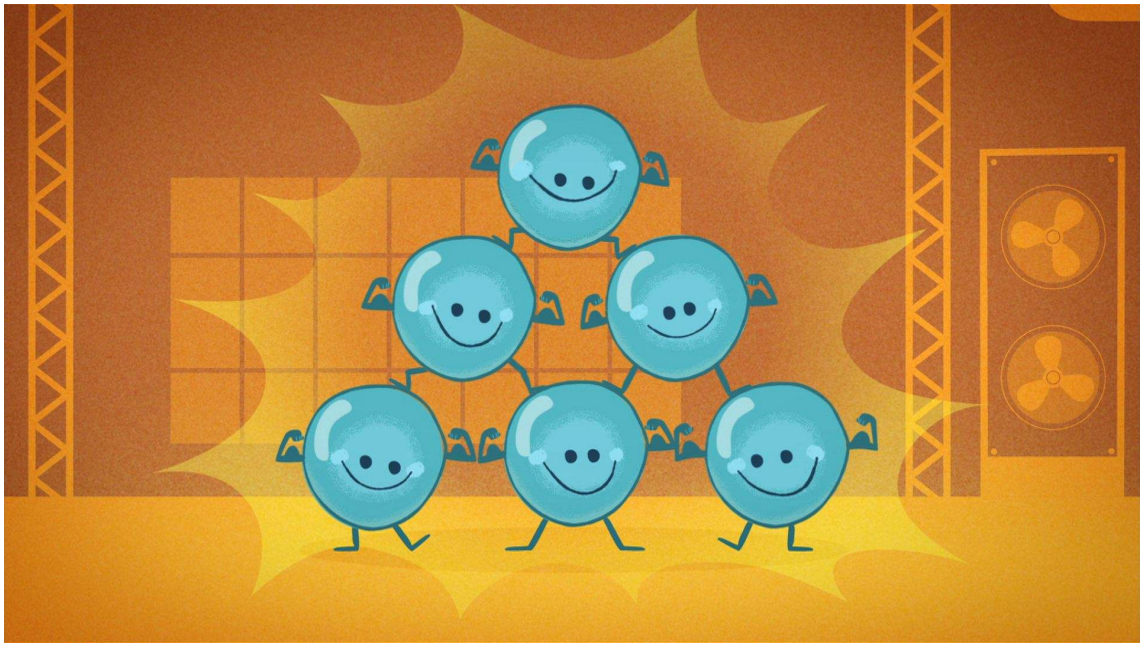

Madjan

Manuel d'utilisation

Jeremy Bardon
Nicolas Brondin
Adrien Garandel
Maxime Pauvert
David Perrai
Anthony Pena

15 décembre 2014

Sommaire

1 Présentation du logiciel

Madjan est un logiciel simulation d'automate cellulaire adapté du jeux de la vie [1]. Cette adaptation comme le "jeu" d'origine n'est pas à proprement parlé un jeux vidéo mais un automate cellulaire définie sur une grille carrée a deux dimensions. Les cellules de la grille contiennent des pions qui naitront ou mourront durant les différentes générations. L'utilisateur n'aura aucune interaction durant le processus du jeux sauf pour le lancement et le paramétrage. La distinction de cette adaption se fait dans le comportement qui est définie à l'aide de plugins [2]. Cette application permet, en fonction d'un paramétrage adéquate,de modéliser de nombreux processus tels que la propagation d'un incendie de fôret, dissémination de virus, etc...

1. inventé par John Horton Conway en 1970
2. module chargé dynamiquement durant l'éxécution

2 Installation

Notre application est multi-plateforme vous trouverez ci-dessous les différents moyens pour lancer l'application **Madjan** en fonction de votre système d'exploitation.

2.1 Prérequis

Pour pouvoir compiler l'application *Madjan* il est nécessaire qu'il soit installé sur votre poste **QT 4.8** disponible à cette adresse QT4.8. Vous pouvez choisir n'importe quelle version (de la 4.8.0 à la 4.8.6). La compilation sur le système d'exploitation Windows n'est pas nécessaire un executable est déjà présent, voir chapitre 2.3.3. Cependant pour les utilisateurs avancés il est possible de recompiler l'application, voir chapitre 2.2.2.

