Tests unitaires

Ce document permet de fournir plus d'informations concernant les tests unitaires. Les tests unitaires permettent de vérifier une fonctionnalité présente dans l'application dans tous les cas possibles d'utilisation. C'est une étape obligatoire dans le processus de développement d'une application, assurant ainsi la stabilité du programme. Sans les tests unitaires, il devient compliqué de pouvoir assurer le fonctionnement normal de toutes les fonctionnalités.

Nous avons effectué les tests unitaires sur 4 fonctionnalités du programme :

- Grille torique

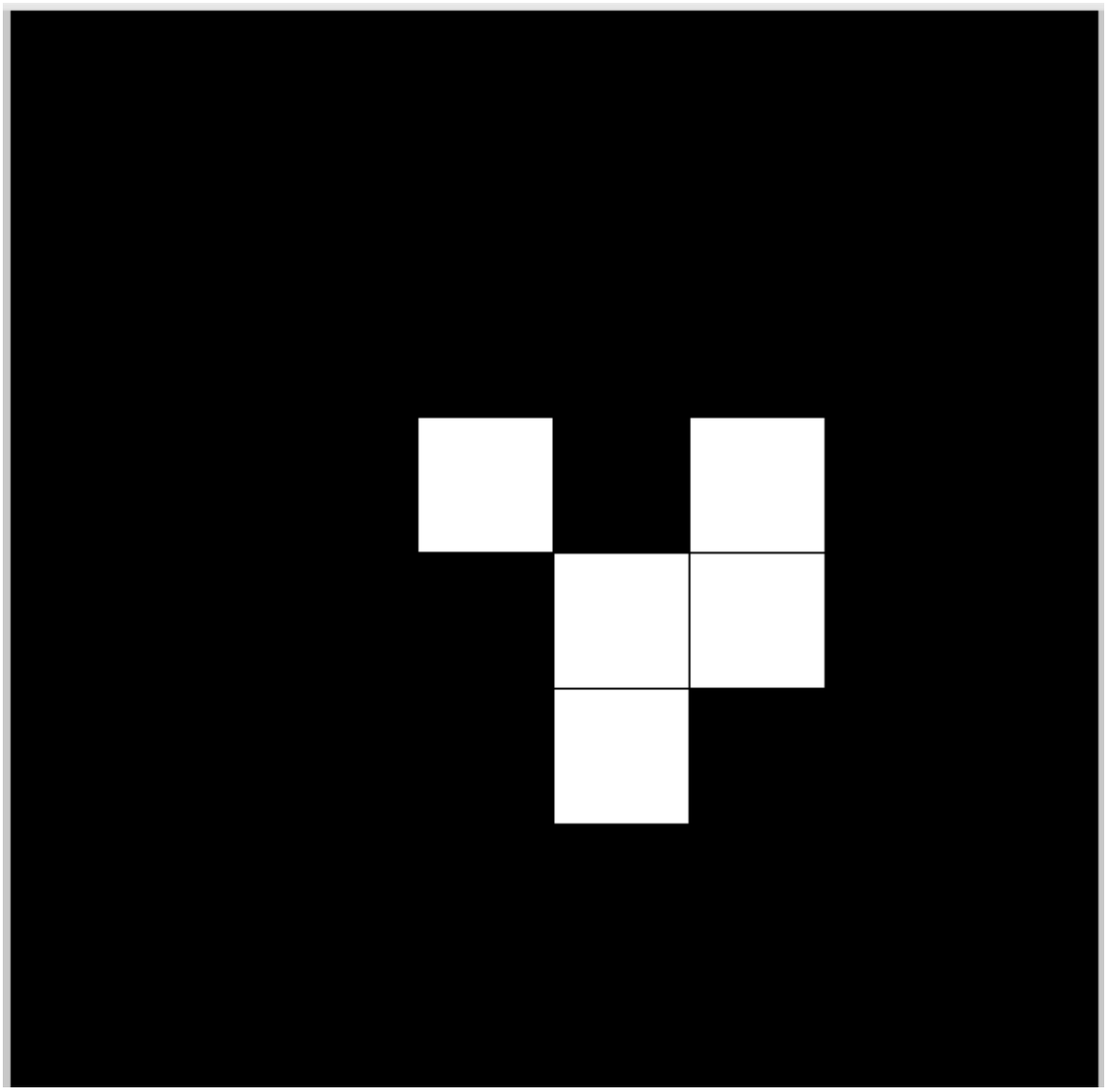
- Cellule obstacle

- Initialisation de grille préprogrammés

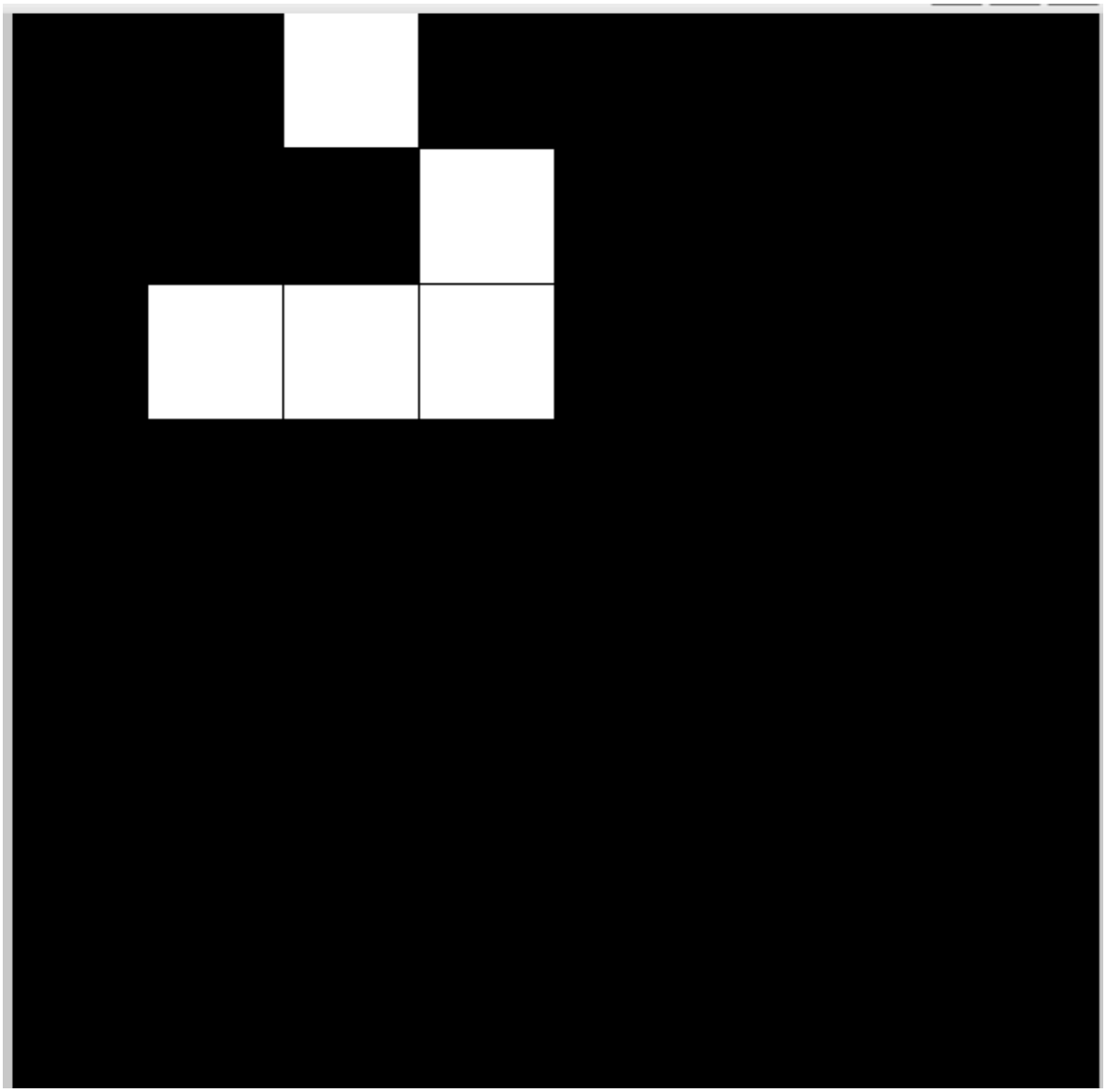
- Parallélisation du traitement de la grille.

Grille torique

La grille torique est une grille "sans bordure", où un élément se déplaçant toujours horizontalement en bas ne serait pas bloqué en bas, mais il reviendrait en haut. Elle permet de simuler une grille où l'on peut se déplacer librement, sans rencontrer de limite.



Le planeur, se déplaçant toujours en bas à droite de manière infinie, se déplace vers la bordure ...



