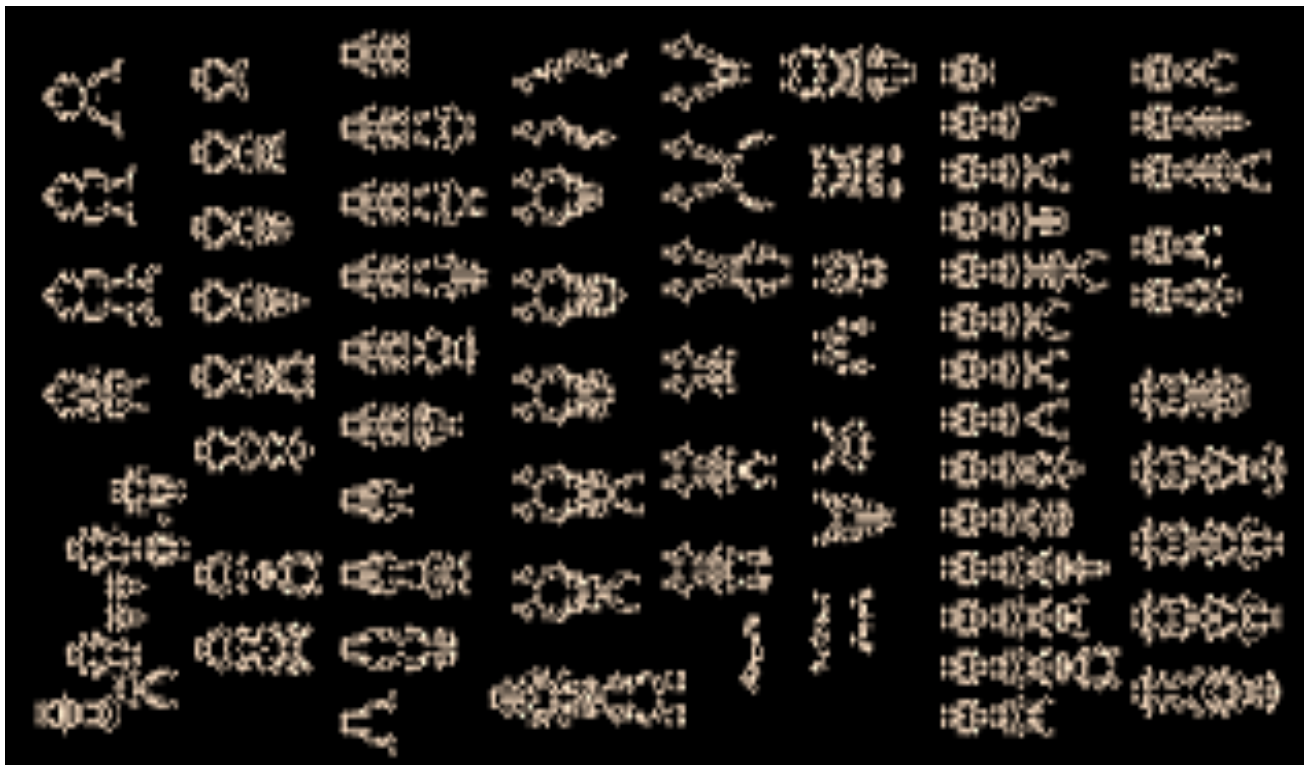


COMPTE RENDU ET RAPPORT TP4b

## IMPLÉMENTAION DU JEU DE LA VIE



## Table des matières :

### Introduction

I- Présentation de l'interface et des fonctions principales

II- Analyse des différentes évolutions des cellules

### Synthèse

### Lexique



## Introduction

Il nous a été confiés dans ce TP la réalisation d'une implémentation du **jeu de la vie** en langage de programmation C. Dans un premier temps nous avons implémentés le jeu sur une grille  $10 \times 10$  (10 lignes et 10 colonnes), sur une grille normale ensuite sur une grille *torique*<sub>1</sub>. L'état initiale de cette grille a été donné, elle simulera un *glider*<sub>2</sub> (planeur) et nous devons analyser son comportement. Dans une seconde partie nous devons réaliser des grilles plus grandes, cette fois ci avec une initialisation aléatoire et omettre des observations. Ce projet contient un fichier C nommé `tp4b.c`, un `Makefile` pour faciliter la commande de compilation si désiré par l'utilisateur, et un rapport nommé **rapport.pdf**. Il y'a un lexique disponible en fin de page donnant une définition pour tous les mots en *italique*<sub>x</sub> contenant un indice dans ce rapport.

### I- Présentation de l'interface et des fonctions principales

```
Bienvenue au jeu de la vie!  
Appuyer sur 1 pour lancer une partie aléatoire en plein écran  
Appuyer sur 2 pour lancer une partie aléatoire à échelle réduite avec des dimensions précises  
Appuyer sur 3 pour lancer une partie aléatoire à échelle réduite avec des dimensions précises avec un temps pour la cadence des évolutions de nos cellules  
Appuyer sur 4 pour lancer une partie avec le glider (planeur) sur une grille 10*10 comme celle de la question 1 de l'énoncé sur une grille NORMALE  
Appuyer sur 5 pour lancer une partie avec le glider (planeur) sur une grille 10*10 comme celle de la question 1 de l'énoncé sur une grille TORIQUE  
Appuyer sur 6 pour lancer une partie non aléatoire en plein écran avec un canon de gliders (planeurs)  
  
NB : Plus petites seront les dimensions de la grille moins précises seront les séquences de résultats  
Plus petites seront les dimensions de la grille plus rapide et plus ennuyant sera le jeu  
Le programme étant très rapide il faudra parfois appuyer rapidement plusieurs fois ctrl-c pour l'arrêter  
En tapant des nombres pour une grille plus petite avec le choix 2, veiller à ne rentrer aucun autre caractère que des chiffres  
Avec le choix 2, si vous choisissez des dimensions pour la grille plus grande que les dimensions de votre écran, l'affichage risque d'être corrompu !  
(conseils : ne pas dépasser 45 lignes ou 75 colonnes pour un écran de 13,3 pouces)  
  
Pour quitter le programme à n'importe quel moment, appuyer sur (contrôle et c simultanément) ctrl-c
```

