Travail académique de substitution au stage

Rapport de fin Juillet

Réalisé par Pablo TOMAS
Master 2 Informatique parcours Génie Logiciel

1. Rappel du sujet

Le but de ce projet est de réaliser un Générateur Procédural de Textures Pixelisées en Deux Dimensions. Celui-ci est fortement inspiré du logiciel *Crypixels* (disponible ici: https://crypixels.com/). Le projet se démarque de ce logiciel dans l'emploi des technologies employées, dans le choix de l'interface graphique et dans les fonctionnalités implémentées.

Le langage de programmation utilisé dans ce projet est le *Rust* (disponible ici: https://www.rust-lang.org/). Ce langage est sécurisé et offre une certaine liberté dans l'implémentation d'un logiciel puisqu'il est bas niveau. Même si l'apprentissage de ce langage n'est pas le plus aisé, son compilateur est très compréhensible et les procédures de tests et de debugging sont accessibles et simplifiées.

Avant de pouvoir générer des textures, il faut cerner les besoins de l'utilisateur. Ce projet avant d'être un générateur est avant tout un éditeur permettant de générer des textures. Pour cela l'utilisateur peut les exprimer au travers de l'interface graphique. Pour rester sur le thème du pixel et conserver une ambiance rétro, j'ai choisi de développer l'interface utilisateur sur le terminal et en ligne de commande. Le langage *Rust* et ses *Crates* offrent de nombreuses possibilités pour ce type de besoin.

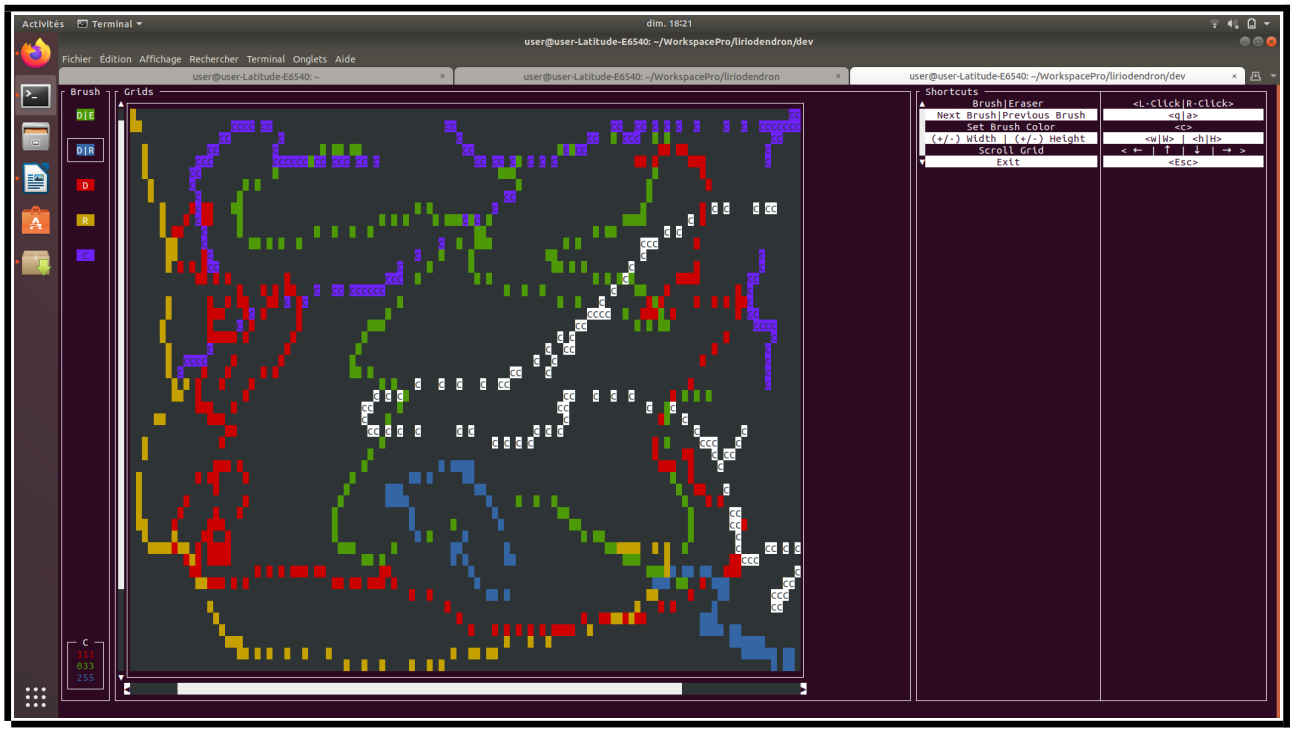
Pour permettre au projet de se lancer en ligne de commande, il est possible de le faire sans *Crates* et d'implémenter son propre parseur de lignes de commandes. Cependant le travail est fastidieux et ce n'est pas l'objectif de ce projet. Qui plus est, les *Crates* proposés par *Rust* sont très bien documentés, faciles à prendre en main, offrent de nombreuses fonctionnalités, sont stables et régulièrement mis à jour. Pour parser les lignes de commandes, le *Crate* <*clap.rs*> (disponible ici: https://github.com/clap-rs/clap) est celui que l'on retrouve le plus souvent cité. Même si <*clap.rs*> est le choix le plus évident, il me paraît plus intuitif de pouvoir encapsuler les données produites par la ligne de commande dans une structure. Le *Crate* <*structopt.rs*> (disponible ici: https://github.com/TeXitoi/structopt) permet justement de faire cela.