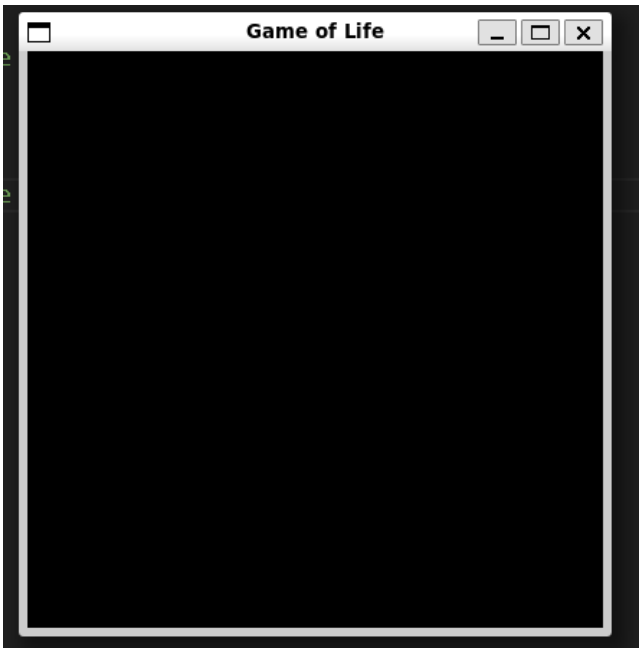
… mais il réapparait en haut à gauche, et ce de manière infinie …

4 tests unitaires ont été produits pour la grille torique :

1/ Grille complètement vide (“borderless1”)

Explication : Génère une grille complètement vide
Résultat attendu : une fenêtre affichant une grille complètement vide

<u>Résultat obtenu :</u> une fenêtre affichant une grille complètement vide après compilation et exécution



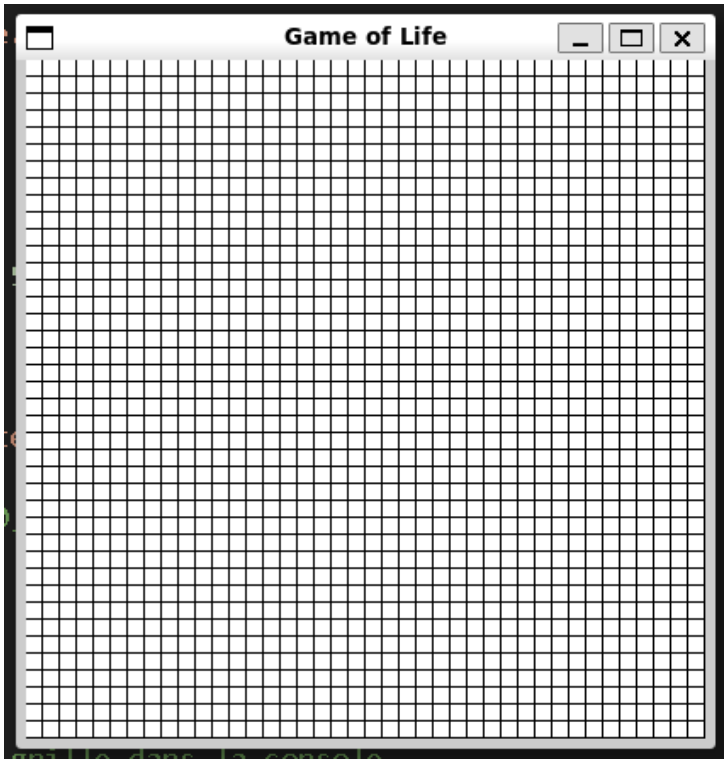
Test réussi !

2/ Grille complètement remplie (“borderless2”)

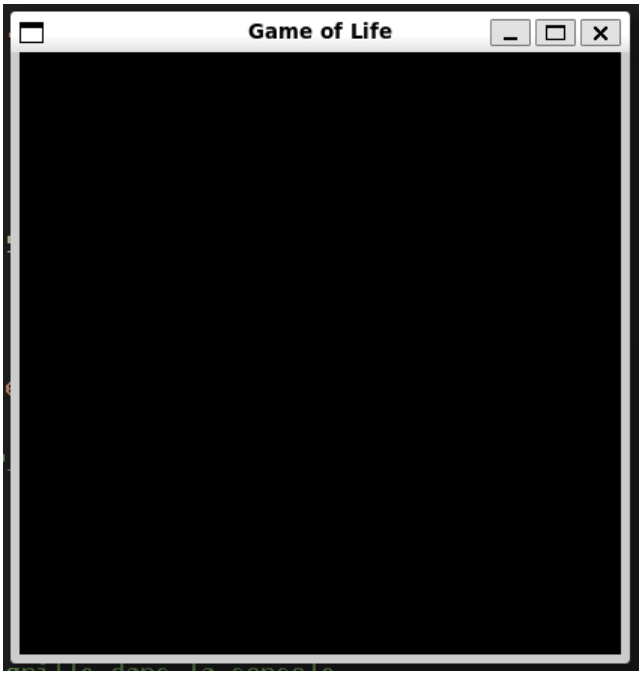
Explication : Génère une grille complètement remplie. Toutes les cellules ayant chacune 8 voisins en vie, elles vont toutes mourir lors de la prochaine itération

Résultat attendu : une fenêtre affichant une grille complètement remplie pour la première itération, puis complètement vide

<u>Résultat obtenu :</u> une fenêtre affichant une grille complètement remplie puis une grille complètement vide après compilation et exécution



Une grille complètement remplie pour la première itération ...



... Puis une grille complètement vide !

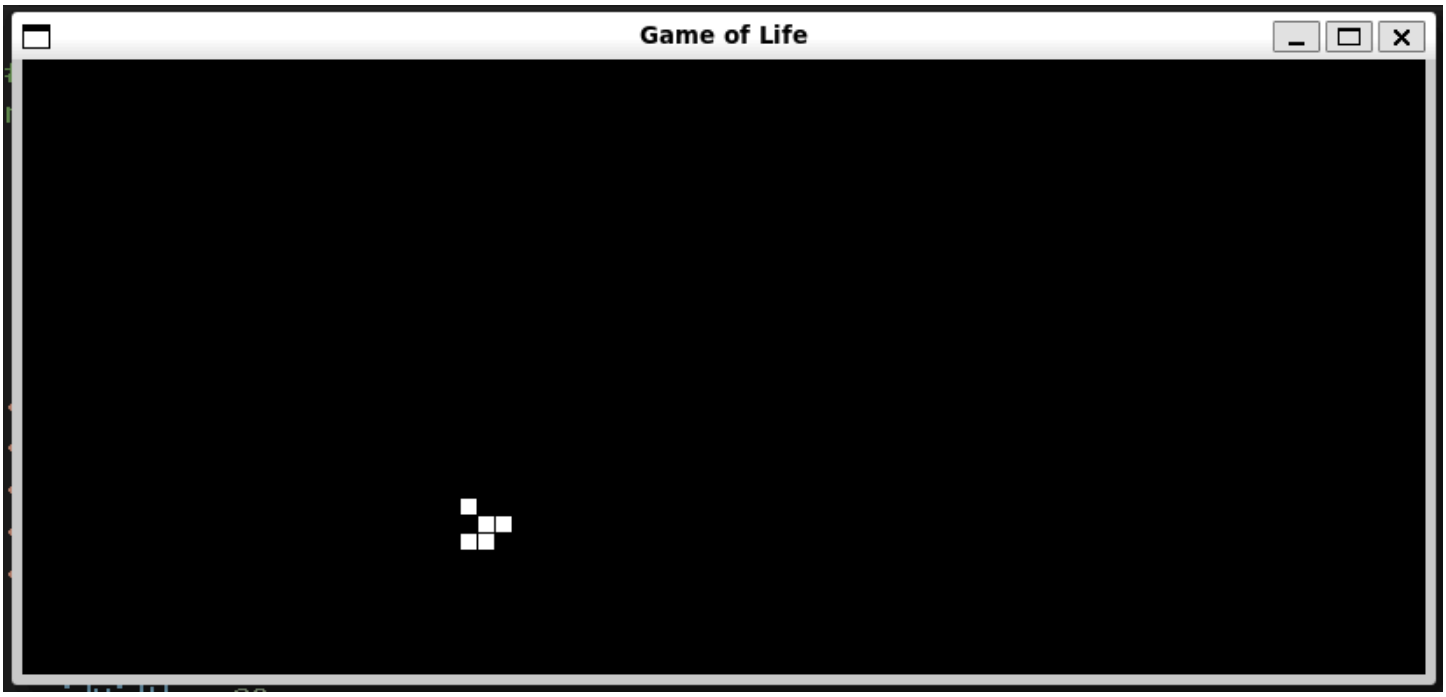
Test réussi !