Il y'a plusieurs choix possibles pour assurer à utilisateur une fluidité maximale. Tous les choix proposés permettent de répondre aux questions du TP, bien encore plus. En plus de ça il y'a des indications donnant toutes les informations utiles pour le déroulement du jeu.

```
2  
Saisissez le nombre de lignes  
  
20  
Saisissez maintenant le nombre de colonnes  
  
30
```

Avec le choix 2 qui consiste à choisir nous même les dimensions de notre grille, nous voyons que toutes les informations nécessaires sont demandées à l'utilisateur pour le lancement, tout en tenant compte du conseil disant que plus les dimensions seront petites, moins intéressant et plus rapide sera le jeu.

```
3
Saisissez le nombre de lignes

20
Saisissez maintenant le nombre de colonnes

30
Saisissez un temps en microsecondes la cadence des évolutions (de préférence supérieur ou égale à 15000 pour faciliter le ctrl-c)

90000
```

Avec le choix 3, nous avons exactement les mêmes demandes, avec en prime l'option pour préciser le nombre de secondes pour l'évolution. Ce choix renferme une option vraiment minutieuse car on regarde nos cellules évoluer tout calmement, avec une belle facilité de prise pour le ctrl-c.

```
int voisinage(int grille[][nbColonnes], int indice_i, int indice_j,
              int ligne, int colonne )
{
    int nbVoisins = 0;
    int i;
    int j;

    // Boucle nous permettant de parcourir les lignes alentours de la cellule actuelle
    for(i = indice_i - 1 ; i <= indice_i + 1 ; i++)
    {
        // On vérifie que l'on ne dépasse pas les lignes voisines de la cellule actuelle
        if( i >= 0 && i < ligne)
        {
            // Boucle nous permettant de parcourir les colonnes alentours de la cellule actuelle
            for( j = indice_j - 1 ; j <= indice_j + 1 ; j++)
            {
                // On vérifie que nous ne dépassons pas les colonnes voisines de la cellule actuelle
                if(j >= 0 && j < colonne)
                {
                    /* Ce test nous permet de vérifier que l'on passe exactement sur toutes les cellules */
                    /* voisines mais que l'on ne passe pas la cellule actuelle dans la vérification */
                    if(i != indice_i || j != indice_j)
                    {
                        if(grille[i][j] == vivant)
                        {
                            nbVoisins++;
                        }
                    }
                }
            }
        }
    }

    return nbVoisins;
}
```

La fonction `voisinage` nous permet de parcourir notre grille tout en comptant le nombre de voisins de la cellule actuelle sur laquelle on se trouve. Elle renvoie le nombre de voisins de la cellule. Tout un nombre de



test minutieux sont mis en place afin de s'assurer de ne pas déborder du voisinage de notre cellule. Grâce aux nombres de lignes et colonnes total de la grille en ajoutant l'indice de ligne et de colonne de la cellule sur laquelle on se trouve l'on peut facilement compter son nombre de voisins.

```
int animer_ou_tuer(int grille[][nbColonnes], int nbVoisins, int indice_i, int indice_j)
{
    /* Vérifie si la cellule vivante à 2 ou 3 voisines et la maintient en vie */
    /* Sinon elle meurt dans le cas contraire */
    if(grille[indice_i][indice_j] == vivant)
    {
        if(nbVoisins == 2 || nbVoisins == 3)
        {
            return vivant;
        }
        else
        {
            return mort;
        }
    }
    // Vérifie si la cellule morte à exactement 3 voisines pour la rendre vivante, sinon elle reste morte
    else
    {
        if(nbVoisins == 3)
        {
            return vivant;
        }
        else
        {
            return mort;
        }
    }
}
```

Avec le nombre de voisins que nous renvoie la fonction `voisinage`, nous avons cette fonction très facile nommée `animer_ou_tuer` qui contient tout simplement les différentes règles du jeu de la vie et se permet de donner la vie ou de tuer une cellule selon son nombre de voisins tout comme l'indique son nom.

```

void deplacement(int grille[][nbColonnes], int transition[][nbColonnes], int ligne, int colonne)
{
    int i;
    int j;

    for(i = 0 ; i < ligne ; i++)
    {
        for(j = 0 ; j < colonne ; j++)
        {
            // Compte le nombre de voisins de transition[i][j] et détermine l'état suivant de la cellule
            transition[i][j] = animer_ou_tuer(grille, voisinage(grille, i, j, ligne, colonne), i, j);
        }
    }

    for(i = 0 ; i < ligne ; i++)
    {
        for (j = 0 ; j < colonne ; j++)
        {
            // Affectation de l'état t + 1 de nos cellules dans la grille
            grille[i][j] = transition[i][j];
        }
    }
}

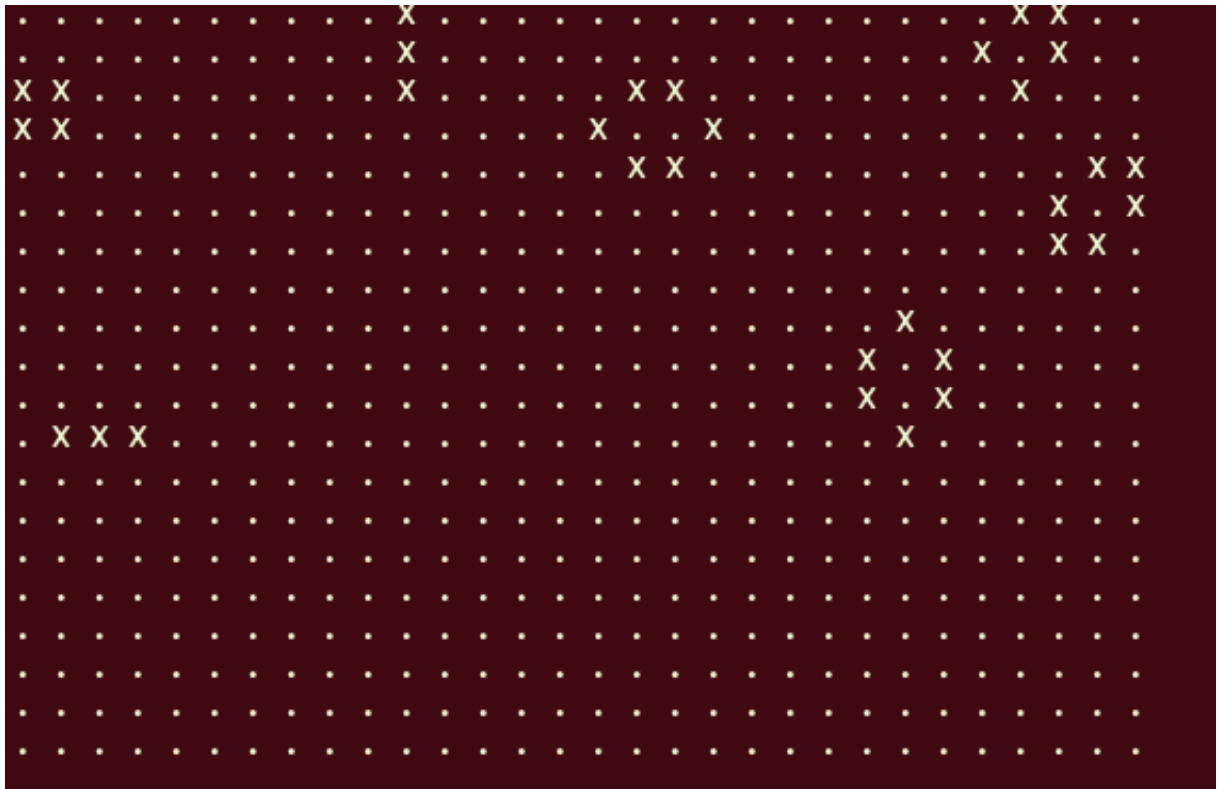
```

