisé par :

- un *réseau* régulier $\mathcal{L}$ (un pavage périodique) dont les éléments s'appellent les cellules ;
- un ensemble fini $\mathcal{Q}$ d'*états* que peuvent prendre les cellules du réseau ;
- un *voisinage* $\mathcal{N}$ qui permet de définir l'ensemble des cellules voisines à prendre en compte dans le calcul de l'état suivant d'une cellule ;
- une *fonction de transition* $\mathcal{F}$ qui définit le calcul de l'état suivant d'une cellule en fonction de son propre état et de l'état des cellules de son voisinage.

Il existe une infinité de types d'automates cellulaires différents. Votre simulateur ne pourra donc pas simuler tous les automates cellulaires possibles.

### 1.2 Réseau, configurations, frontières

Dans cette application, nous nous restreignons aux réseaux à 2 dimensions dans lesquels les cellules sont des carrés. Un réseau est caractérisé par sa largeur et sa hauteur en nombre de cellules : c'est donc un réseau fini de forme rectangulaire. Chaque cellule est caractérisée par ses coordonnées sur ce rectangle. Une *configuration* $\mathcal{C}_t$ est une fonction qui associe à chaque cellule du réseau un état. Lorsque l'on simule un automate cellulaire, on part d'une configuration initiale $\mathcal{C}_0$. À chaque étape de la simulation, l'horloge est incrémentée et la configuration $C_{t+1}$ à l'instant $t+1$ est calculée à partir de la configuration $\mathcal{C}_t$ à l'instant $t$.
Chaque *bord* du rectangle s'appelle une *frontière*. Dans cette application, on utilisera des frontières périodiques : le réseau est considéré comme un *tore* où les cellules d'un bord sont considérées comme adjacentes aux cellules du bord opposé.

### 1.3 L'alphabet

Dans l'application, les cellules des automates pourront prendre jusqu'à au moins 8 états différents (mais vous pouvez faire plus), ce nombre pouvant être configuré. Chaque état sera caractérisé par un indice, une couleur et éventuellement par un label. La couleur et le label de chaque état sera configurable dans une vue de l'application.
Par exemple, dans l'automate *Life Game* de Conway (1970), chaque cellule a deux états possibles. L'un d'eux est blanc et on peut lui donner le label « vivant ». L'autre est noir et on peut lui donner le label « mort » (évidemment, d'autres couleurs peuvent être choisies). Dans l'automate *Brian's brain*, chaque cellule à 3 états possible : vert (« repos »), rouge (« excité »), jaune (« réfractaire »). Dans l'automate

*Wire World* de Brian Silverman (1987), chaque cellule a 4 états possibles : « vide »), « conducteur », « électron tête », « électron queue ». L'automate *Universal Constructor* de von Neuman (1940) en compte 29 !

## 1.4 Les voisinages

Le voisinage d'une cellule $x$ de coordonnées $(i, j)$ est l'ensemble des cellules avec lesquelles $x$ pourra interagir. L'application permettra de définir et configurer un voisinage en proposant au moins les voisinages classiques suivants :

- voisinage de von Neumann : les voisines d'une celulle sont celles adjacentes horizontalement et verticalement (au nord, au sud, à l'ouest, à l'est), *i.e.* $\mathcal{N}_{(i,j)} = \{(k, l)|abs(k-i)+abs(l-j) \leq 1\}$ ;
- voisinage de von Neumann généralisé avec un rayon $r$ : $\mathcal{N}_{(i,j)} = \{(k, l)|abs(k-i)+abs(l-j) \leq r\}$ ;
- voisinage de Moore : on ajoute au vosinnage de von Neuman, les cellules adjacentes en diagonale (nord ouest, nord est, sud est, sud ouest), *i.e.* $\mathcal{N}_{(i,j)} = \{(k, l)|abs(k-i) \leq 1$ et $abs(l-j) \leq 1\}$ ;
- voisinage de Moore généralisé avec un rayon $r$ : $\mathcal{N}_{(i,j)} = \{(k, l)|abs(k-i) \leq r$ et $abs(l-j) \leq r\}$ ;
- voisinage arbitraire : liste des coordonnées relatives par rapport à $(i, j)$ des cellules faisant partie du voisinage.

Par exemple *Life Game* utilise le voisinnage de Moore alors que *Langston's Loop* (1984) utilise le voisinnage de von Neumann. Certains voisinage sont plus compliqués (voir par exemple *le voisinage de Margolus* de 4 cellules qui alterne entre deux positions suivant la parité de l'horologe).