3/ Grille avec un planeur (“borderless3”)

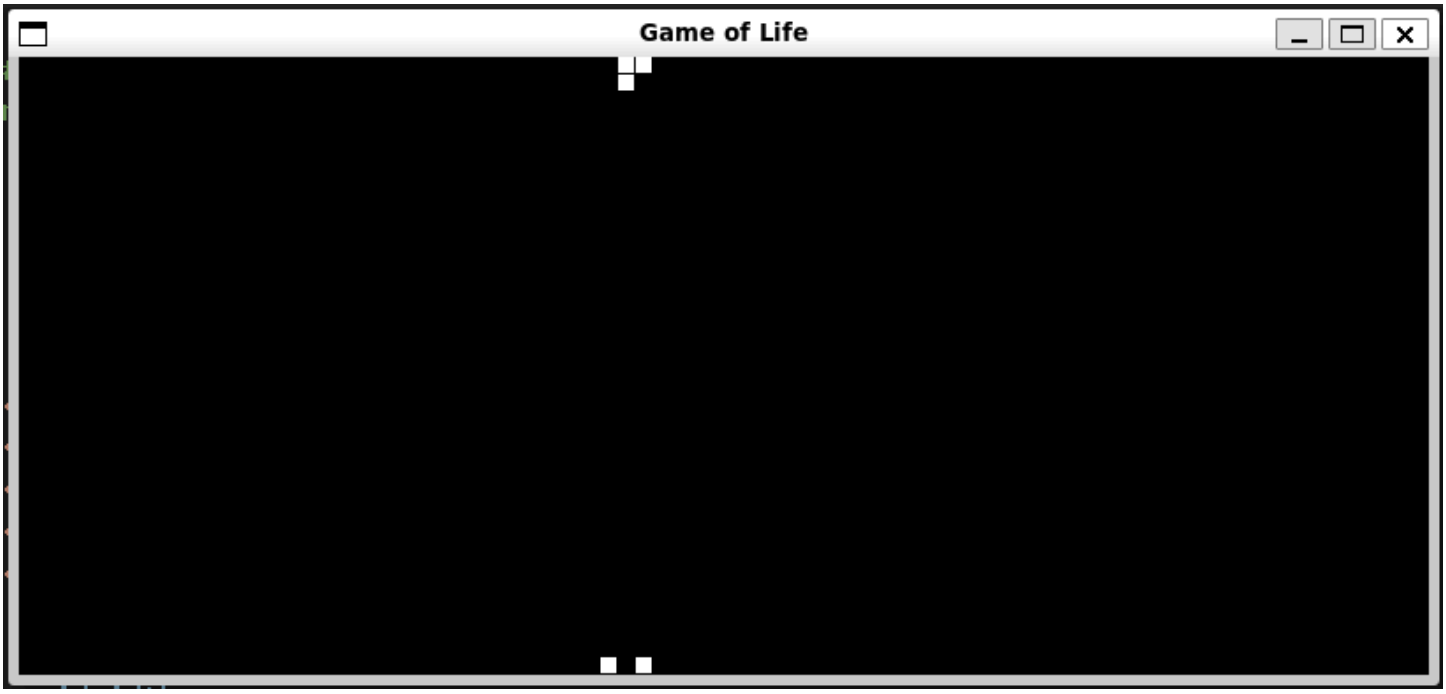
Explication : Génère une grille contenant un planeur allant en bas à droite.

Résultat attendu : une fenêtre affichant une grille contenant un planeur, se déplaçant en bas à droite. Une fois qu’il croise une bordure, il doit réapparaitre à l’opposé de cette bordure.

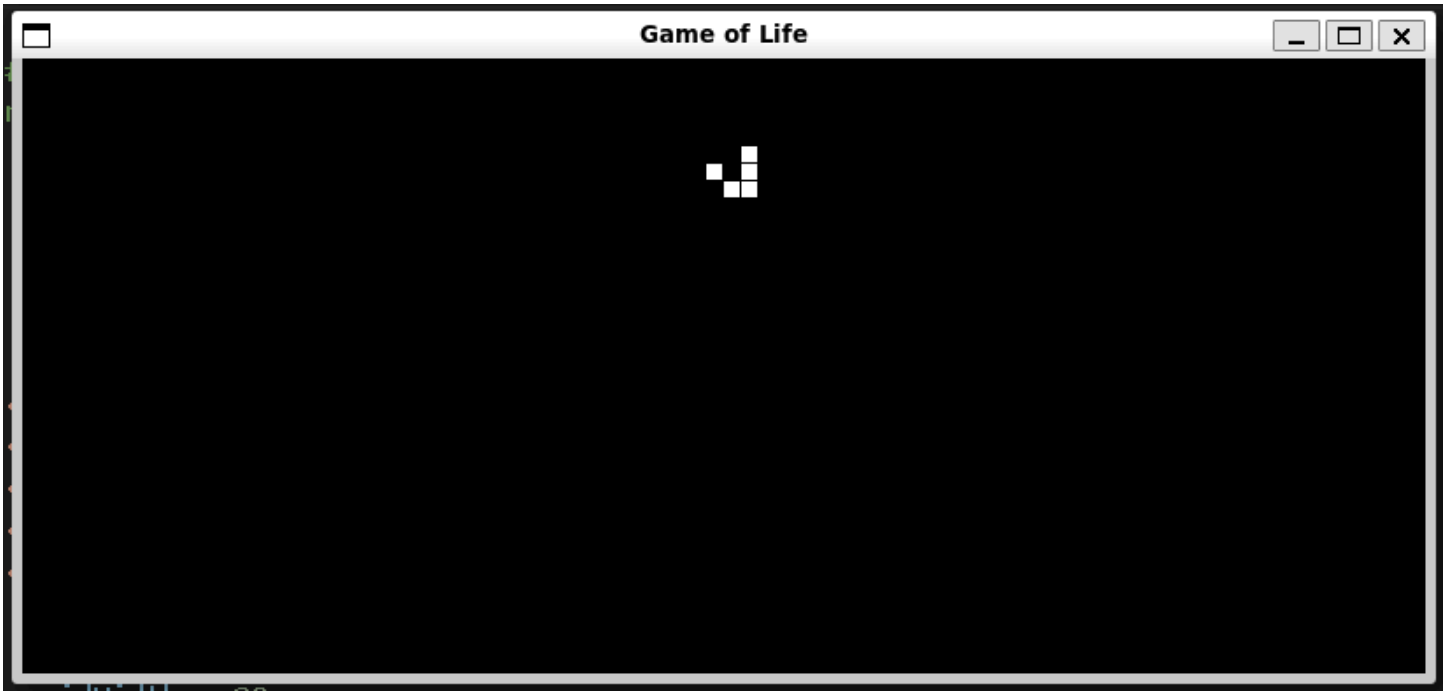
Résultat obtenu : une fenêtre affichant une grille avec un planeur. Celui-ci réapparait progresssivement dans la bordure opposée.



Le planeur se dirige en bas, il doit réapparaitre en haut ...



Il se reforme en haut ...



Il est réapparu en haut !

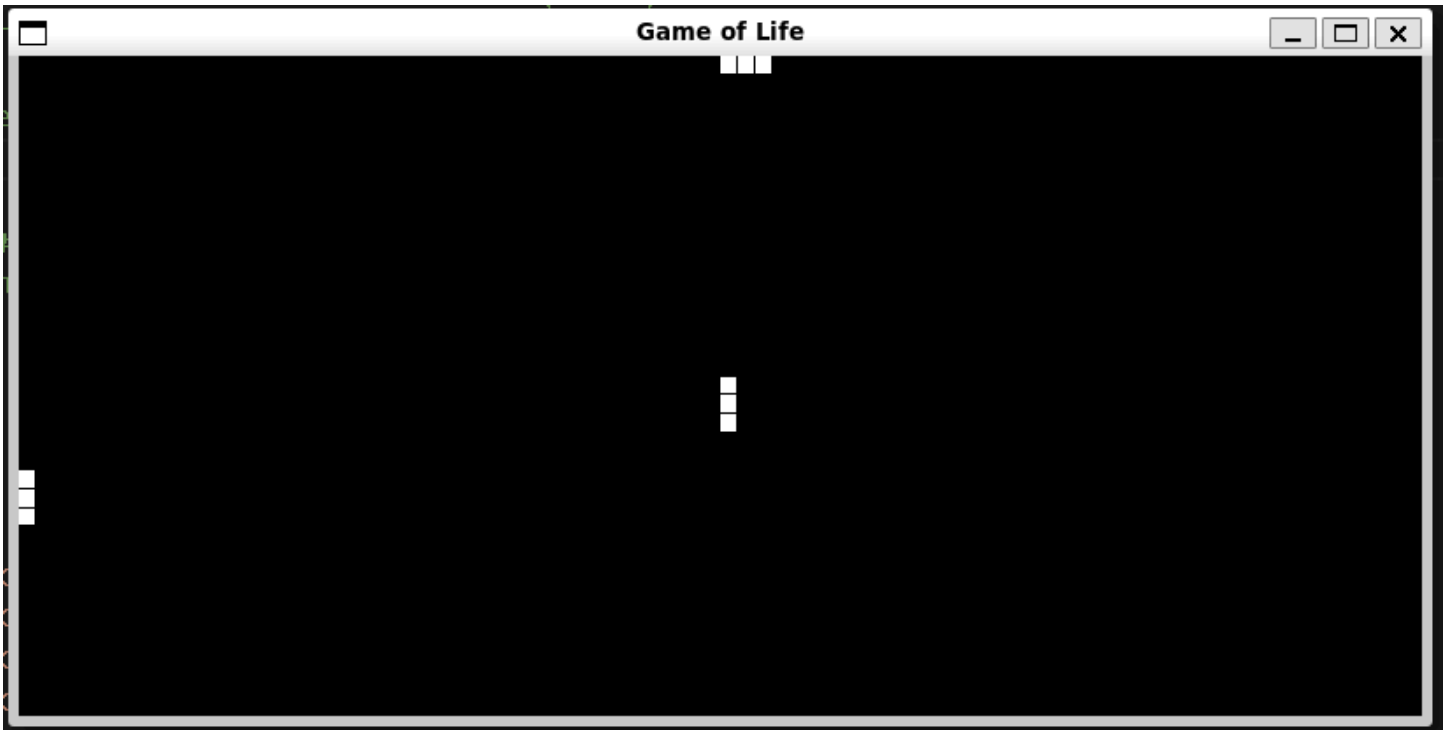
Test réussi !