Pour éviter de faire des fonctions absurdement longues, nous avons cette dernière fonction `déplacement` qui renferme la fonction `voisinage` et `animer_ou_tuer`. Elle a pour effet de bord la mise en place de l'état suivant de nos cellules à l'aide d'une matrice `transition`. Toutes les explications des fonctions sont disponibles dans les prologues au niveau du fichier code. C'est l'assemblage de ces trois fonctions `déplacement`, `animer_ou_tuer` et `voisinage` qui constitue le jeu. Ensuite nous avons les fonctions mineures comme l'affichage, l'initialisation et d'autres encore, dont toutes sont dotées de leur prologue respectif.

Pour une grille *torique*<sub>1</sub> nous avons exactement les mêmes configurations, la seule différence est que nous sommes dans un monde *torique*<sub>1</sub> au lieu d'un normal.

## II- Analyse des différentes évolutions des cellules





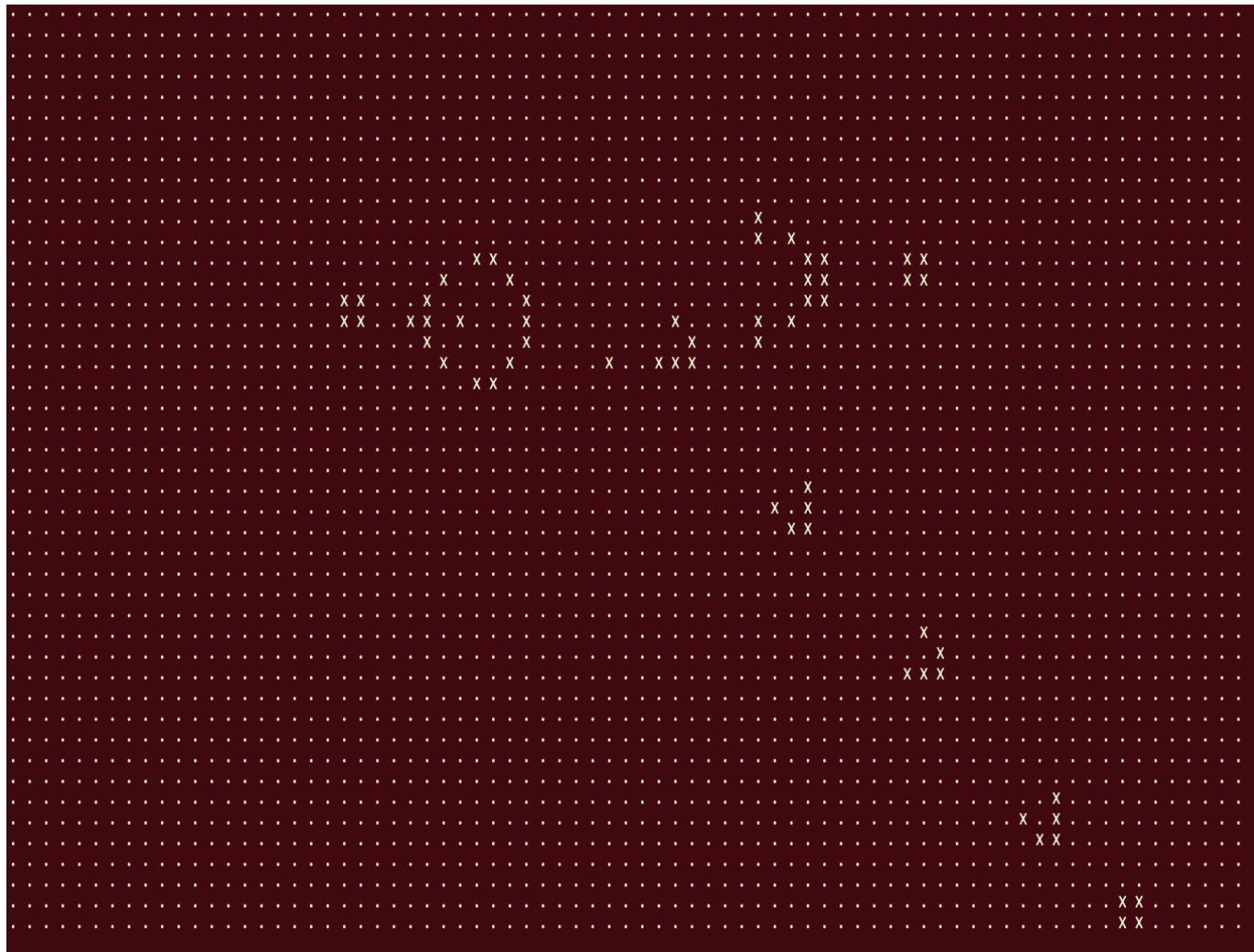
Jeu sur une grille 20\*30

Nous reprenons notre exemple du haut où l'utilisateur a tapé 20 lignes et 30 colonnes pour une grille à échelle réduite.