## 1.5 Les fonctions (règles) de transition

Le rôle de la fonction de transition $\mathcal{F}$ est de calculer $\mathcal{C}_{t+1}$ à partir de $\mathcal{C}_t$. Il en existe de nombreuses dans la littérature. Voici quelques exemples :

- Dans *Life Game* : une cellule morte possédant exactement 3 voisines vivante devient vivante (elle naît) ; une cellule vivante reste vivante (elle survit) si elle possède deux ou trois voisines vivantes. Dans tous les autres cas, la cellule devient morte (elle meurt).
- Dans *Brian's brain* : les cellules excitées deviennent toujours réfractaires au pas de temps suivant ; les cellules réfractaires reviennent toujours au repos au pas de temps suivant ; une cellule au repos devient excitée au pas de temps suivant si elle a exactement 2 cellules voisines (voisinage de Moore) excitées.
- Dans *l'automate circulaire de Griffeath* qui possède 4 états $Q = \{0$ (jaune), $1$ (orange clair), $2$ (orange foncé), $3$ (rouge)$\}$, une cellule passe de l'état $i$ à $i + 1$ (modulo 4) dès que $i + 1$ (modulo 4) est présent dans au moins 3 cellules voisines.
- Dans *Langston's Loop*, la règle est beaucoup plus compliquée.

La règle de transition peut être définie en extension en précisant l'état résultant de chaque configuration possible du voisinage (voir par exemple la table de transiiton de l'automate *Langston's Loop*). Elle peut être définie en intension en exprimant des propriétés ou des conditions comme dans *Life Game*, *Brian's brain*, *Langston's Loop*.

La plupart des automates de la littérature utilisent des règles qui ont été généralisées par la suite. Par exemple les automates *Life Game* et *Day & Night* utilisent tous les deux des cas particuliers d'une règle plus générale qui fait partie de la classe des règles dites *totalistiques*. Dans *l'automate circulaire de Griffeath*, on utilise une règle dite *outer-totalistique* (l'état suivant de la cellule dépend aussi de son propre état). De nombreuses classes de règles ont été ainsi identifiées.

## 1.6 Les structures

Pour un automate donné, on appelle *structure* un motif caractéristique qui apparait dans l'univers de cet automate. Un motif peut se distinguer par une forme évocatrice ou un comportement particulier.

Par exemple, l'automate *Game Life* possède un certain nombres de structures qui ont été classifiées :

- Les structures stables sont des ensembles de cellules ayant stoppé toute évolution : elles sont dans un état stationnaire et n'évoluent plus tant qu'aucun élément perturbateur n'apparaît dans leur voisinage.
- Les oscillateurs sont des structures qui se transforment de manière cyclique, en revêtant plusieurs formes différentes avant de retrouver leur état initial.
- Les vaisseaux sont des structures capables, après un certain nombre de générations, de produire une copie d'elles-mêmes, mais décalées dans l'univers du jeu.
- Les puffeurs sont des structures qui se déplacent en laissant derrière elles une traînée constituée de débris.
- Les canons sont des oscillateurs lâchant des débris, capables de produire des vaisseaux, à un rythme variable.
- etc.

# 2 Manipulation - Éléments d'interface

Le simulateur doit permettre de simuler un automate cellulaire, *i.e.* visionner dynamiquement l'évolution d'un automate de génération en génération. Ainsi les fonctionnalités élémentaires suivantes sont attendues :

- paramétrer une simulation ;
- définir une configuration initiale ;
- lecture, arrêt, remise à zéro d'une simulation.
- charger/enregistrer un paramétrage ou une configuration pour un paramétrage donné.

Vous êtes libre d'organiser votre interface du moment qu'elle permet de piloter facilement l'application.

## 2.1 Paramétrage d'une simulation

L'application devra permettre de paramétrer une simulation, c'est à dire de :

- paramétrer (manuellement ou à partir d'un modèle) $\mathcal{Q}$ : l'ensemble des états possible des cellules de l'automate ;
- définir les dimensions du réseau $\mathcal{L}$, les conditions aux limites sur les frontières, et l'état initial de chaque cellule (choisi ou aléatoire) ;
- choisir et configurer un voisinage $\mathcal{N}$ ;
- choisir et configurer une fonction de transition $\mathcal{F}$.