Pour l'interface graphique sur terminal, les *Crates* disponibles sont là aussi nombreux. Le plus cités est le *Crate* <*tui.rs*> (disponible ici: https://github.com/fdehau/tui-rs). <*tui.rs*> propose 4 différents Backend pour pouvoir travailler sur le terminal. Le seul d'entre eux supportant aussi bien les terminaux UNIX et Windows est <*crossterm.rs*> (disponible ici: https://github.com/crossterm-rs/crossterm). Mes choix se sont donc tournés vers ces technologies.

Pour ce qui est de la génération d'images et l'utilisation de technologies procédurales, les *Crates* <*image.rs*> (disponible ici:

https://github.com/image-rs/image), <*noise.rs*> (disponible ici: https://github.com/razaekel/noise-rs) et <*rand.rs*> (disponible ici: https://github.com/rust-random/rand) ne semblent pas avoir de véritables concurrents permettant de remettre en cause leur utilisation dans ce projet.

Voici ce à quoi ressemble actuellement l'interface utilisateur sur la vue d'édition:



Le projet est disponible ici:
https://bitbucket.org/trapped_in_a_while_loop_team/liriodendron/src/master/

2. État de l’art

a. Logiciels de génération procédurale de textures pixellisées

Le logiciel *Crypixels* dont ce projet est inspiré possèdent de nombreuses fonctionnalités permettant à l’utilisateur d’exprimer au mieux ses besoins. En plus de posséder une interface graphique claire, son utilisation est très simple et bien documentée. L’utilisateur peut:

- Créer une grille à éditer,
- Utiliser une brosse pour remplir les cellules de la grilles,
- Remplir les cellules de la grille avec différents contenus,
- Choisir la taille de la brosse,
- Accéder à un barre d’outils comprenant un zoom, un bouton pour annuler ou refaire une action, un bouton pour changer de vue et un bouton pour générer des textures,
- Exporter les textures générées en les copiant ou en la sauvant,
- Sauver et Charger une grille,
- Nettoyer une grille,
- Changer la taille de sa grille,
- Modifier le contenu de le grille en effectuant des rotations,
- Modifier la génération de la texture en appliquant un effet miroir verticalement et/ou horizontalement,
- Désactiver l‘effet de miroir sur la ligne ou la colonne centrale lors de la génération,
- Choisir le nombre de textures à générer,
- Choisir combien de pixels représente une cellule de la grille,
- Choisir la couleur de fond de la texture,
- Un mode 1-bit avec un choix pour les deux couleurs principales,
- Choisir la couleur des bordures de sa texture,
- Choisir la variation de couleur de sa texture,
- Ajuster la quantité de bruit sur les textures générées,
- Ajuster la saturation sur les textures générées,
- Ajuster la luminosité des textures générées,
- Ajuster la direction du gradient quand la variation de couleur est utilisée,
- Forcer l’utilisation de certaines couleurs dans ses textures plutôt que d’utiliser des couleurs aléatoires,
- Choisir le nom des textures générées,
- Sauver toutes les textures générées dans un dossier spécifié,
- Utiliser des raccourcis clavier pour utiliser certaines fonctionnalités.

Bien que ce logiciel propose un grand nombre de fonctionnalité pour permettre à l’utilisateur de paramétrer au mieux la génération de ses textures, il manque des fonctionnalités pour modifier la génération des pixels. En effet, on remarque que la plupart des fonctionnalités de ce logiciel ne concernent que les couleurs de la texture.

La seule fonctionnalité qui s'applique aux pixels est celle permettant d'ajuster la quantité de bruit appliquée sur les textures générées. Il existe de nombreux moyens pour modifier une fonction de bruit et il manque des fonctionnalités permettant à l'utilisateur de personnaliser sa fonction de bruit.

En plus de cela, ce logiciel ne permet pas d'avoir accès à plusieurs grilles sur l'espace d'édition. De cette manière il aurait été possible pour l'utilisateur de superposer plusieurs grilles sur une seule texture et avoir ainsi des textures plus complexes.

Ensuite il ne semble pas possible de pouvoir choisir le format de sortie des textures générées. Il n'est pas fait mention d'une telle possibilité dans la documentation ou sur le site marchand du logiciel.

Enfin, le logiciel est seulement accessible aux utilisateurs possédant un environnement Windows ou Mac-OS.

Il est possible de trouver d'autre projets possédant également certaines fonctionnalités déjà listées au dessus. Ils sont directement intégrés sur navigateur ou sont disponibles sur *github*. Malheureusement, les procédés de génération et les algorithmes utilisés ou décrits ne différent en rien de ceux utilisés par *Crypixels*. Certains utilisent des automates cellulaires pour générer les textures mais étant donné l'infinie variété d'automates cellulaires qu'il existe, il est difficile de pouvoir en parler si les auteurs ne donnent pas davantage de détails sur l'implémentation de l'algorithme utilisé. *Crypixels* semble être le logiciel le plus complet et le plus documenté. En listant directement ses qualités et ses défauts, il n'est pas nécessaire de lister les autres logiciels trouvés et leur fonctionnalités. Cela rendrait le rapport redondant puisque ceux-ci ne proposent rien d'autre que ce que *Crypixels* proposent déjà.