Ce rapport étant en format PDF qui est un format qui ne digère pas bien les  $GIF_3$  animés, sinon on aurait pu placer l'évolution de cette grille 20\*30. Dans ce cas nous n'avons même pas besoin d'avoir l'évolution des cellules pour voir que le jeu a été rapide et que le jeu se termine sur un état stable définitif. **Les cellules vivantes étant représentés par 'X' et les mortes par '.'** .

Nous verrons tard qu'il existe également des *états stables évolutifs*<sub>9</sub>.





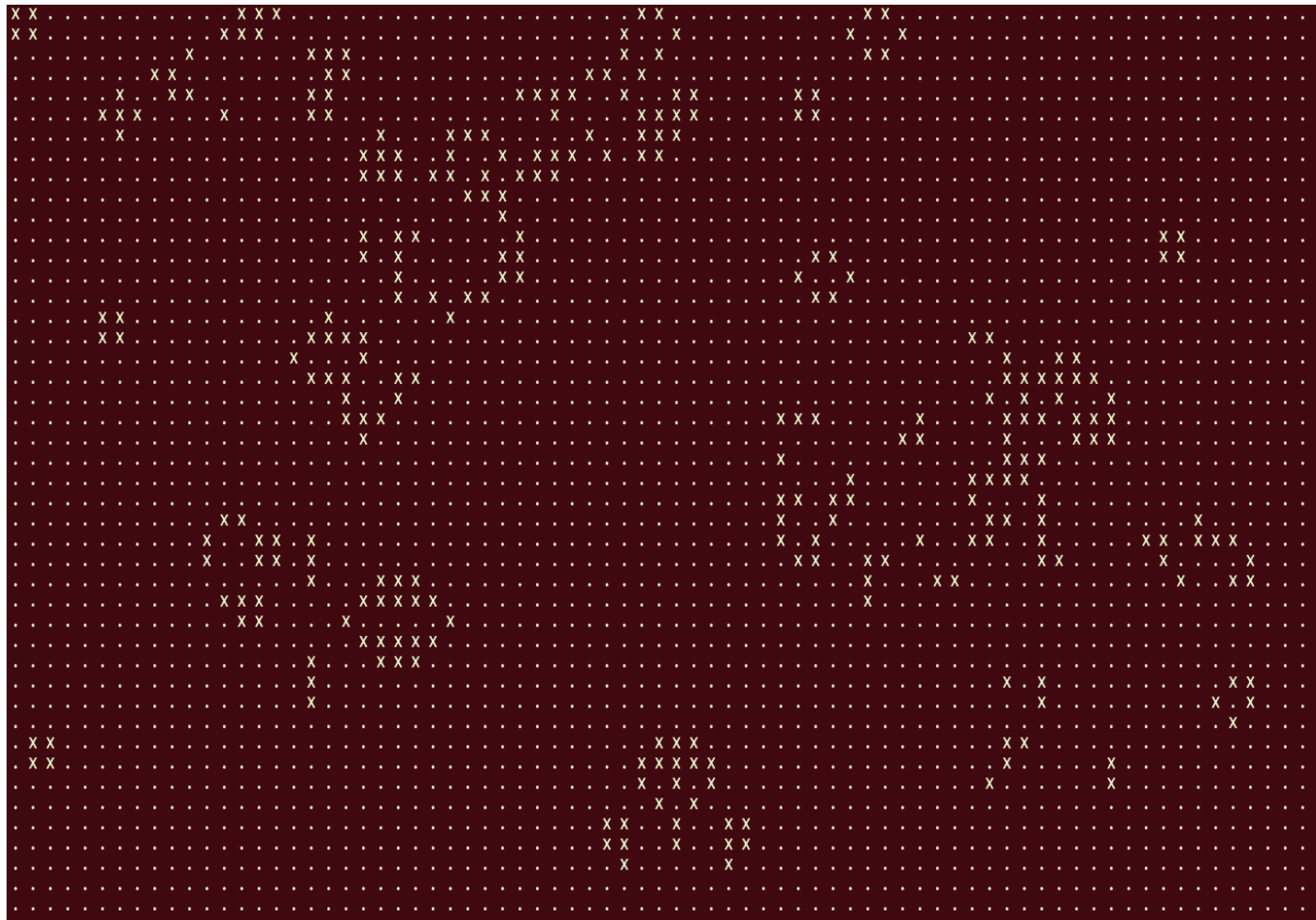
Jeu réalisant un *glider*<sub>2</sub> sur une grille 45\*75

Nous avons là, la figure du choix 6, ceci peut être considéré comme un *état stable*<sub>8</sub>, car jusqu'à ce que l'on mette fin au jeu nous avons une même figure qui se répète à l'infini. Bien sûr faire le test sur une console sera plus explicite et esthétique. La configuration pour réaliser ce canon à glider<sub>2</sub> est issue de ce site " <https://www.geeksforgeeks.org/conways-game-life-python-implementation/> " ou l'on a l'implémentation en python, au niveau de la fonction `addGosperGliderGun`.

Quant au *glider*<sub>2</sub> demandé à la première question, nous avons le choix 4 pour le voir une seule fois, et le choix 5 pour le voir sur une grille *torique*<sub>1</sub>.