4/ Grille avec les blinkers (“borderless4”)

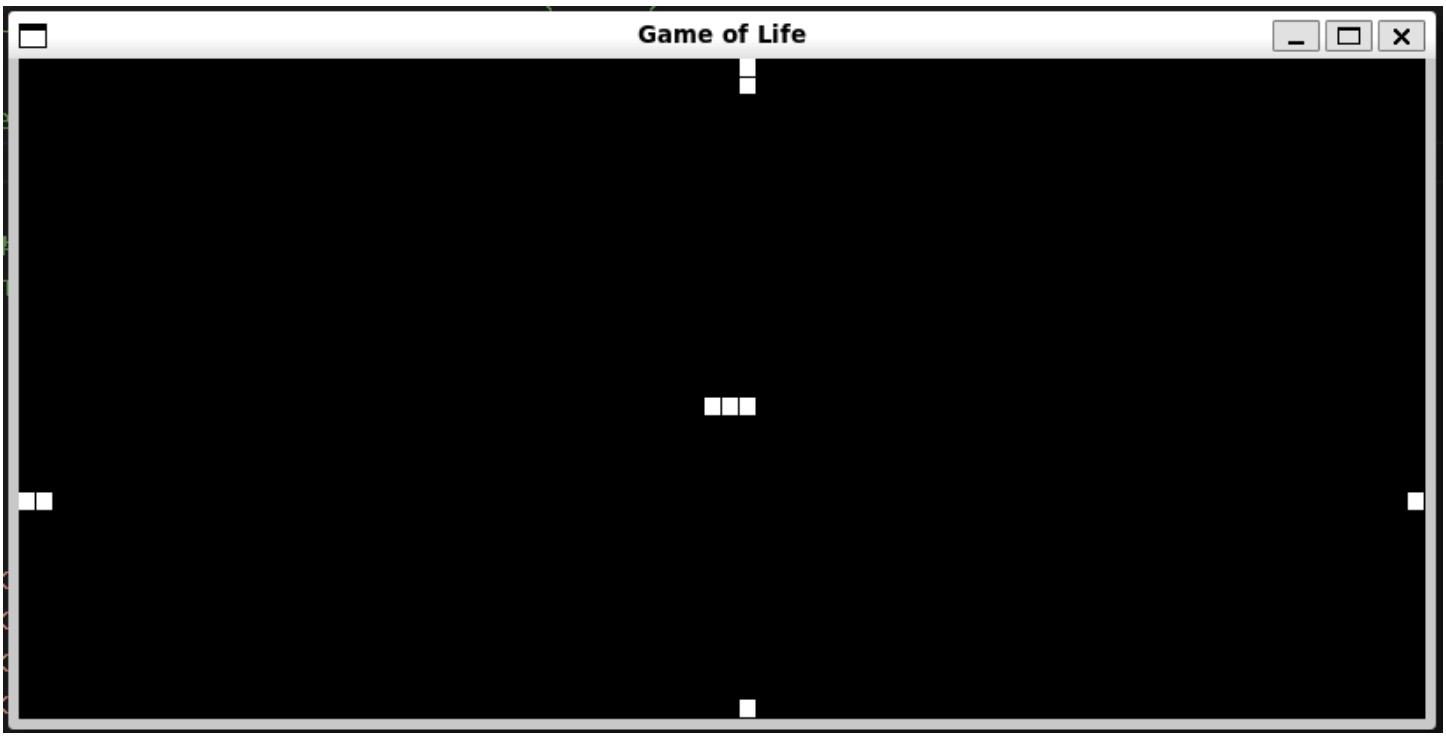
Explication : Génère une grille vide. Deux blinkers se trouvent sur des bordures, tandis qu’un blinker se trouve au centre.

Résultat attendu : une fenêtre affichant une grille, qui, au fur et a mesure que la grille se met à jour, fait clignoter les blinkers

Résultat obtenu : une fenêtre affichant une grille contenant 3 blinkers clignotant.



Une grille avec les blinkers ...



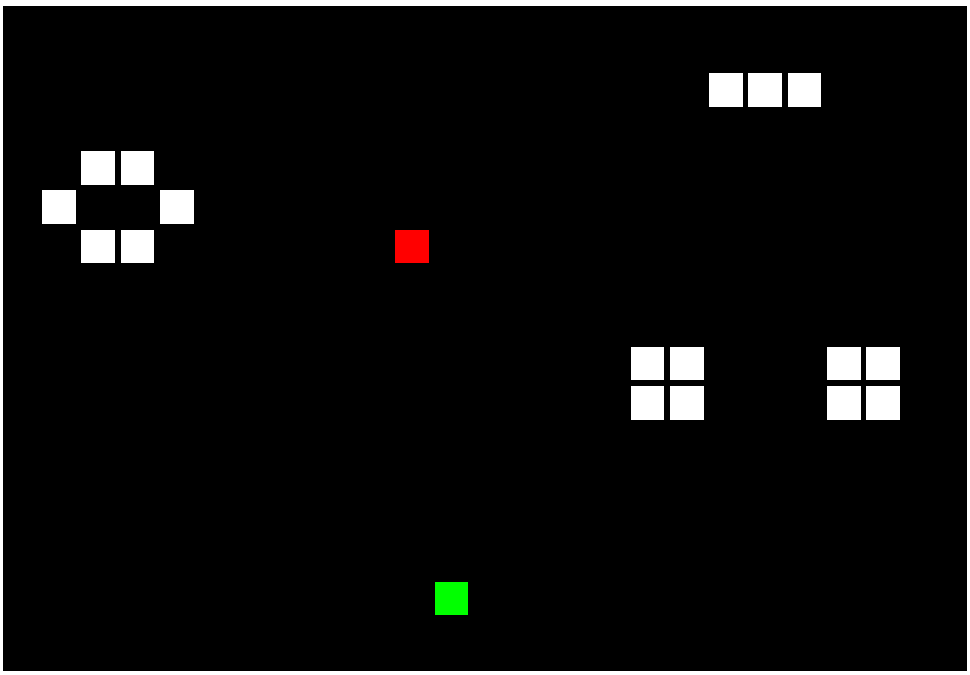
Ils clignotent !

Test réussi !

Test réussi : 4/4

Cellules Obstacles

Une cellule obstacle est une cellule qui ne change pas d'état. Elle est soit vivante, soit morte mais elle ne peut pas changer d'état. Les cellules obstacles affichés en vert sont les cellules obstacles vivantes, et les rouges sont mortes.



Une cellule obstacle vivante (verte), une autre morte (en rouge) et des cellules classiques en blancs.

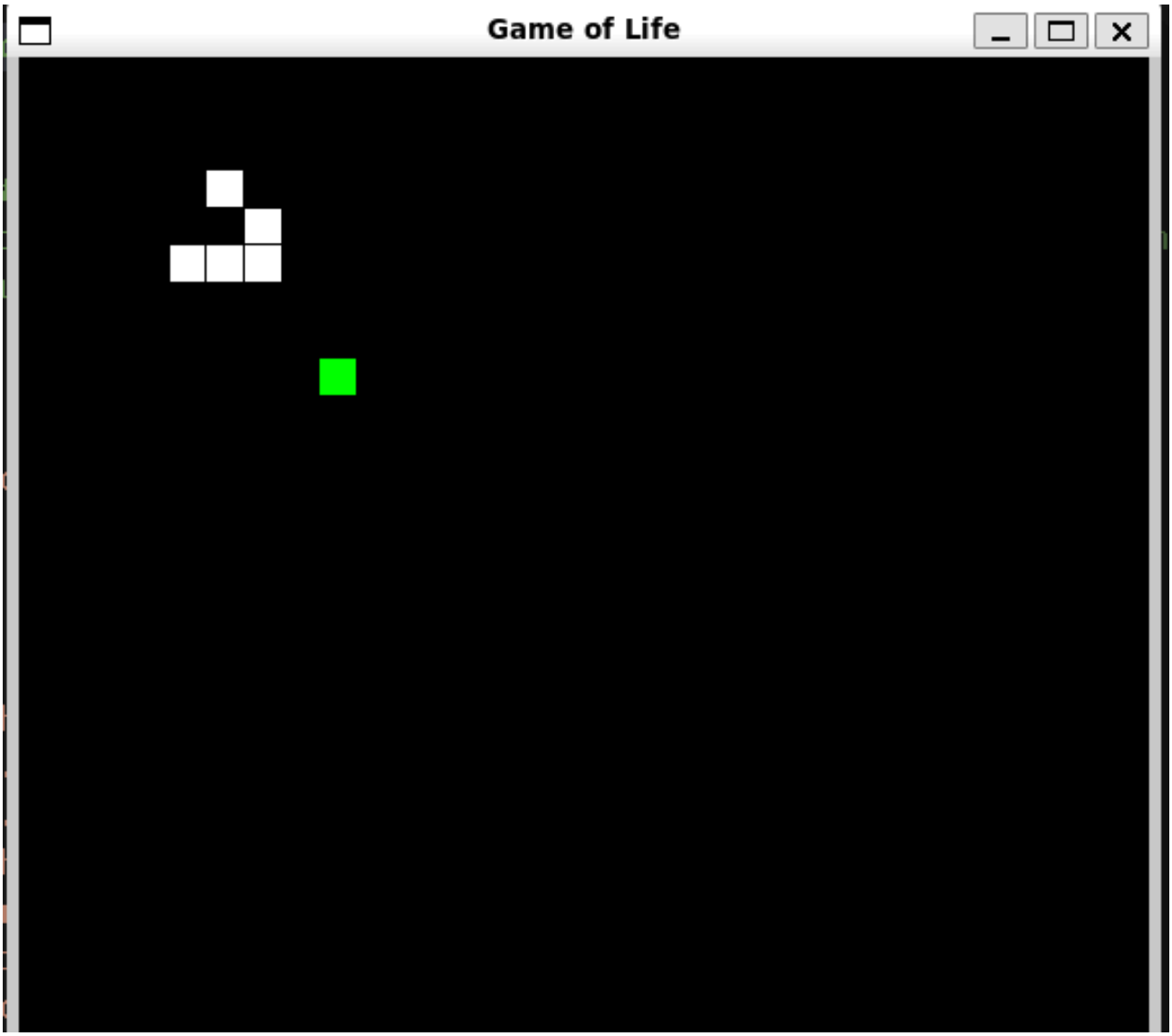
3 tests unitaires ont été produits pour les cellules obstacles :