2.2 Compilation

2.2.1 Compilation sous Mac et linux

Un script d'installation destiné à la compilation de l'application est présent dans le dossier source Madjan.
Pour l'exécuter :

- Ouvrez un terminal.
- Placez-vous dans le dossier *madjan/*.
- Lancez le script install.sh : $./install.sh

2.2.2 Compilation sous Windows

Pour recompiler l'application sous le système d'exploitation Windows il vous faut installer *gnu-make* disponible à cette adresse : gnu-make. Ainsi que *qt-creator* disponible à cette adresse : qt-creator. Ensuite il vous faudra configurer *qt-creator* pour recevoir le projet en utilisant madjan.pro se trouvant le dossier src/. A noter que ce le fichier madjan.pro est partager par les différents système d'exploitation, il vous faudra sous Windows copier les dossiers de plugins directement dans le dossier de plugins présent dans le dossier de compilation.

2.3 Exécution

Après la compilation du programme un exécutable permettant de lancer l'application sera disponible en fonction du système d'exploitation que vous utilisez. A noter que pour windows il n'est pas nécessaire de compiler l'application.

2.3.1 Exécution sous Linux

Dans le dossier source de l'application Madjan vous pouvez lancer l'application soit en passant par un gestionnaire de fichier et en double cliquant sur l'exécutable nommer "Madjan" soit l'aide d'un terminale, vous placez dans le dossier source et lancez l'exécutable.

2.3.2 Exécution sous Mac

Dans le dossier source de l'application Madjan vous pouvez lancer l'application soit en passant par un gestionnaire de fichier et en double cliquant sur l'exécutable nommer "Madjan.app" soit l'aide d'un terminale, vous placez dans le dossier source et lancez l'exécutable.

2.3.3 Exécution sous Windows

Pour lancer l'application un exécutable madjan.exe est disponible dans le dossier /bin-windows/release/ de l'application.

3 Utilisation

3.1 Règles

La simulation de l'automate cellulaire suit différentes règles qui peuvent être paramétrées par un plugin :

3.1.1 Propagation

La propagation est un élément clé de l'application, elle permet de désigner la façon dont les cellules meurent, naissent etc...
Par défaut l'application propose une propagation pour une cellule qui se base sur son nombre de cellules voisines (dans notre cas 8 comme sur la figure 1).

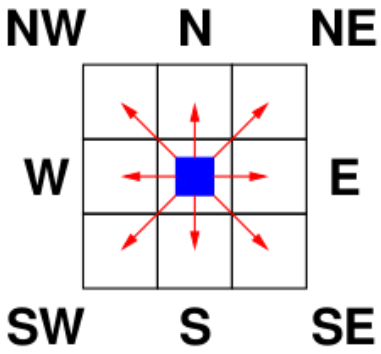

FIGURE 1 – les 8 voisins d'un pion

On peut voir sur le tableau 1 les états possibles d'un pion en fonction de ses voisins d'une génération n à une génération $n+1$

nombre de voisin génération n	**destinée génération $n+1$**
moin de 2	meurt de solitude
plus de 3	meurt de surpopulation
3	survit

TABLE 1 – tableau de la destinée d'un pion en fonction du nombre de voisin

Une dernière règle subsiste, celle où une cellule ne contient pas de pion mais possède au moins 2 voisins il y à alors la naissance d'un pion.

3.1.2 Type des pions

Les pions actifs sont tous par défaut de type "vivant", dans ce cas ils sont en cour de vie. Dans le cas où il n'y a pas de pion visible dans une cellule de la grille, un pion inactif existe tout de même mais est typé "Empty"(vide). D'autres types de pions peuvent être définis pour représenter des états intermédiaires des pions :

- naissant : pion naissant
- mourrant : pion mourant
- ...

Ces définitions restent des exemples de types de pion et peuvent être définis à l'aide de plugins. Seul le type de pion "Empty" qui est un pion inactif ne peut être modifié.

3.2 Interface

Dès lors que vous avez lancez l'exécutable de l'application *Madjan*, vous vous retrouverez face à l'interface utilisateur ou vous aurez la possiblité de configurer l'application, de lancer la simulation, etc...

3.2.1 Choix du plugin

Un bouton nommé *Plugins* vous permet d'afficher la liste des plugins disponibles et d'en sélectionner un en cliquant sur le plugin souhaité. L'application suivra les paramètres définies par le plugin choisi.