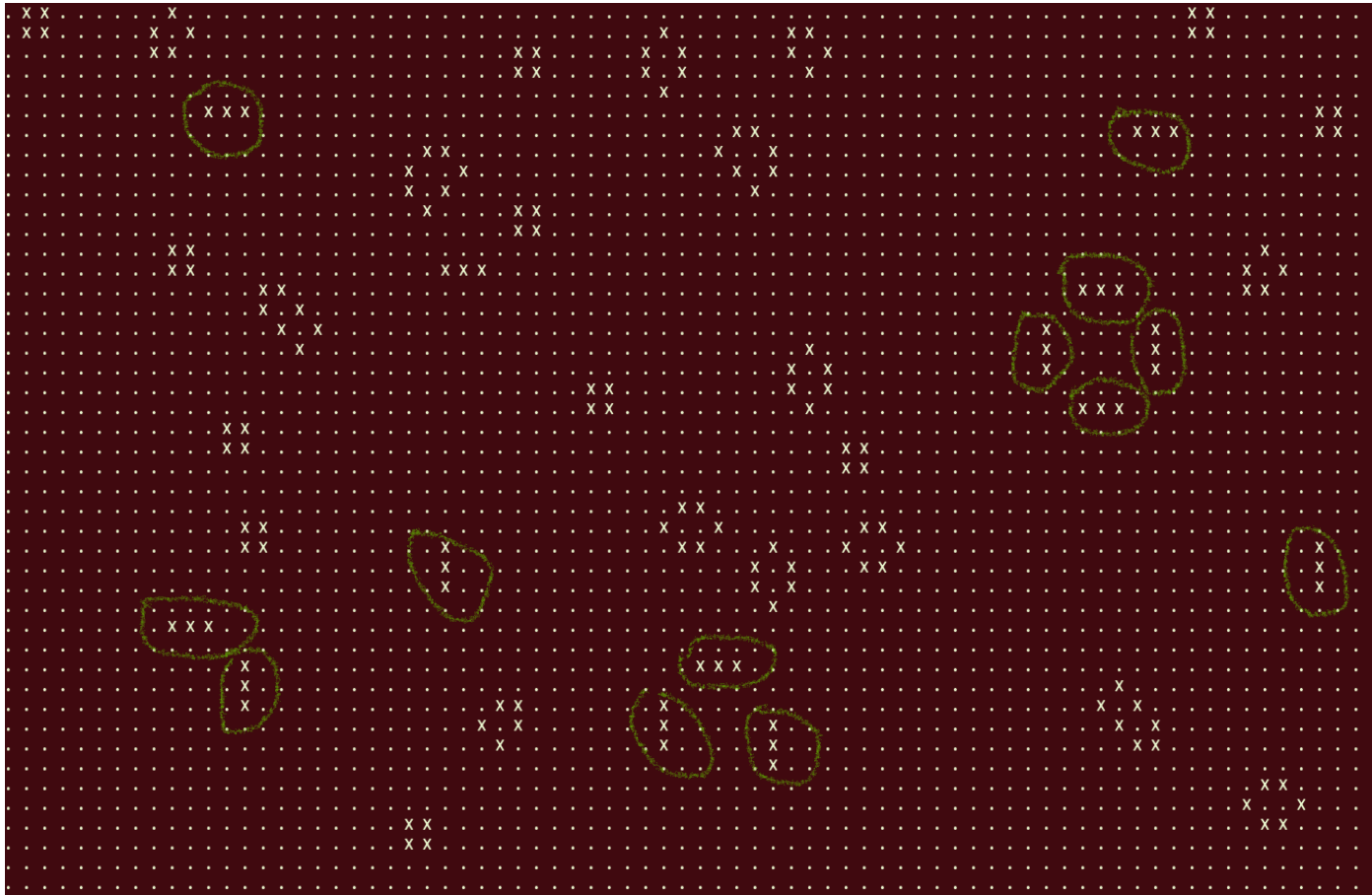
Ce rapport aurait pu être nourrissant s'il pouvait contenir des vidéos des évolutions de certaines cellules. Mais de faute que les fichiers finaux soient extrêmement lourds ou de compromettre les consignes du projet, et de plus les

logiciels de modifications de vidéos sont payants 😞, nous allons nous contenter d'images tout en essayant d'argumenter le plus, et nous verrons que rien qu'à travers les images et à l'aide de notre lexique, nous pouvons emmêtré des observations.



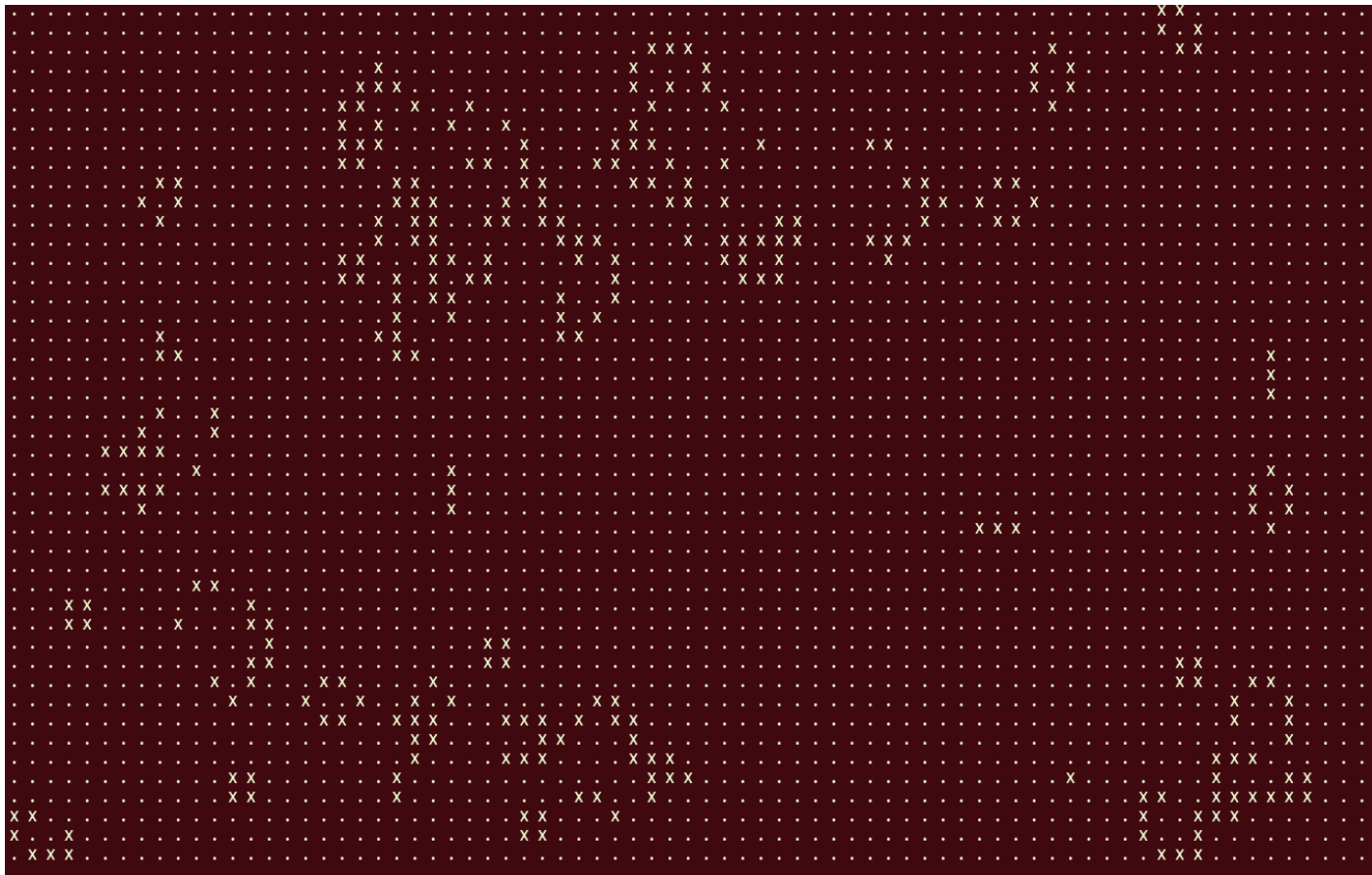
Jeu sur une grille 45\*75

Voici à quoi ressemble le cours d'une partie aléatoire en plein écran.