1/ Planeur et cellule obstacle vivante ("obstacle1")

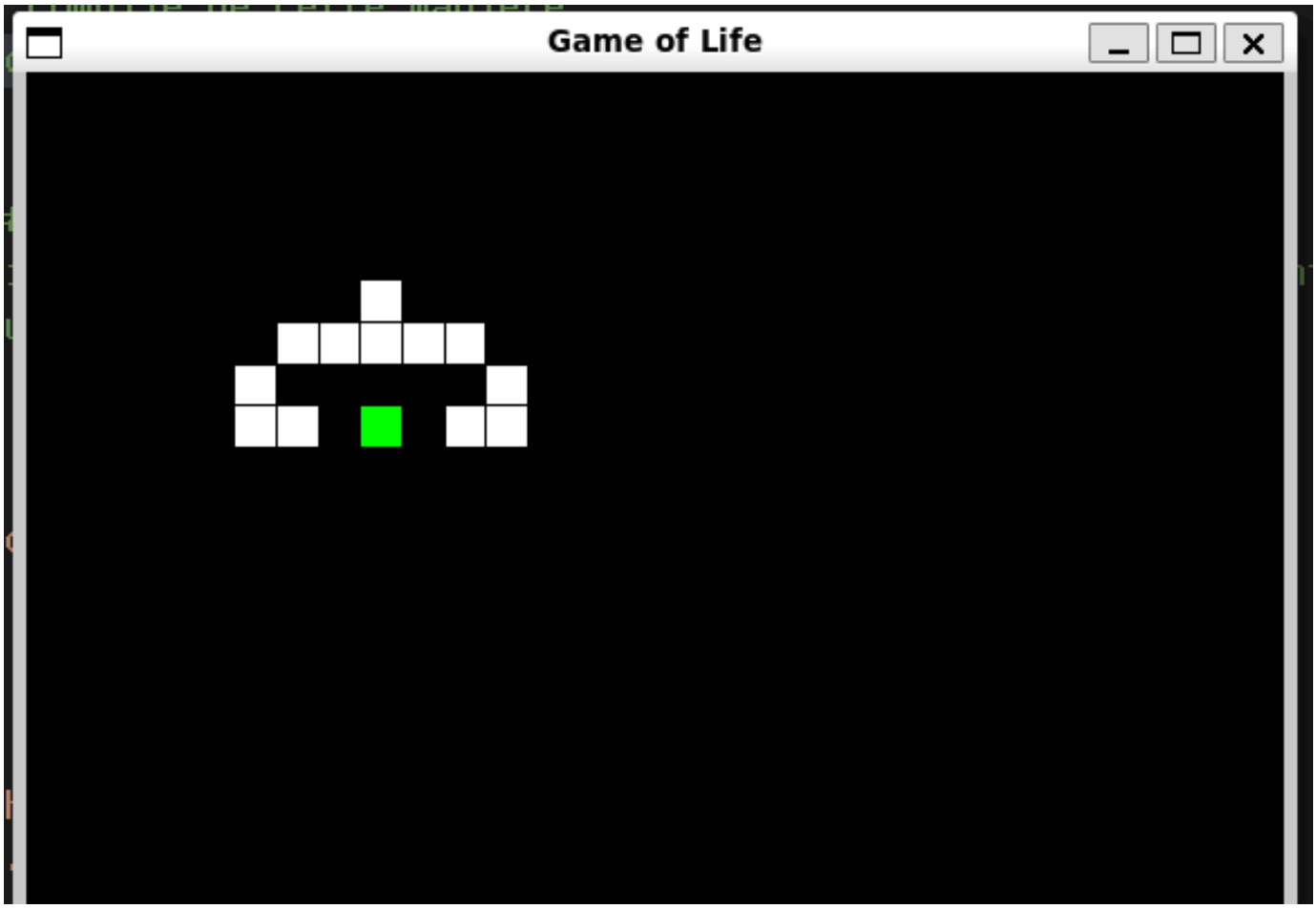
Explication : Génère une grille contenant une cellule obstacle vivante et un planeur fonçant vers la cellule obstacle.

Résultat attendu : une fenêtre affichant une grille, et un planeur s'écrasant sur la cellule obstacle verte, coupant sa trajectoire

Résultat obtenu : une fenêtre affichant une grille avec un planeur s'étant écrasé sur la cellule obstacle.



Le planeur se dirige vers la cellule obstacle ...



Le planeur s'est écrasé et a formé un état stable autour de la cellule obstacle

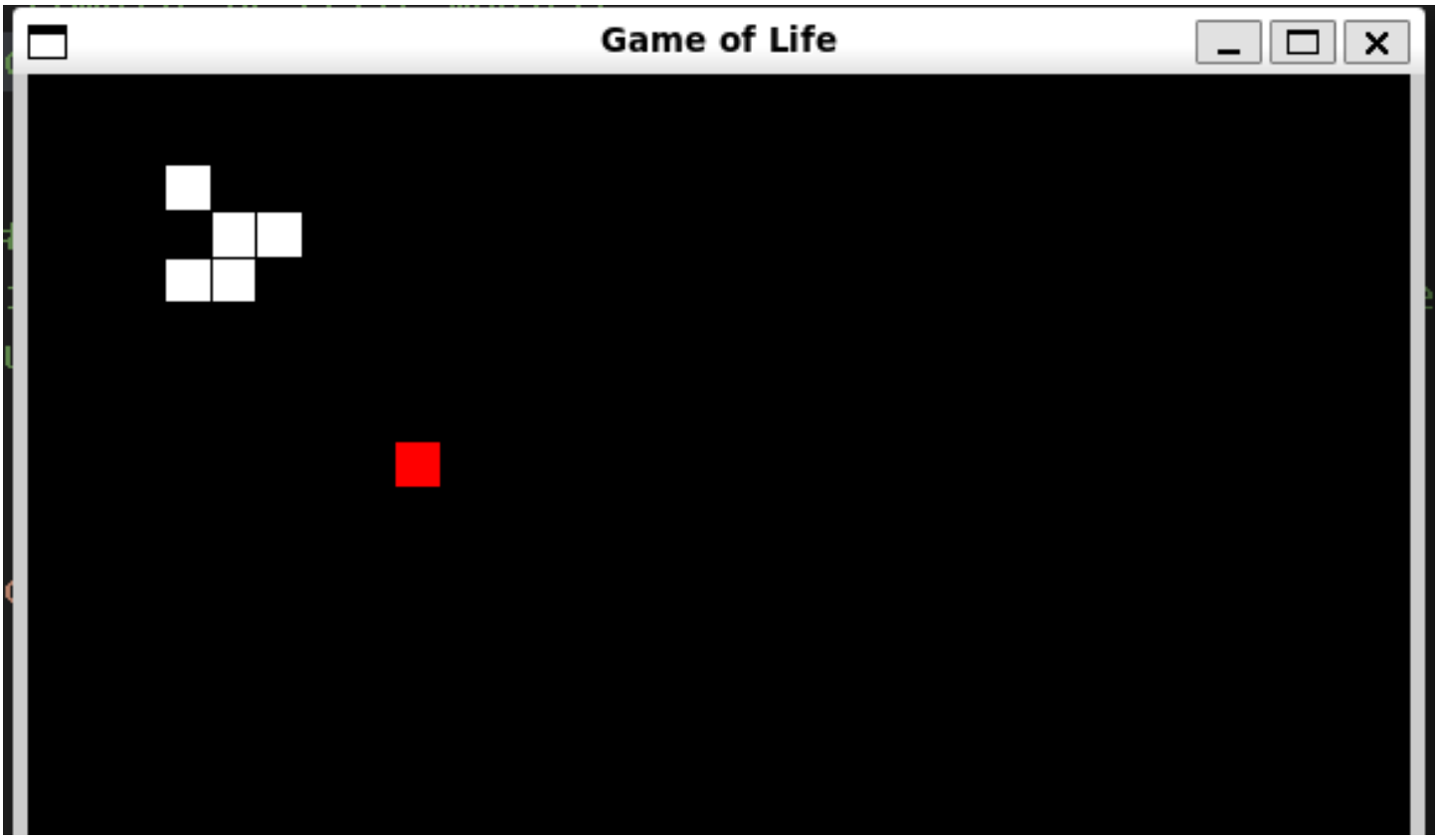
Test réussi !