Vous êtes libres d'ajouter d'autres éléments de configuration si ceux-ci vous paraissent pertinents.
Dans la suite, on appelera « *modèle d'automate* » un triplet (ensemble d'état, voisinage, règle de transition) bien identifié et parémétré. Ainsi, *Life Game*, *Brian's brain*, *Langston's Loop*, *Langston's Ant*, *l'automate circulaire de Griffeath* sont des modèles. L'application devra permettre de charger des modèles, c'est à dire de paramétrer entièrement une simulation en choisissant un modèle parmi une bibliothèque. L'application devra aussi permettre de sauvegarder un modèle.
Un modèle est caractérisé par un titre permettant de l'identifier, d'une description et éventuellement d'un auteur et d'une année de création (lorsque ces éléments sont connus).
Initialement, votre bibliothèque de modèles devra contenir les modèles suivants :

- Game Life
- Brian's brain
- L'automate circulaire de Griffeath
- Langston's loop

### 2.1.1 Définir/charger/enregistrer une configuration

L'application doit permettre de définir les dimensions du réseau et l'état initial de chaque cellule. Cette définition peut se faire :

- manuellement : les éléments sont saisis par l'utilisateur ;
- aléatoirement : les dimensions sont saisies par l'utilisateur mais la configuration (l'état initial de chaque cellule) est tirée aléatoirement ;
- par chargement : l'utilisateur peut choisir une configuration parmis un ensemble de configurations liées au modèle courant (si elle en possède).

Une fois qu'une configuration est tirée aléatoirement ou qu'une configuration préexistante est chargée, l'utilisateur peut toujours en modifier manuellement les caractéristiques. L'utilisateur peut à chaque instant enregistrer une configuration pour le modèle courant. Si le modèle courant est un nouveau modèle saisi manuellement par l'utilisateur, il doit au préalable enregistrer le modèle. En effet, chaque configuration est nécessairement associée à un modèle particulier.
Vous fournirez au moins une configuration enregistrée (parmi les structures connues) pour chacun des modèles que doit contenir initialement votre bibliothèque. Chaque configuration aura un titre permettant de l'identifier dans la bibliothèque.

### 2.1.2 Exécution de la simulation

Il peut y avoir plusieurs modes de simulation :

- Mode automatique : dans ce mode, les transitions entre les différents états de l'automate sont réalisées automatiquement après un pas de temps configurable (vitesse de lecture). Dans ce mode, l'utilisateur peut choisir d'arrêter et de reprendre l'exécution de la simulation.
- Mode pas à pas : Dans ce mode, l'utilisateur déclenche manuellement le passage de l'état associé à l'instant t à l'état associé à l'instant t+1

Dans tous les modes, le simulateur garde en mémoire un certain nombre des dernières configurations parcourues. Ce nombre est paramétrable. Cela permet de revenir en arrière de plusieurs pas d'horloge. Dans tous les modes, le simulateur peut revenir sur la configuration initialement définie. Dans le cas ou après un certain nombre de pas d'horloge le simulateur revenait à cette configuration initiale, le simulateur fait apparaître la période de cette instance (le nombre de pas d'horloge nécessaire pour revenir à une configuration). L'utilisateur peut à chaque instant enregistrer la configuration courante pour le modèle courant.

## 2.2 Sauvegarde du contexte (optionnelle)

Au démarrage de l'application, l'état de l'application, les paramétrages présents lors de la dernière exécution sont récupérés.

## 2.3 Bibliothèque de structures (optionnelle)

Vous pouvez enrichir les modèles en leur associant une bibliothèques de structures (sous formes de petites configurations) qui peuvent être copier-collées pour faciliter la constitution d'une configuration initiale. Une structure est aussi caractérisée par un titre permettant de l'identifier, d'une description et éventuellement d'un auteur et d'une année de création (lorsque ces éléments sont connus).

## 2.4 Evolution de l'application

L'architecture de votre application devra permettre d'intégrer de nouveaux élément sans remettre en cause l'application. Les choix de conception devront donc permettre de rendre l'application évolutive et notamment garantir la facilité (sans impacter le reste du programme) d'ajout des composants suivants :

— ajout de nouveaux voisinnages
— ajout de nouvelles règles
— éventuellement l'ajouts des éléments d'IHM de paramétrage liés à des nouvelles règles ou voisinnage.

Vous devrez illustrer cette possibilité en expliquant comment implémenter un autre automate de votre choix parmi :

— Wire world
— Langton's Ant

# 3 Consignes

— Le projet est à effectuer en groupe de 5 ou 6 étudiants (du même groupe de TD).
— En plus des instructions standards du `C++/C++11/C++14`, vous pouvez utiliser l'ensemble des bibliothèques standards du `C++/C++11/C++14`.
— Ne faites les parties "optionnelles" qu'après avoir fait le reste. Si ces parties ne sont pas faites, il n'y a aucune pénalité dans la notation.

# 4 Livrables attendus

## 4.1 Rapports intermédiaires

Pour mener ce projet, vous devrez vous organiser de manière efficace. Il sera donc important de tenter de réfléchir aux tâches que vous allez devoir réaliser en vous posant pour chacune d'elle les questions suivantes :

— Cette tâche peut-elle être encore découpée en sous-tâches ?
— Quelle est la complexité de la tâche ?
— Quelle est la durée estimée pour réaliser cette tâche ?
— Quelles sont les relations de dépendance entre cette tâche et les autres ?