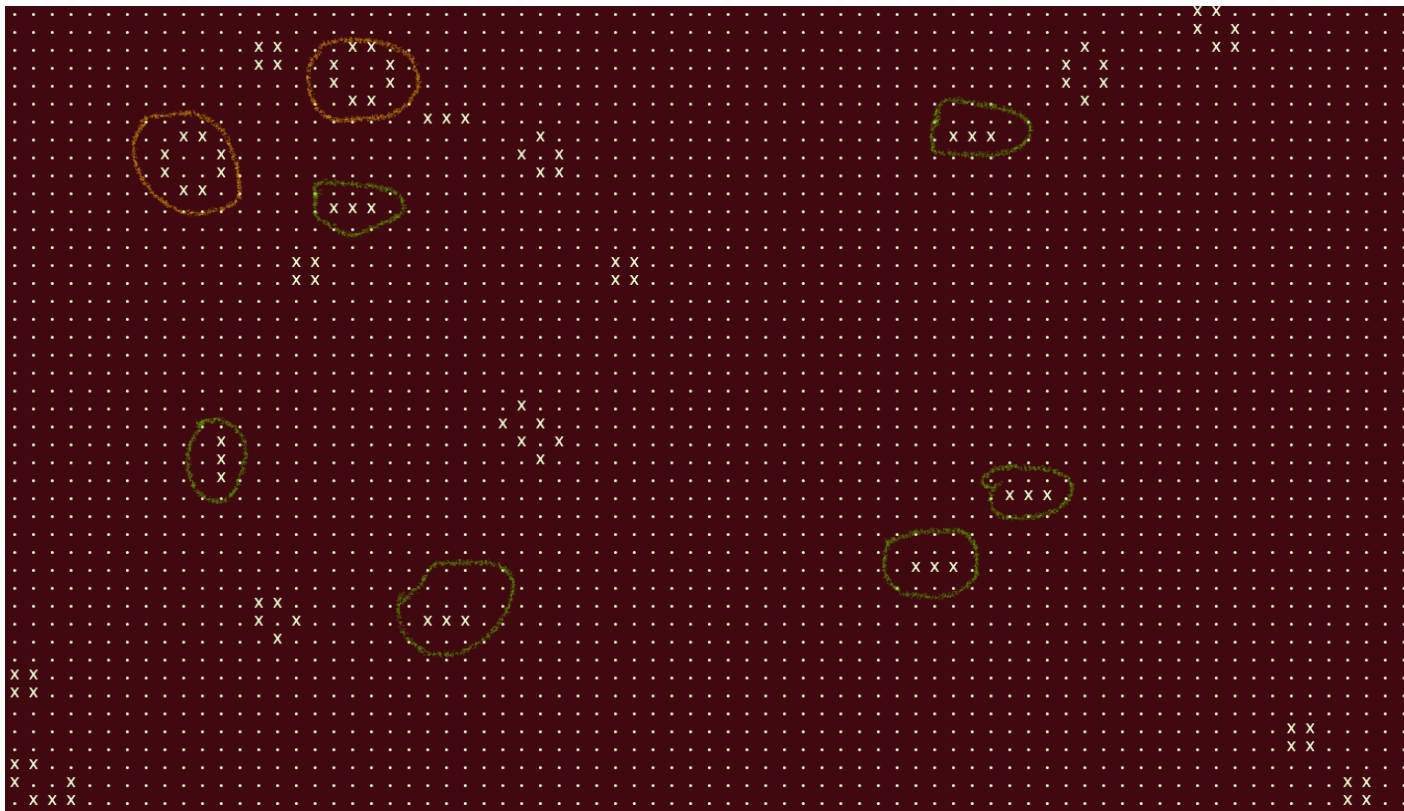


Fin de la 1<sup>ère</sup> partie au bout de 9 secondes (9 génération)

La première partie s'est terminée plutôt simplement. Elle se termine sur un *état stable*<sub>8</sub> contenant la répétition d'une figure nommée le *clignotant*<sub>4</sub> marqué en vert sur notre image. C'est une figure oscillante de période deux.