3.2.2 Choix de la taille de la grille

Deux cases *Height* (hauteur) et *Width* (largeur) vous permettrons de définir la taille initiale de votre grille. Pour appliquer les paramètres de cette grille, vous devrez cliquez sur le bouton *Apply* (appliquer) se trouvant à droite des cases.

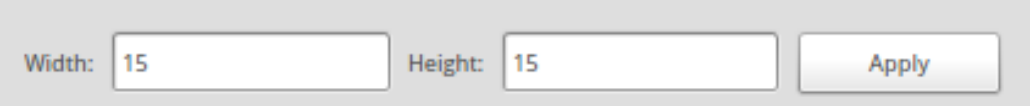

FIGURE 2 – cases de définition des tailles de la grille

Si vous souhaitez changer la taille de votre grille durant le déroulement de la simulation, vous pouvez redéfinir les champs *Width* et *Height* et appliquer cette nouvelle configuration de grille avec le bouton *Apply*.
A noter que les pions seront réinitialisés mais que la simulation sera toujours en cour tant que vous n'aurez pas appuyé sur le bouton *Pause*.

3.2.3 Placer les pions soit même

Il est possible de choisir le positionnement des différents pions avant de lancer la simulation. Pour ce faire, une liste des différents types de pion disponibles est présente à gauche de la grille. Vous pourrez alors cliquer sur un des types de pion et choisir la cellule de la grille dans laquelle vous souhaitez le mettre.