b. Algorithmes de génération procédurale de textures pixellisées

Le premier algorithme est celui décrit ici: http://web.archive.org/web/20080228054410/http://www.davebollinger.com/works/pixelspaceships/. Il s'agit en premier lieu de remplir une grille ou un tableau d'entier en donnant à chaque cellule une valeur en fonction de ce que cela représentera dans la texture que le programme sera amené à générer. Pour certaines valeur spécifiques dans la grille, la cellule peut prendre deux valeurs.. C'est dans ce cas là que l'on va appliquer un bruit ou une fonction utilisant le hasard pour déterminer laquelle de ces deux valeurs sera sélectionnée pour être utilisée dans la génération de la texture. Une fois que toutes les valeurs de la grille sont déterminées, celle ci représente la texture à générer. Il ne reste plus qu'à déterminer une couleur en fonction des valeurs dans la grille et à l'appliquer pour obtenir une image. Voilà ce que cela donne:

```
1   INPUT:  Grid G
2   LET Image I = empty;
3   LET ColorRGB rgb = randomRGBColor();
4   FOR each row r of the grid G:
5       FOR each column c of the grid G:
6           SWITCH G[r, c]:
7               CASE body-border-pixel:
8                   G[r, c] = random(border-pixel, body-pixel);
9                   colorize_image(I, rgb, G[r, c]);
10              END CASE
11              CASE body-empty-pixel:
12                  G[r, c] = random(empty-pixel, body-pixel);
13                  IF G[r, c] == empty-pixel:
14                      FOR each neighbor n of the cell G[r, c]
15                          IF n == body-pixel
16                              G[r, c] = border-pixel;
17                          ENDIF
18                      ENFOR
19                  ENDIF
20                  colorize_image(I, rgb, G[r, c]);
21              END CASE
22              CASE empty-pixel:
23                  FOR each neighbor n of the cell G[r, c]
24                      IF n == body-pixel
25                          G[r, c] = border-pixel;
26                      ENDIF
27                  ENFOR
28                  colorize_image(I, rgb, G[r, c]);
29              END CASE
30              DEFAULT:
31                  colorize_image(I, rgb, G[r, c]);
32              END DEFAULT
33          END SWITCH
```

```
34          END FOR
35      END FOR
36      OUTPUT: Image I
```

N'étant pas totalement explicite voici la fonction *colorize_image*() dans le cas où une cellule de la grille représente un pixel de l'image. Si une cellule représentait un plus grand nombre de pixel, cela n'augmenterait pas significativement la complexité de cet algorithme. La variable N dont la valeur n'est pas indiquée dans la fonction *colorize_image*() est une constante supérieure ou égale à 1 pour exprimer qu'un « border-pixel » est plus sombre qu'un « body-pixel »:

```
1   INPUT: (Image I, CorlorRGB rgb, Cell C at row r and column c in a grid G)
2   FUNCTION colorize_image:
3           SWITCH C:
4                   CASE body-pixel:
5                           I[r, c] = ColorRGBA(rgb * noise2D(r, c), 255);
6                   END CASE
7                   CASE border-pixel:
8                           I[r, c] = ColorRGBA((rgb/N) * noise2D(r, c), 255);
9                   END CASE
10                  CASE empty-pixel:
11                          I[r, c] = ColorRGBA(0, 0, 0, 0);
12                  END CASE
13                  DEFAULT:
14                          Do nothing;
15                  END DEFAULT
16          END SWITCH
17  END FUNCTION
18  OUTPUT: Image I modified
```

Dans l'algorithme 1, lorsqu'un « empty-pixel » se trouve à côté d'un « body-pixel » celui-ci prend la valeur d'un « border-pixel ». C'est pour cette raison qu'il faut vérifier les valeurs de tous les pixels dans le voisinage des « empty-pixels ». On remarque la présence de deux boucles pour itérer au travers chaque cellule de la grille et pouvoir ainsi remplir l'image. La première boucle itérant au travers les lignes de la grille et la seconde boucle itérant au travers les colonnes de la grille, on peut en déduire que la complexité de cet algorithme est de l'ordre de $w * h$ (où w est le nombre de lignes dans la grille et h le nombre de colonnes dans la grille).

Le second algorithme est décrit ici: http://davideyork.com/gengam-2016. En entrée l'algorithme prend un ensemble de grilles. Chaque grille de cet ensemble décrit une partie différente de la texture à générer. Il est donc important que soit les parties colorées par les différentes grilles ne se chevauchent pas, soit il faut décrire une ordre de priorité entre les différentes grilles pour qu'au cas où deux grilles de génération colorent une même partie de la texture, le programme puisse choisir quelle grille s'occupent prioritairement de cette partie de la texture. Il est important de