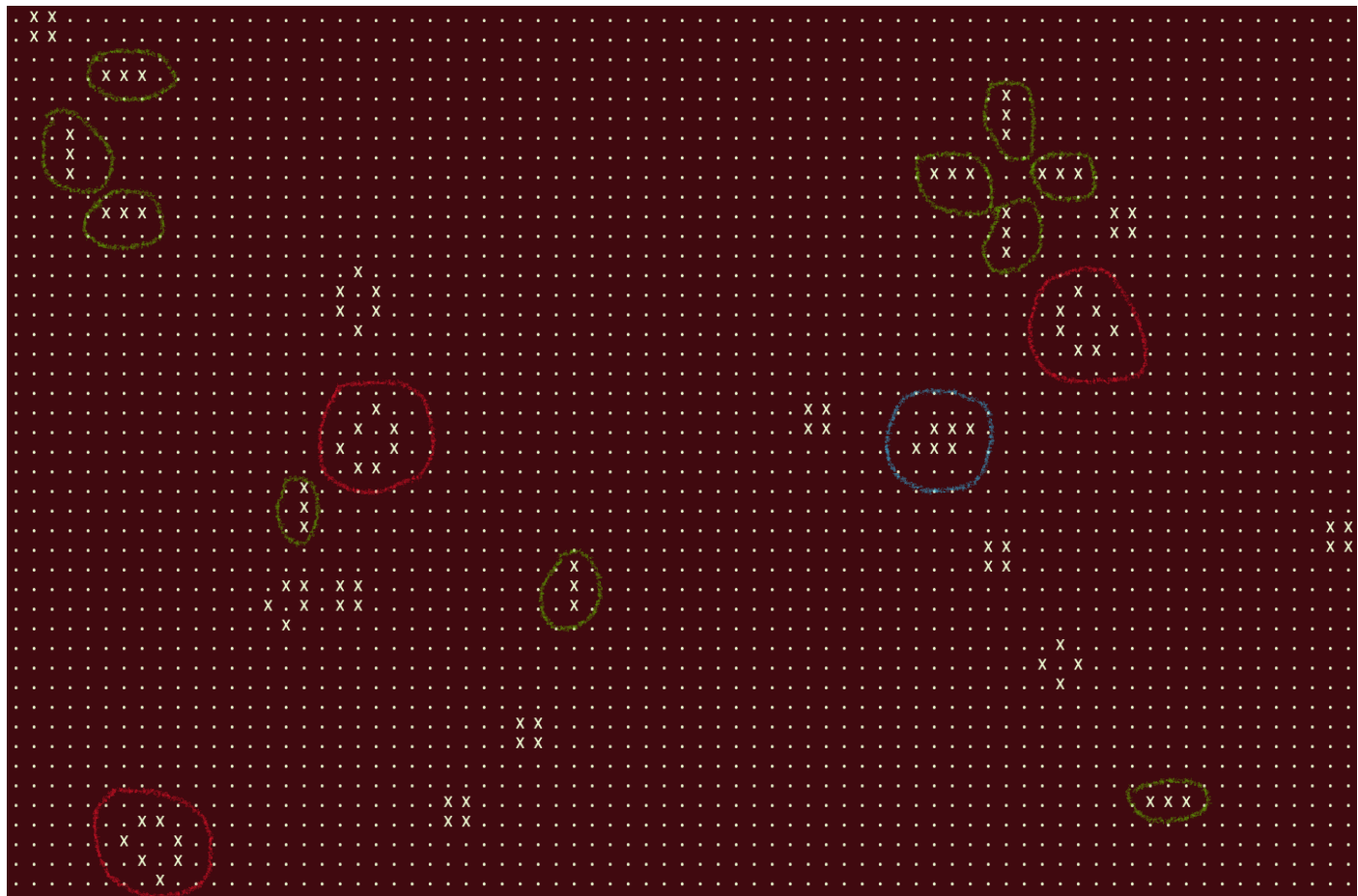


Partie 2 : grille 45\*75



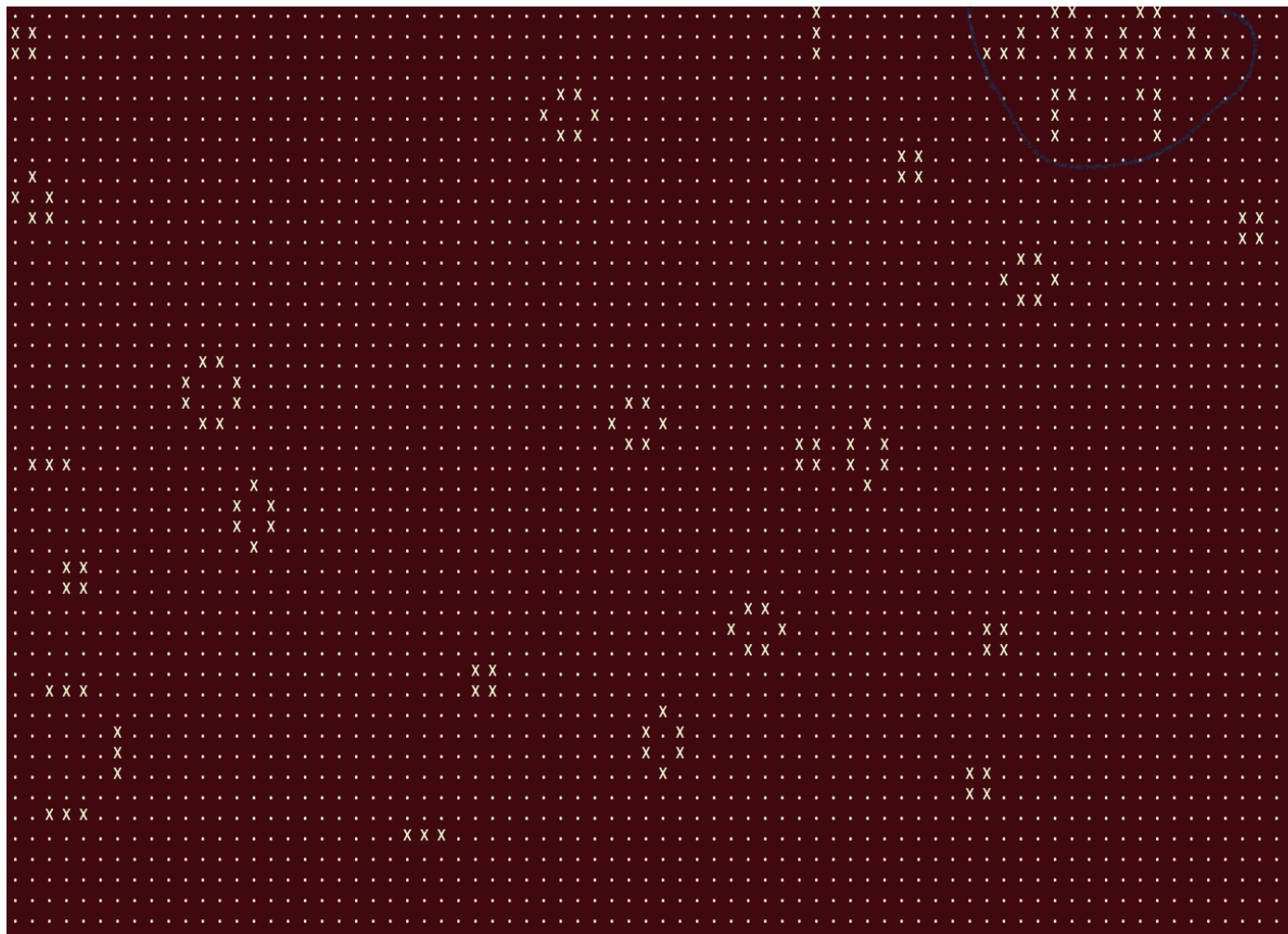
Fin partie 2

En lançant une deuxième partie nous voyons qu'elle est pareil que la première au niveau de la fin avec juste quelques cellules vivantes en moins mais contenant toujours des *clignotant*<sub>4</sub> en **vert** et une nouvelle figure en **orange** appelé la *mare*<sub>5</sub>. Contrairement au *clignotant*<sub>4</sub>, cette figure n'est pas oscillante elle n'a qu'une seule forme.