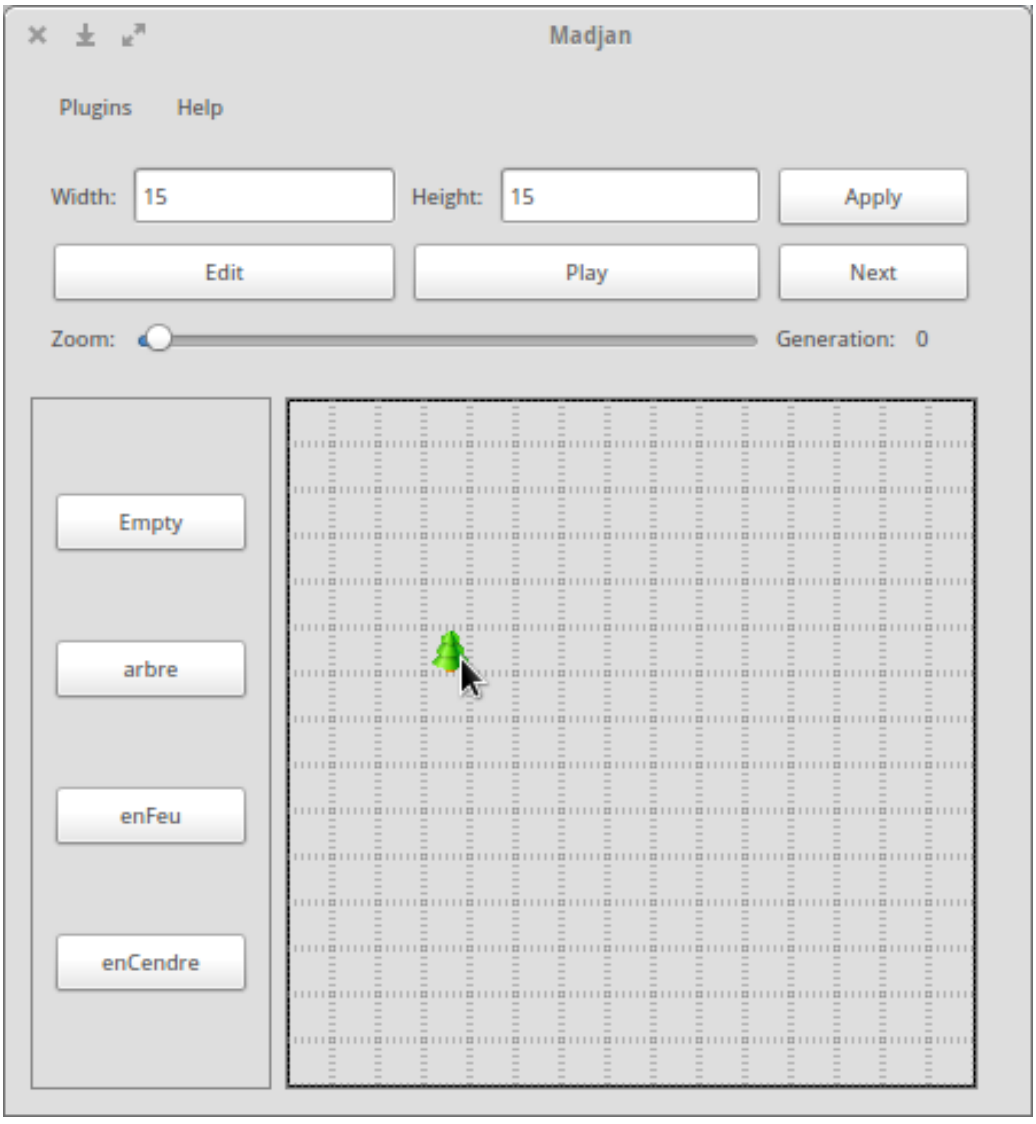

FIGURE 3 – placement d'un pion de type "arbre" dans la grille

Dans cette exemple nous ajoutons un pion de type arbre dans une cellule de la grille après avoir cliqué à gauche sur le bouton correspondant à ce type de pion.

3.2.4 Placer les pions aléatoirement

Si vous ne souhaitez pas définir l'emplacement de vos pions un par un dans la grille vous pouvez choisir une disposition générée aléatoirement dans la grille. Pour pouvoir utiliser cette fonction vous pouvez cliquer sur le bouton *edit* (edition). Ce dernier vous ouvrira une fenêtre où deux colonnes sont présentent, la colonne *Création* où vous serons demandés les paramètres de création de la grille aléatoire et la colonne *Stop Conditions* où vous pourrez indiquer si vous le souhaitez des paramètres d'arrêt de la simulation. Les paramètres *Width* et *Height* vous permettrons d'indiquer la taille de la grille souhaitée. Les autres paramètres correspondent aux pourcentages de pions attendu sur la grille dans la colonne *Creation* et aux pourcentages de pions pour lesquels la simulation s'arrête dans la colonne *Stop Conditions*.

Attention Les pourcentages sont à indiquer sans le symbole de pourcentage %. Pour les pourcentages de la colonne *Stop Conditions* les pourcentages doivent être précédé d'un signe inférieur "<" ou un signe supérieur ">" pour indiquer une conditions d'arrêt supérieur ou égale ou inférieur ou égale au pourcentage indiqué.

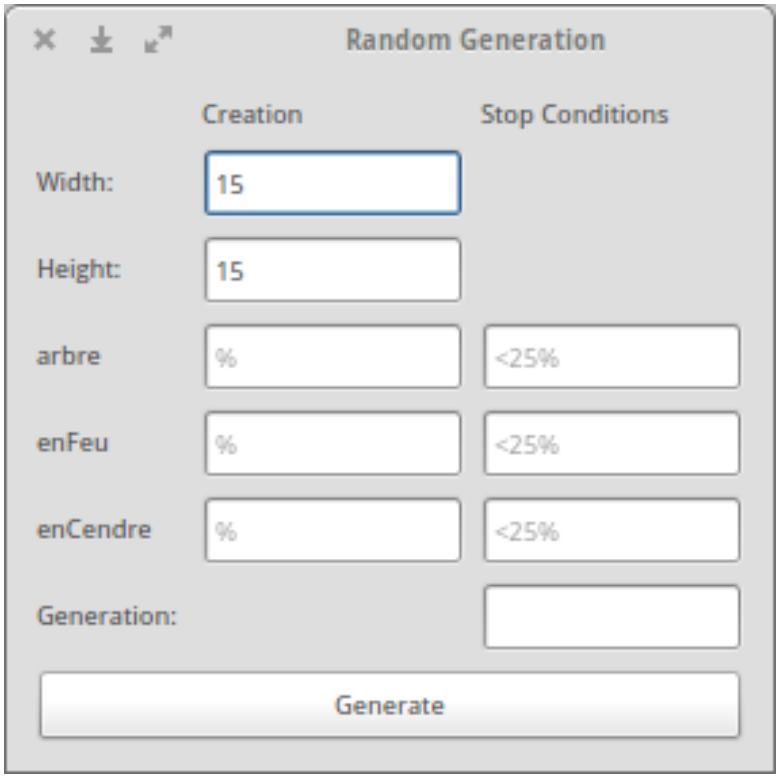

FIGURE 4 – fenêtre de la configuration de la génération aléatoire

Pour lancer la génération aléatoire de la grille il vous suffira, après avoir indiqué les différents paramètres, d'appuyer sur le bouton *Generate*.