2/ Planeur et cellule obstacle morte ("obstacle2")

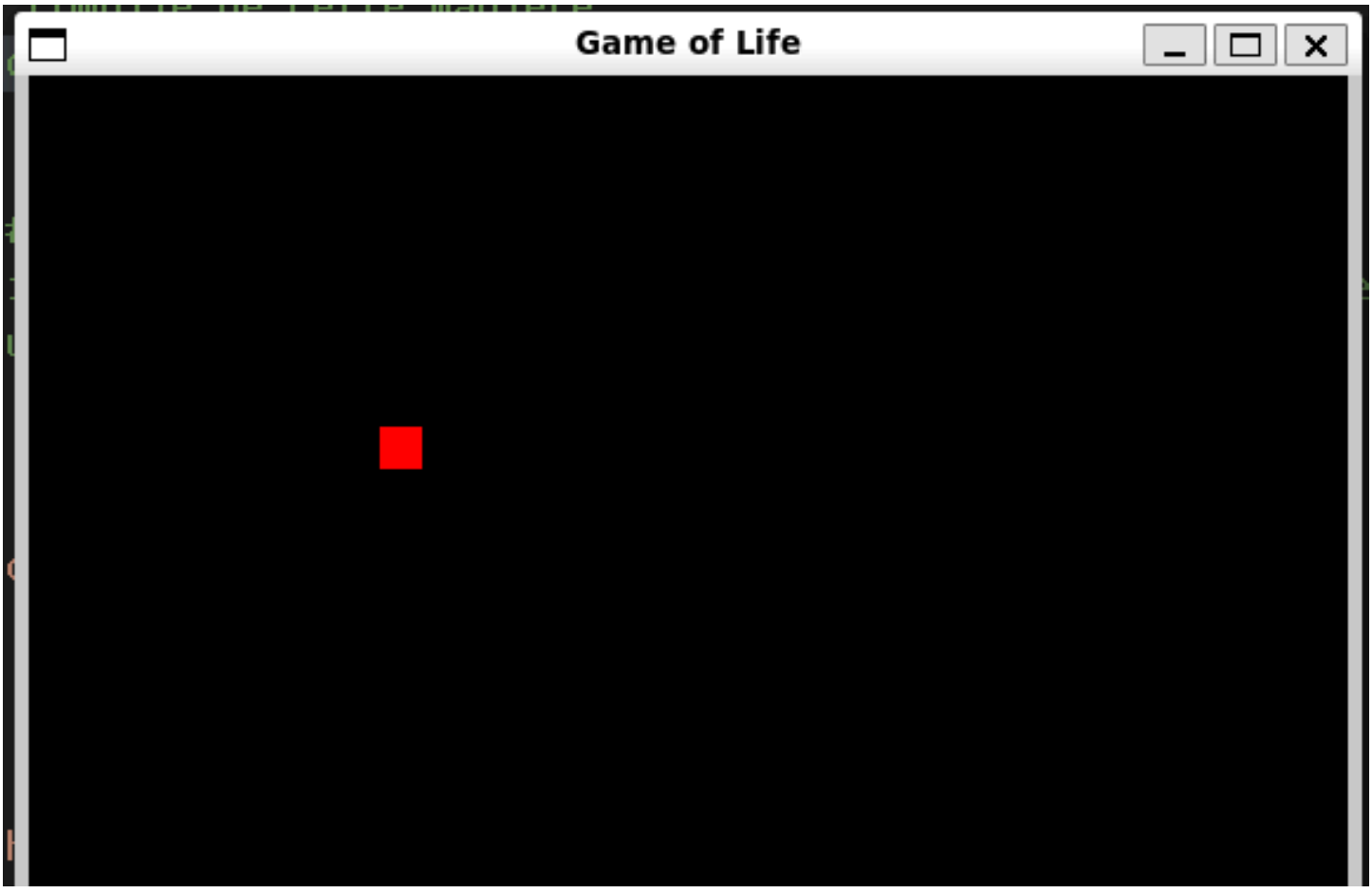
Explication : Génère une grille contenant une cellule obstacle morte et un planeur fonçant vers la cellule obstacle.

Résultat attendu : une fenêtre affichant une grille, et un planeur s'écrasant sur la cellule obstacle rouge, faisant disparaître le planeur

Résultat obtenu : une fenêtre affichant une grille avec un planeur disparaissant sur la cellule obstacle.



Le planeur se dirige vers la cellule obstacle ...



Le planeur s'est écrasé et a disparu

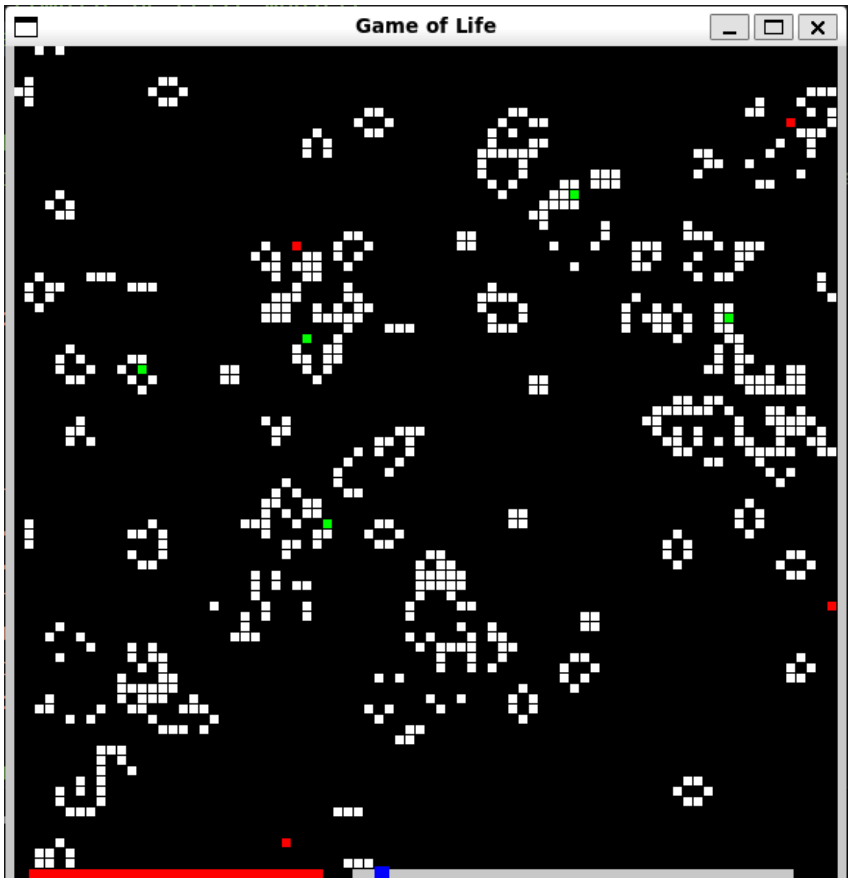
Test réussi !

3/ Grille aléatoire (“obstacle3”)

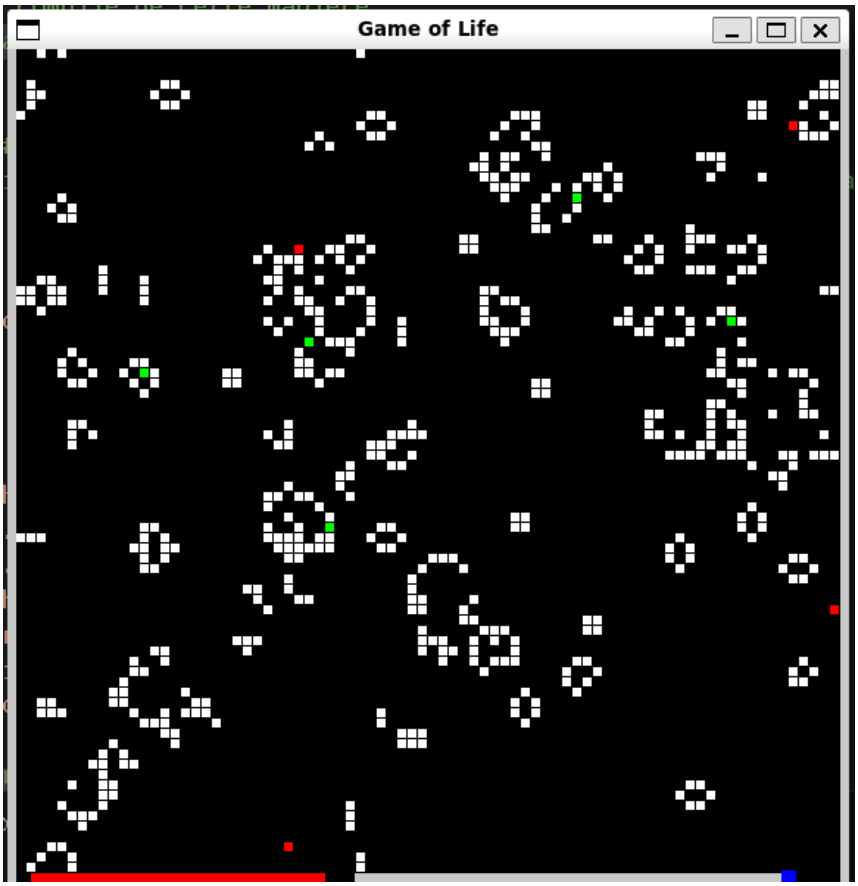
Explication : Génère une grille aléatoirement contenant des cellules obstacles disposés aléatoirement (1% des cellules classiques ont une chance d’être une cellule obstacle)