préciser que toutes les grilles de cet ensemble doivent avoir exactement les mêmes dimensions. Pour chaque grille il faut ensuite choisir une couleur ou un ensemble de couleur. L'algorithme ressort une texture. Voici ce que cela donne si l'on utilise le premier algorithme en complément pour déterminer les cellules remplient par l'utilisateur sur les différentes grilles de l'interface graphique:

```
1   INPUT:  Set S of grids sorted by ascendant priority order
2   LET Image I = empty;
3   FOR each grid G of the set S:
4           LET ColorRGB rgb = randomRGBColor();
5           FOR each row r of the grid G:
6                   FOR each column c of the grid G:
7                           SWITCH G[r, c]:
8                                   CASE body-border-pixel:
9                                           G[r, c] = random(border-pixel, body-pixel);
10                                          colorize_image(I, rgb, G[r, c]);
11                                  END CASE
12                                  CASE body-empty-pixel:
13                                          G[r, c] = random(empty-pixel, body-pixel);
14                                          IF G[r, c] == empty-pixel:
15                                                  FOR each neighbor n of the cell G[r, c]
16                                                          IF n == body-pixel
17                                                                  G[r, c] = border-pixel;
18                                                          ENDIF
19                                                  ENFOR
20                                          ENDIF
21                                          colorize_image(I, rgb, G[r, c]);
22                                  END CASE
23                                  CASE empty-pixel:
24                                          FOR each neighbor n of the cell G[r, c]
25                                                  IF n == body-pixel
26                                                          G[r, c] = border-pixel;
27                                                  ENDIF
28                                          ENFOR
29                                          colorize_image(I, rgb, G[r, c]);
30                                  END CASE
31                                  DEFAULT:
32                                          colorize_image(I, rgb, G[r, c]);
33                                  END DEFAULT
34                          END SWITCH
35                  END FOR
36          END FOR
37  END FOR
38  OUTPUT: Image I
```

Cet algorithme, en plus des deux premières boucles du premier algorithme, comprend une troisième boucle. Sachant que toutes les grilles sont de mêmes dimensions, on peut donc en déduire que la complexité de cet algorithme est de cet

ordre: *w* * *h* * *n* (où *w* est le nombre de lignes dans la grille, *h* le nombre de colonnes dans la grille et *n* le nombre de grille dans l'ensemble).

En appliquant cet algorithme avec un ensemble de deux grilles où l'une représente un feuillage et l'autre représente un tronc, on obtient les résultats suivants:

c. Évaluation du générateur

Dans l'article de recherche disponible ici: http://pcgbook.com/wp-content/uploads/chapter12.pdf, les auteurs décrivent un moyen d'évaluer la qualité d'un générateur en fonction du contenu généré. Pour cela, l'article décrit deux approches qui peuvent être appliquées ensemble ou séparément. L'article propose d'appliquer leur méthode sur un générateur en lien avec les jeux vidéos mais leur méthode n'est pas destinée exclusivement à ce domaine et s'applique à tout type de générateur.

La problématique de l'article tient dans le fait qu'il n'est pas possible d'évaluer visuellement le contenu généré par un générateur comme il serait tentant de le faire. En effet, un générateur peut en principe générer des millions, des milliards ou des chiffres bien plus grands de possibilités. En principe, il n'est pas absolument pas possible de visualiser toutes ces possibilités. Donc le développeur se contentera de visualiser seulement un tout petit nombre de ces possibilités pour se satisfaire de ce résultat et les généraliser à l'ensemble du contenu pouvant être généré par son outil. Malheureusement c'est totalement insuffisant, peu rigoureux et très subjectif pour se faire une véritable idée de la valeur qualitative du générateur que l'on est en train d'implémenter.