Partie sur une grille 45\*75

En moins de 10 parties de nous voyons cette fois ci deux nouvelles figures qui sont le *pain*<sub>6</sub> en **rouge** et le *crapeau*<sub>7</sub> en **bleu**, avec toujours en **vert** nos *clignotant*<sub>4</sub>. Le *pain*<sub>6</sub> est également une figure 100% stable car elle ne contient qu'une seule forme. Mais le *crapeau*<sub>7</sub> lui, est oscillant et contient 2 formes. Certaines figures se répètent plusieurs fois mais de façon inversée ou à l'envers.



Partie sur une grille 45\*75

Nous pouvons avoir également des fragments de forme connue, comme ici nous voyons la moitié bas d'un *pulsar*<sub>10</sub> représenté en **bleu**. La figure complète du *pulsar*<sub>10</sub> est disponible dans le lexique. Ça veut dire que à chaque évolution du *pulsar*<sub>10</sub> elle a apparait sous forme découpé en ne montrant que la moitié bas de la figure complète.



Même partie sur une grille 45\*75 représentant le *pulsar<sub>10</sub>*

Nous voyons ici dans la même partie une autre forme de *pulsar<sub>10</sub>* qui est évidemment apparue à moitié encore (voir lexique). Nous voyons que la première et la seconde photo de cette partie sont différentes, c'est normal car en plus du *pulsar<sub>10</sub>* il y'avait des *clignotants<sub>4</sub>* qui dont certains ont du coup changer de forme au niveau de la seconde photo.

## Synthèse

Il existe encore plein d'autre formes pour le jeu de la vie qui ont chacune leurs propriétés comme celles que nous avons vues, certains stables et d'autres oscillantes avec des périodes qui peuvent différés. A en voir le jeu il a l'air assez banal, alors qu'à très grande échelle il peut être considéré comme générateur

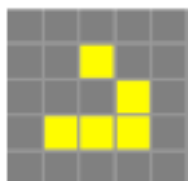
de hasard, car nous avons très souvent des situations complètement chaotiques, qu'on ne peut donc pas du tout prévoir, nous l'avons bien vu lors de nos tests précédents. Il a été montré que le jeu de la vie peut renfermer toutes les capacités que peut contenir une machine de turing, ce qui peut permettre de réaliser d'énormes choses, beaucoup plus que le simple fait de "jouer". Beaucoup de passionnés de simulation et d'**automates cellulaires** s'amuse souvent à changer les règles du jeu de la vie pour découvrir de nouveaux horizons pour cela c'est sur l'affichage qu'il faudra miser pour avoir de gigantesques grilles et analyser leurs comportements.

**Stephen Wolfram** (grand Informaticien et Physicien des années 80) a étudié un jeu sur des automates cellulaires semblable au jeu de la vie. Il se fait sur une ligne de 8 cases et contient aussi des règles ( $2^8$  possibilités, 256 règles) dont toutes ont été trouvées par **Wolfram** et dont l'ensemble du projet et bien plus sont disponible ici " <https://www.wolframscience.com/nks/> " ou encore " <https://www.wolframalpha.com> "

## Lexique

**Torique<sub>1</sub>** : Un tableau torique ou matrice torique, est considéré(e) comme une figure géométrique circulaire. C'est-à-dire nous sommes sur une sphère et notre figure tourne, tourne et tourne encore

**Glider<sub>2</sub>** : Un Glider, appelé planeur en français, est le nom donné par



cette figure . Au cours du jeu c'est une figure qui évolue de case en case en gardant les mêmes formes.

**GIF<sub>3</sub>** : C'est un format d'images numériques permettant de créer des images animées ou à fond transparent. Elles ont une extensions .gif .

**Le Clignotant<sub>4</sub>** : C'est l'une des figures à laquelle nous pouvons aboutir en jouant au jeu de la vie. C'est une figure qui contient trois cellules horizontales, et à l'instant suivant elles

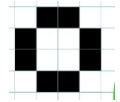


deviennent verticale, ensuite à nouveau horizontale et ainsi de suite.

Comme ceci : 

Les cellules noires étant vivantes et les cellules blanches mortes.

**Mare<sub>5</sub>** : la mare est une figure 100% stable qui ne contient qu'une seule forme, elle contient 8 cellules vivantes qui sont de la forme



**Pain<sub>6</sub>** : Cette figure contient également une seule forme modéliser de

cette manière .

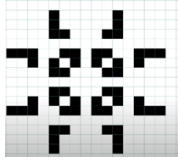
**Crapeau<sub>7</sub>** : C'est une forme oscillante comme le clignotant qui se


représente sous deux formes qui sont : .

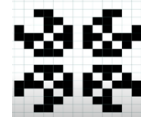
**État stable<sub>8</sub>** : C'est lorsqu'une partie se termine sur des motifs qui se répèteront à l'infini sans changement.

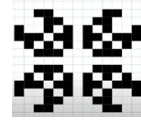
**État stable évolutif<sub>9</sub>** : C'est lorsqu'une partie se termine sur un état qui est considéré comme un état stable, mais qui évolue avec me temps. On peut avoir par exemple durant 10 secondes (ou 10 générations) un même motif, ensuite le motif change et devient stable durant 30 génération ensuite change, ainsi de suite.

**Pulsar<sub>10</sub>** : C'est une figure oscillante de période 3, ça veut dire qu'elle se répète tous les 3 tours. L'évolution de cette figure étant trop rapide nous ne pouvons pas cibler toutes ses formes, mais la première forme qui nous ait apparue dans notre rapport est la moitié de la suivante



Elle nous ait apparue dans le jeu sous cette forme   
Nous voyons bien que c'est la moitié bas de la première forme.



Une autre des 3 formes du pulsar est celle-ci  qui nous est

apparue dans notre jeu sous cette forme  qui est son exacte  
moitié bas.

Retrouvez sur " <https://www.youtube.com/watch?v=S-W0NX97DB0> " toutes les  
figures qui ont été présentées dans ce lexiques.