3.2.5 Gestion de la simulation

Pour pouvoir agir sur la simulation tel que le lancement ou la mise en pause, voici la liste des commandes disponibles :

- Bouton *Play* : lance une simulation continue de toutes les étapes de génération qui sont exécutées à la suite. Ce bouton est remplacé par le bouton Pause.
- Bouton *Pause* : met en pause la simulation. Ce bouton est remplacé par le bouton Play
- Bouton *Next* : simule la génération suivante et permet d'avoir chaque itération de la simulation.
- *Zoom* : cette commande permet d'agrandir ou de réduire la grille, de gauche à droite la commande agrandie la grille et de droite à gauche réduit la grille.

A noter qu'il y a la présence d'un indicateur *Génération* qui permet de savoir à quelle génération se trouve la simulation.

4 Plugins

Les plugins vont vous permettre de définir vous même les différents comportements de l'application. Les plugins disponibles sont au format *XML* disponible dans le dossier `/plugins`.

4.1 Plugins disponibles

Par défaut l'application dispose de trois plugins disponibles dans le dossier plugin de cette dernière.

4.1.1 Plugins du jeux de la vie

Ce plugin par défaut configure l'application avec des règles de propagations présentent dans le chapitre 3.1.1 ainsi qu'un type de pion qui est de type "vivant" et qui en fonction des règles de propagations se transforme en type "Empty" ou reste de type "vivant".

4.1.2 Plugin de feu de forêt

Ce plugin a pour but de présenter la propagation d'un feu de forêt en fonction des conbustibles présents, en l'occurrence des arbres.
Les types de pions disponibles sont :

- arbre : ce pion représente le conbustible.
- enCendre : ce pion représente l'état en cendre.
- enFeu : ce pion représente l'état en feu d'un arbre.

Ces pions sont représentés par des images :

- arbre : image d'un arbre vert.
- enCendre : image d'un arbre en cendre.
- enFeu : image d'un arbre en feu.

Les règles de propagations sont simples, si un pion de type "arbre" a un voisin de type "enFeu" à la prochaine génération il deviendra un pion de type "enFeu". A la génération suivante il deviendra un pion de type "enCendre".

4.1.3 Plugin de "super" feu de forêt

Ce plugin a pour but de présenter la propagation d'un feu de forêt en fonction des conbustibles présent, en l'occurrence des arbres. En comparaison avec la version simple un type de pion intermédiaire a été rajouté.
Les types de pions disponibles sont :

- Arbre : ce pion représente le conbustible
- Arbre en cendre : ce pion représente l'état en cendre.
- Arbre en feu : ce pion représente l'état en feu d'un arbre .
- Arbre en Cendre : ce pion représente l'état en départ de feu d'un arbre.

Ces pions sont représentés par des images :

- arbre : image d'un arbre vert.
- enCendre : image d'un arbre en cendre.
- enFeu : image d'un arbre en feu.
- enPetitFeu : image d'un arbre en départ de feu.

Les règles de propagations sont les suivantes :

- si un arbre a plus de deux voisins de type arbre en départ de feu il devient un abre en départ de feu.
- si un arbre a plus de quatre voisins de type arbre en départ de feu il devient un arbre en feu.
- si un arbre a un ou plus de un voisin de type en feu il devient en feu.
- un arbre de type en départ de feu deviendra de type en feu.
- un arbre de type en feu deviendra de type en cendre.

4.2 Charger son plugins

Si vous souhaitez crée votre propre plugin pour définir vos propres caractéristiques des jetons (images, comportements, etc...), vous pouvez ajouté un plugin à la racine du projet dans le dossier `plugins/` qui se trouve au même niveau que le dossier `src/`. Dans le dossier `plugins/` vous pourrez définir n'importe quel nom de dossier qui contiendra que le fichier `plugin.xml` et si vous le souhaitez le dossier source d'images pour les pions.