La première méthode proposée par les auteurs est une évaluation dite « top-down ». Cette approche s'applique en définissant en premier lieu des « expressivity metrics ». Ces « expressivity metrics » sont définies par le concepteur du générateur. Elles se doivent de définir au mieux les propriétés du générateur. Si les « expressivity metrics » sont trop proches des paramètres d'entrées la méthodes d'évaluation ne peut que apporter que des résultats confirmatifs. Si les « expressivity metrics » sont pensées pour évaluer les sorties du générateur, cela reviendrait à vouloir démontrer que l'outil génère ce qu'on lui demande de générer sans s'attarder sur les générations particulières ou inattendues. Pour cela le choix des « expressivity metrics » doivent donc être le plus éloignées possible des paramètres d'entrée du générateur.

Ces « expressivity metrics » doivent permettre d'évaluer l'« expressive range » du générateur en définissant des axes sur un graphique. L'« expressive range » représente l'ensemble des possibilités (en terme de généralités tout comme en terme de particularités) que l'outil est capable de générer. L'« expressive range » du générateur en étant placé sur un graphique à plusieurs dimensions (où chaque dimension représente une « expressivity metric ») peut être visualisé. L'objectif est d'en avoir une idée globale selon les critères définis en amont.

Selon le choix de ces critères, la forme de l'« expressive range » varie totalement. Il est donc important de s'attarder suffisamment sur les « expressivity metrics » pour les définir correctement. Après cela, pour visualiser l'« expressive range », il faut produire un grand nombre de génération avec l'outil. L'échantillon produit par le générateur se doit d'être représentatif de ce que l'outil peut générer. Dans le cas où le générateur peut produire une infinité de possibilité (ou quelque

chose qui peut s'en approcher humainement), l'auteur propose de créer l'échantillon représentatif itérativement. En visualisant l'échantillon à chaque itération sur le graphique de l' « expressive range », il faudrait s'arrêter lorsque l'échantillon se ressemble d'une itération sur l'autre.

Une fois l'échantillon représentatif produit, le graphique de l' « expressive range », si les « expressivity metrics » sont correctement définies, peut révéler des défauts de conception. En réalisant plusieurs graphiques où à chacun d'eux il est attribué une configuration différente de paramètres d'entrées, il est possible de mesurer le niveau de contrôle des entrées sur les sorties du générateur. Ainsi, en observant la « controllability » du générateur le concepteur peut se faire une idée de l'importance qu'un changement de propriétés dans le système peut avoir sur son expressivité (et donc sur les générations produites).

La seconde méthode décrite par l'article est dite « bottom-up ». Pour mesurer la pertinence du contenu généré par l'outil, l'approche la plus évidente serait de s'enquérir directement auprès des utilisateurs de leur expérience avec l'outil. L'article recommande d'utiliser des questionnaires permettant aux utilisateurs de noter leur expérience sur différents critères et aspects du générateur. Les différentes questions peuvent intégrer des éléments relatifs au contexte propre à l'utilisateur et à ses besoins. En effet, le contexte dans lequel l'utilisateur baigne peut directement influencer son expérience avec l'outil.

Une autre manière de mesurer l'expérience utilisateur est l'utilisation d'intelligences artificielles qui pourrait évaluer directement le contenu généré par l'outil pendant ou après que l'utilisateur utilise l'outil sans forcement demander un avis à l'utilisateur. Plus l'utilisateur génère et utilise le contenu produit par l'outil, plus l'intelligence artificielle peut raisonnablement penser que soit 1) l'utilisateur est satisfait par l'outil s'il ne donne pas d'appréciation sur l'outil 2) son avis a plus de valeur que les autres si l'utilisateur partage son appréciation de l'outil (puisqu'il a testé un plus grand contenu que les autres utilisateurs). Certains aspects de l'outil peuvent être mesurés par l'utilisateur sans explicitement lui demander son avis sur cet aspect. L'intelligence artificielle peut raisonnablement penser qu'une certaine fonctionnalité est qualitative si l'utilisateur passe la majorité de son temps à l'employer lors de son utilisation de l'outil.

L'article conclut en affirmant que l'utilisation des deux méthodes simultanément plutôt que séparément permet d'évaluer et de comprendre le générateur implémenté plus globalement dans sa conception et dans son utilisation.