Résultat attendu : une fenêtre affichant une grille se comportant normalement

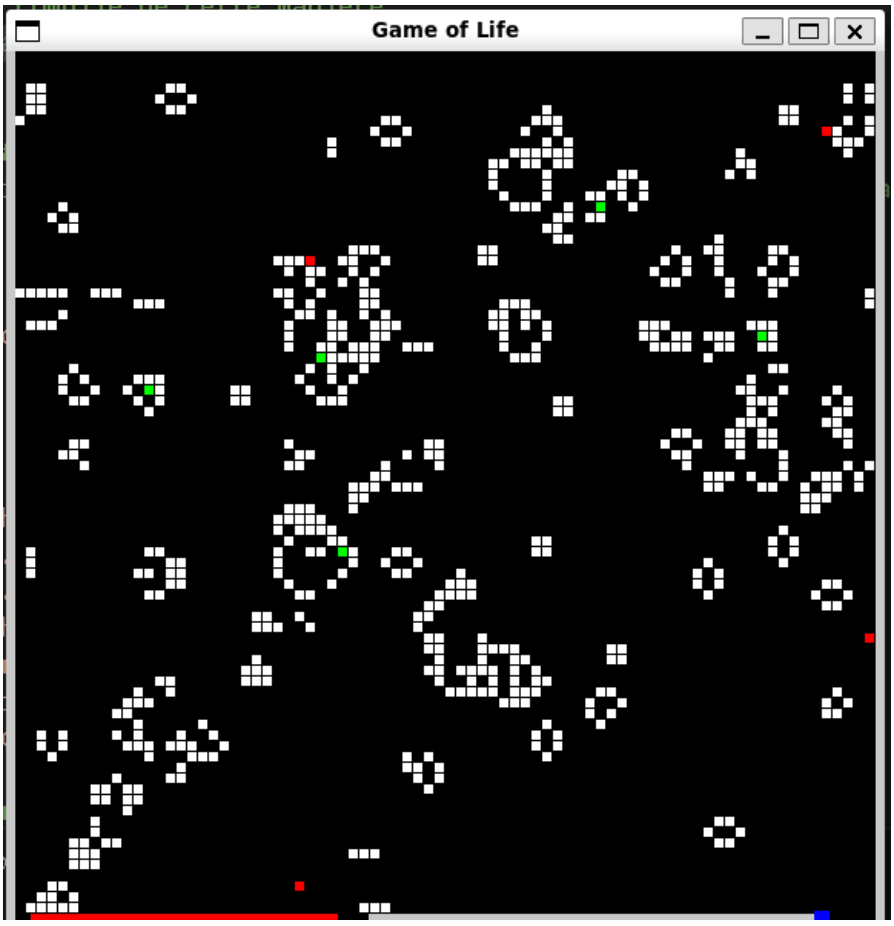
<u>Résultat obtenu :</u> une fenêtre affichant une grille aléatoirement se comportant normalement.



Une grille se comportant normalement ...



... itération ...



... après itération ...

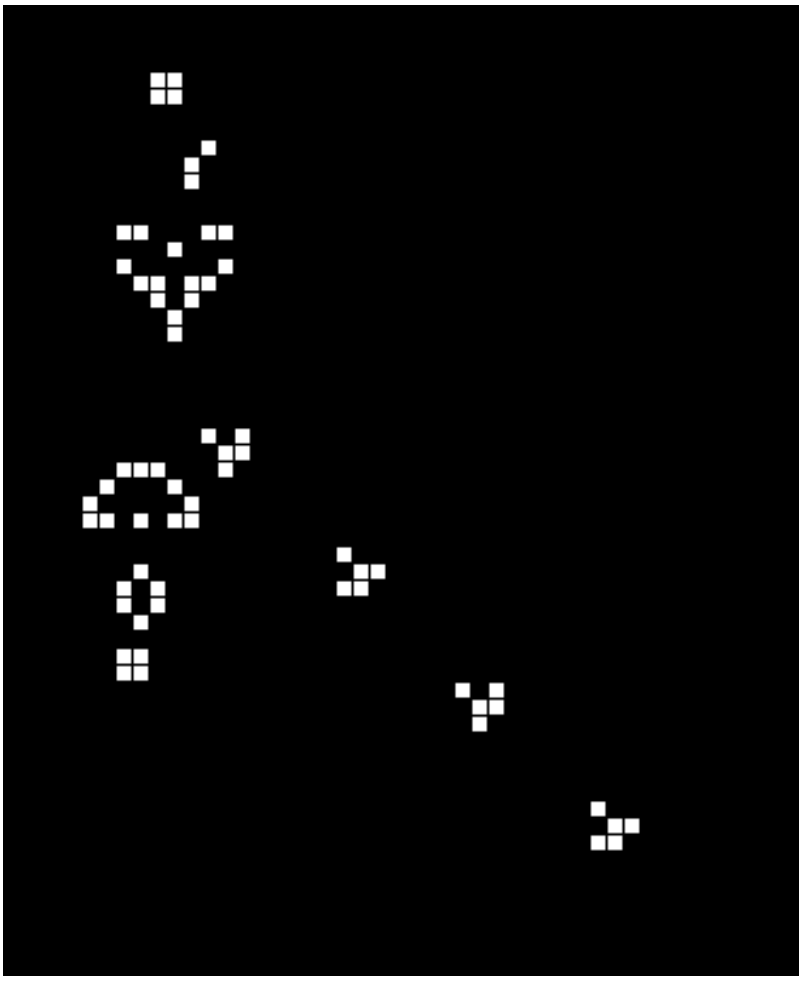
Test réussi !

Test réussi : 3/3

Initialisation de grille préprogrammés

Il est possible de créer une grille désirée depuis l'application. Il y a 3 modèles de base :

- Le planeur (nécessite une grille de 5x5 au minimum)
- Le générateur de planeur (génère un planeur toutes les 30 itérations) (nécessite une grille d'au moins 40x40 au minimum)
- Une grille aléatoire (peut contenir 1% de cellules obstacles si précisé)



Un générateur de planeur (gauche) générant des planeurs toutes les 30 itérations

3 tests unitaires ont été produits pour les initialisations de grilles :

1/ Création d'un planeur ("reset")

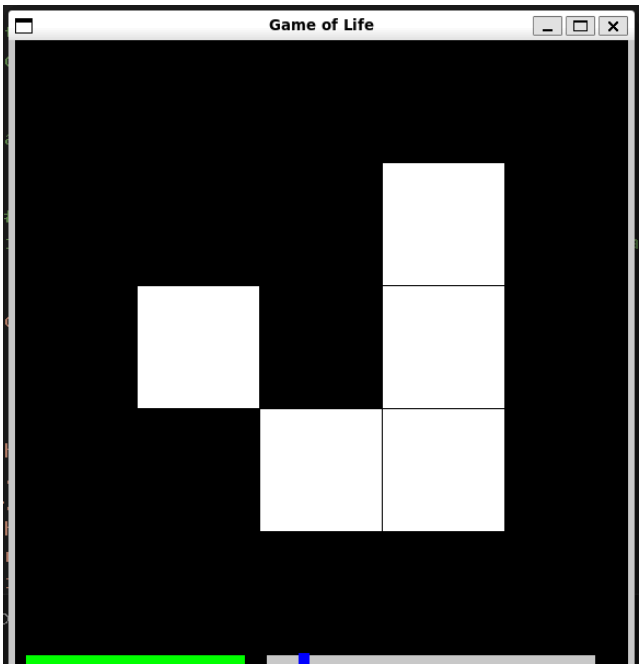
Explication : Génère une grille 5x5 contenant un planeur.

Résultat attendu : une fenêtre affichant une grille 5x5 avec un planeur

Résultat obtenu : une fenêtre affichant une grille 5x5 avec un planeur.

```
└─$ ./main reset test glider 5 5
Mode reset activé
Initialisation de la grille...
Grille générée et sauvegardée avec succès !
```

On génère la grille



Le planeur est bien affiché dans une grille de dimension 5x5

Test réussi !