4.3 Règles de création d'un plugin

Pour crée un plugin différentes balises peuvent être intégrées dans un fichier nommé `plugin.xml` :

4.3.1 Balises recommandées

Certaines balises sont recommandées pour respecter la structure d'un plugin :

1. **`<plugin>...</plugin>`** : la balise racine du document xml où toutes règles seront indiqués dedans.
2. **`<name>`**`un nom unique`**`</name>`** : le nom du plugin apparaissant dans l'interface utilisateur (obligatoire).
3. **`<description>`**`une description`**`</description>`** : description du plugin(optionelle)
4. **`<version>`**`la version`**`</version>`** : version du plugin (optionelle)
5. **`<cellBackground>`**`fondDuneCellule.format`**`</cellBackground>`** : le nom du fichier image de fond d'une cellule

4.3.2 Les pions

Vous pouvez définir les règles de vos pions à l'aide des balises suivantes :

- **`<pawns>...</pawns>`** : balise où seront présents le ou les pions et la ou leurs propriétés :
  - **`<pawn>...</pawn>`** : balise où seront présentent les propriétés du pions :
    - **`<id>`**`un id unique`**`</id>`** : id du pion qui doit être unique
    - **`<name>`**`un nom`**`</name>`** : le nom du pion
    - **`<icon>`**`image.format`**`</icon>`** : l'icon au format d'une image du pion. (attention à ne pas mettre de /devant le nom du répertoire ou de l'image)

Plusieurs balises *pawn*(pion) peuvent être définies avec leurs propriétés seulement si les identifiants sont différents sinon l'application ne prendra pas en compte les pions qui partage le même identifiant.

4.3.3 Les Règles de propagations associés au pions

Les règles de propagations peuvent être écrite en lien directe avec les pions définis dans le fichier `plugin.xml`. Pour en écrire de nouvelles vous pouvez utiliser les balises suivantes :

- **`<rules>...</rules>`** : balise qui contient la ou les règles associés aux pions :

  - **`<rule`** `pawn="nom de l'état du pion" newPawn="nom du nouveau pion crée"`**`>`**
    `...`
    `</rule>`
    : balise permettant de définir le nouveau pion (*newPawn*) qui sera généré à la place du pion (*pawn*) en fonction des règles appliquées :

    - **`<pawnNumberIsEqual`** `pawn="nom du pion"`**`>`**
      `nombre de pion voisin`
      `</pawnNumberIsEqual>` :
      balise qui définie une conditions permetant à un pion d'être transformé, à la génération suivante, si le nombre de type de pion *pawn* est **égale** au *nombre de pion* voisin indiqué.
    - **`<pawnNumberIsLower`** `pawn="nom du pion"`**`>`**
      `nombre de pion voisin`
      `</pawnNumberIsLower>` :
      balise qui définie une conditions permetant à un pion d'être transformé, à la génération suivante, si le nombre de type de pion *pawn* est **inférieur** au *nombre de pion* voisin indiqué.
    - **`<pawnNumberIsGreater`** `pawn="nom du pion"`**`>`**
      `nombre de pion voisin`
      `</pawnNumberIsGreater>` :
      balise qui définie une conditions permetant à un pion d'être transformé, à la génération suivante, si le nombre de type de pion *pawn* est **supérieur** au *nombre de pion* voisin indiqué.

Comme pour les pions il est possible de définir plusieurs règles de propagations avec leurs conditions d'exécutions.
Dans le cas ou plusieurs règles sont appliqués pour le même pions seul celle ou le pion satifsait les conditions sera exécutée (utilisation du booléen OU). Enfin pour qu'une règle soit appliquée, le pion doit satisfaire toutes les conditions de cette dernière.

Attention à ne pas définir des conditions contradictoires tels qu'un nombre de voisin supérieur et inférieur, dans un tel cas la règle contenant ces conditions ne sera pas appliquée.

En outre si deux règles définies, pour un pion, possèdent les mêmes conditions mais pas le même nombre de voisin pour satisfaire la ou les conditions alors la dernière règle sera exécutée.

Table des matières