Vous devrez ensuite prioriser vos tâches (indipensable, importante, utile, bonus) et les répartir entre les différents membres du groupe de projet.
Au cours du projet, il vous est demandé de rendre compte de votre organisation. vous devrez ainsi rendre 3 courts comptes-rendus (2 à 3 pages maximum) lors des semaines suivantes :

1. la semaine du 5 avril ;
2. la semaine du 10 mai ;
3. la semaine du 31 mai.

Chaque rapport comportera :

— La liste des tâches mises à jour (tâches a priori pour le rapport 1) avec l'estimations de leur durée en mettant en évidence les nouvelles taches par rapport au précédent compte-rendu (rapports 2 et 3).
— Une répartition a priori des tâches qui restent à faire entre les différents membres du groupe de projet.
— L'état d'avancement par rapport au précédent compte-rendu (pour le compte rendu 2 et 3) : la répartition entre les membres du groupe et la durée a posteriori des tâches déjà effectuées.

Bien que la liste des tâches s'affinera au cours du temps, vous tâcherez d'être précis sur les tâches dont les échéances sont à court et moyen-terme par rapport au rendu du rapport.

## 4.2 Livrable final

Le livrable final est composé des éléments suivants :

- **Code source :** l'ensemble du code source du projet. Attention, **ne pas fournir d'exécutable ou de fichier objet**.
- **Documentation** : une documentation complète en `html` générée avec `Doxygen`.
- **Video de présentation avec commentaires audio** : une courte video de présentation dans laquelle vous filmerez et commenterez votre logiciel afin de démontrer le bon fonctionnement de chaque fonctionnalité attendue (max 10 min, 99 Mo).
- **Rapport :** Un rapport en format `.pdf` (max 15 à 20 pages) composé des parties :
  - un bref résumé de ce que permet votre application (en précisant parmi les opérations attendues celles qui ont été implémentées et celles qui ne l'ont pas été) ;
  - la description de votre architecture en justifiant les choix d'architecture ;
  - une argumentation détaillée où vous montrez que votre architecture permet facilement des évolutions (notamment au travers de l'exemple choisi).
  - une description détaillée du planning constaté de votre projet ;
  - une description détaillée de la contribution personnelle de chacun des membres du groupe sur les différents livrables (cette partie sera notamment utilisée pour la notation). Vous évaluerez en % la part de contribution de chaque membre sur l'ensemble. Chaque membre devra aussi reporter une évaluation du nombre d'heures de travail qu'il a consacré au projet.

L'ensemble des livrables est **à rendre avant le 13 juin à 23h59 au plus tard (partout dans le monde)**. Les éléments du livrable devront être déposés sur moodle dans la partie prévue à cet effet.

# 5 Évaluation

Le barême de l'évaluation du projet est comme suit :

- **Couverture des fonctionnalités demandées** : 4 points
- **Couverture des modèles demandés** : 4 points
- **Choix de conception et architecture** : 5 points. En particulier sera évaluée la capacité de l'architecture à s'adapter aux changements.
- **Evaluation des livrables** : 5 points (code source, rapport)
- **Respect des consignes sur les livrables** : 2 points (échéance, présence de l'ensemble des livrables et respect des consignes sur les livrables).

Remarque : une note inférieure ou égale à 8/20 au projet est éliminatoire pour l'obtention de l'UV.

# 6 Conseils

- Il est fortement recommandé d'utiliser un logiciel de gestion de versions afin de faciliter le travail collaboratif.
- Plusieurs TDs utilisent le thème du projet afin de commencer à vous familiariser avec les différentes entités de l'application qui est à développer. On ne perdra pas de vue que les questions développées dans ces TDs ne constituent pas une architecture pour le projet. Celle-ci devra être complètement retravaillée en tenant compte de l'ensemble des entités du sujet.
- La partie difficile du projet est la conception de votre architecture : c'est là dessus qu'il faut concentrer vos efforts et passer le plus de temps au départ.
- Il est conseillé d'étudier au moins les design patterns suivants qui pourraient être utiles pour élaborer l'architecture de votre projet : decorator, factory, abstract factory, builder, bridge, composite, iterator, template method, adapter, visitor, strategy, facade, memento. En plus de vous donner des idées de conception, cette étude vous permettra de vous approprier les principaux modèles de conception.
- Pour la persistance des informations, vous êtes libres d'élaborer vos formats de fichier. Il est tout de même conseillé d'utiliser XML et d'utiliser les outils XML de Qt.

— Au lieu d'utiliser des fichiers, vous pouvez utiliser un SGBD comme SQLite.
— L'apparence de l'application ne sera pas prise en compte dans la notation. Soyez avant tout fonctionnels. Ca peut être moche.