2/ Création d'un générateur de planeur ("reset")

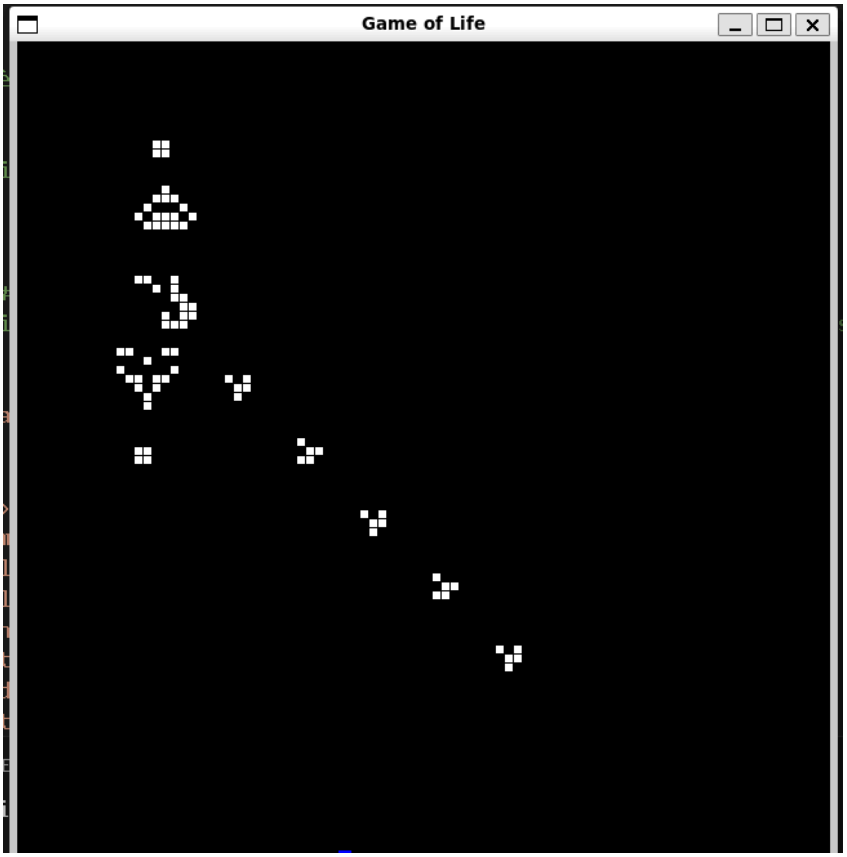
Explication : Génère une grille 90x90 contenant un générateur de planeur.

Résultat attendu : une fenêtre affichant une grille 90x90 contenant un générateur de planeur

<u>Résultat obtenu :</u> une fenêtre affichant une grille 90x90 contenant un générateur de planeur

```
└─$ ./main reset test canon 90 90
Mode reset activé
Initialisation de la grille...
Grille générée et sauvegardée avec succès !
```

On génère la grille



Nous avons bien un générateur de canon !

Test réussi !

3/ Création d'une grille aléatoire ("reset")

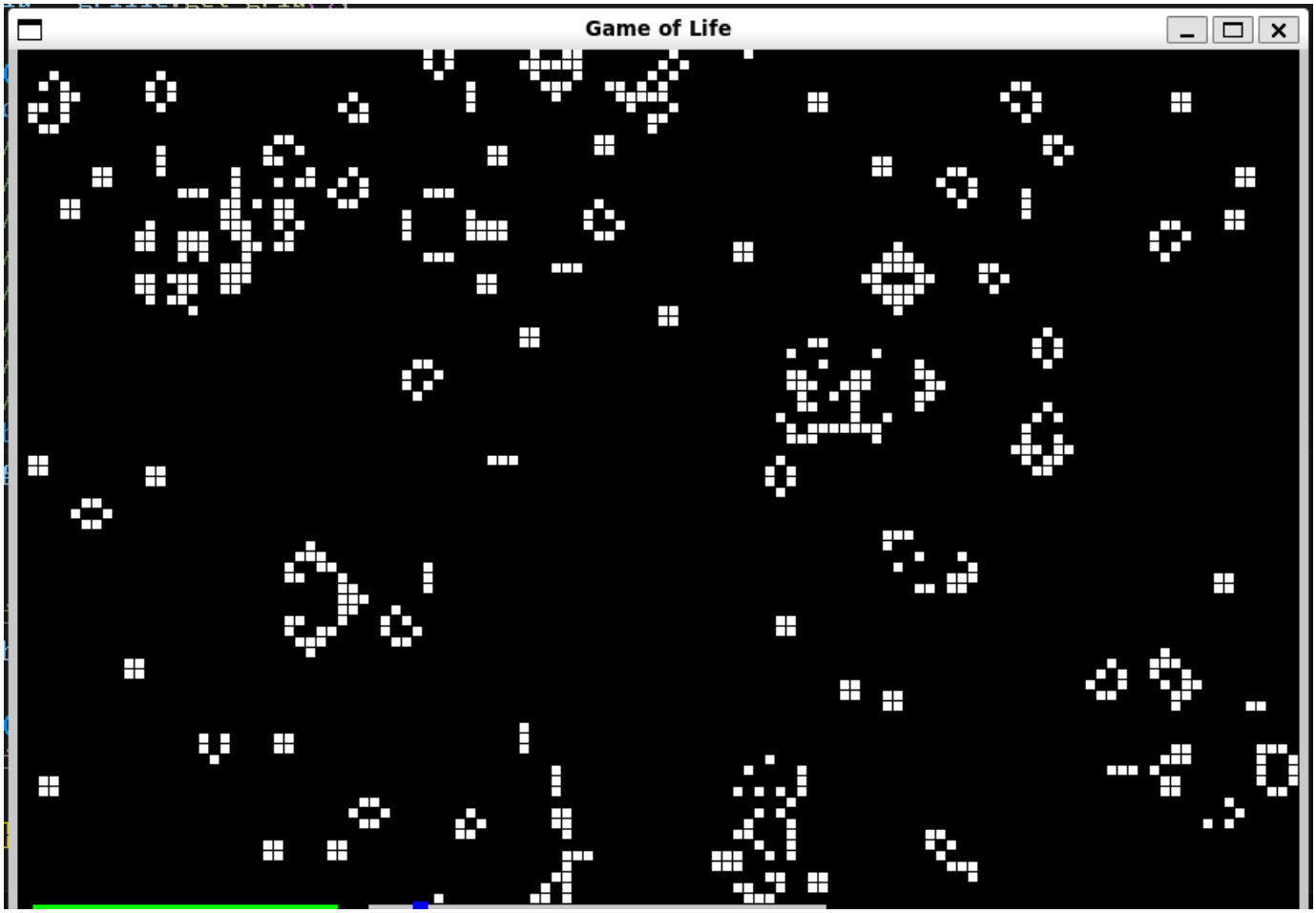
Explication : Génère une grille 120x80 aléatoire.

Résultat attendu : une fenêtre affichant une grille 120x80 aléatoire

Résultat obtenu : une fenêtre affichant une grille 120x80 aléatoire

```
└─$ ./main reset test aleatoire 120 80
Mode reset activé
Initialisation de la grille...
Nombre de cellules obstacles : 0
Grille générée et sauvegardée avec succès !
```

On génère la grille



Nous avons bien une grille aléatoire !

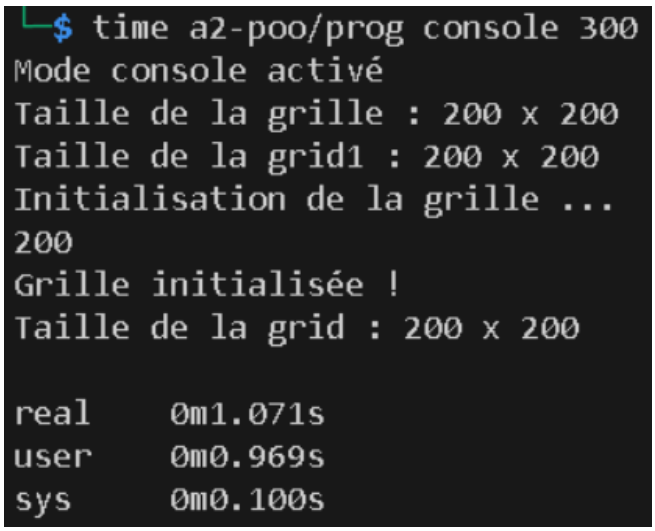
Test réussi !

Test réussi : 3/3

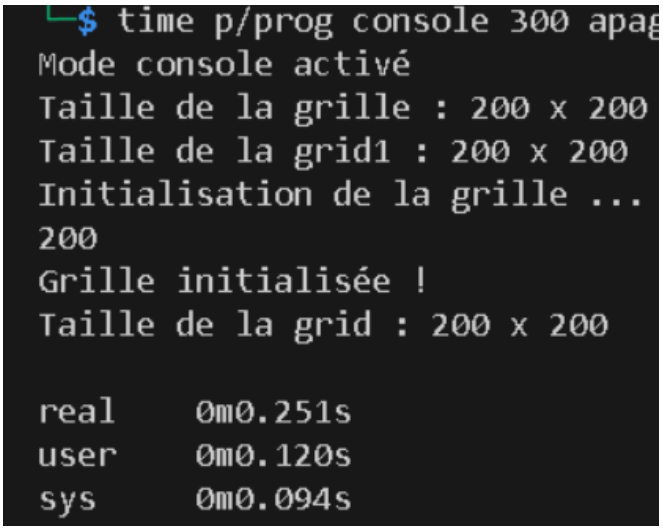
Parallélisation du traitement de la grille

Afin d'optimiser le code, l'utilisation des threads est grandement recommandée. Le multithreading permet, en quelques sortes, de pouvoir mettre à jour plusieurs parties de la grille en même temps, améliorant ainsi les performances globales du logiciel.

Des tests ont été faits : le premier sans mutlithreading, et le deuxième avec.



Sans multithreading



Avec multithreading

Nous constatons que le multithreading accélère les performances.