C++ / Évaluation « Jeu de la Vie »

Robotique/UE 3.1 – Janvier 2022

Supports de cours disponibles sur
`www.simon-rohou.fr/cours/c++`

A. Présentation

Le Jeu de la Vie est un automate cellulaire imaginé en 1970. Malgré son nom de « jeu », il ne repose pas sur l'intervention de joueurs mais se déroule de manière autonome. Des règles simples et pré-établies permettent l'évolution déterministe d'un monde représenté par une grille 2D. Le jeu consiste à définir un état initial du monde (voir par exemple la Fig. 1), puis à observer son évolution.

Le monde (le `World`) est une grille théoriquement infinie, mais sera dans notre cas bornée par les intervalles $[0, w] \times [0, h]$. Chaque case est une cellule (appelée `Cell`) prenant deux états : *vivante* ou *morte*. Chaque cellule est influencée par ses 8 voisines (horizontales, verticales et diagonales adjacentes). L'état des cellules voisines conditionne l'état de la cellule courante selon la règle :

- si la cellule courante est morte, et possède exactement trois cellules voisines vivantes, alors elle devient vivante ;
- si la cellule courante est vivante, et possède deux ou trois cellules voisines vivantes, elle reste vivante ;
- sinon, la cellule courante meurt.

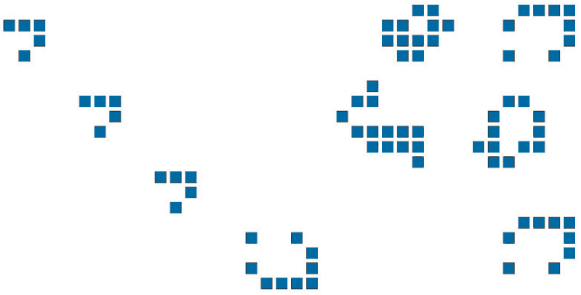

FIGURE 1 – État initial d'un monde composé de plusieurs vaisseaux spatiaux (des `spaceships`), qui sont des amas de cellules particuliers se déplaçant sans disparaître en cours de jeu. Les cellules bleues sont dites *vivantes*, les blanches sont *mortes*. Cet état initial laisse présager une situation tendue à l'aube d'une guerre imminente.

Règles

L'état d'une cellule peut être défini plus simplement par :

$$(S = 3) \text{ OU } (E = 1 \text{ ET } S = 2) \tag{1}$$

avec :

- S : nombre de cellules vivantes dans le voisinage de la cellule courante ($S \in \{0, \dots, 8\}$) ;
- E : état précédent de la cellule ($E = \{0, 1\}$, 0 pour une cellule morte, 1 pour une cellule vivante).

Critères d'évaluation

- Développement des questions
- Organisation propre des sources (indentation, nom des variables, fonctions)
- Commentaires et lisibilité du code
- Respect de la *Const Correctness*
- Pas de fuite mémoire
- *Malus :* rendu de votre exécutable binaire ou de fichiers temporaires ou compilés (`.o`, *etc.*)

Lisez attentivement les questions suivantes, et implémentez-les dans l'ordre, en compilant votre programme systématiquement.

B. Commentaires

1. Toutes les fonctionnalités développées ci-après devront être commentées.

C. Une cellule..

2. Télécharger les fichiers `Cell.h`, `World.h`, `vibes.cpp/h` sur le site web du cours.
   La classe `Cell` est pré-définie avec :
   - un constructeur par défaut.
   - un constructeur paramétré créant une cellule en (x, y).
   - une méthode `display()` affichant une cellule avec VIBes.
   - une méthode `set_alive(..)` donnant vie à une cellule, ou la faisant mourir.
   - une méthode `alive()` retournant `true` si une cellule est vivante.
3. Compléter les définitions des méthodes dans `Cell.h` afin de respecter la *const correctness*.
4. Donner une valeur par défaut à l'argument de la méthode `set_alive(..)`.
5. Définir une énumération [1] `CellState` permettant de donner à un *état* de cellule les valeurs suivantes : `alive`, `newly_alive`, `dead`, `newly_dead`.
6. Proposer les membres de classe de `Cell`, renseignant la position et l'état de la cellule (morte par défaut).
7. Définir les droits d'accès des méthodes et des attributs de `Cell`.
8. Protéger le fichier `Cell.h` contre l'inclusion cyclique.

D. ..et son implémentation

Créer et implémenter `Cell.cpp`.

9. La méthode `display()` affichera la cellule sous la forme d'une boîte carrée ($[x, x+1] \times [y, y+1]$).
   La couleur de la cellule est liée à son état :
   - `alive` : couleur `blue`
   - `newly_alive` : couleur `green`
   - `dead` : couleur `white`
   - `newly_dead` : couleur `red`
10. La méthode `set_alive(..)` change l'état `CellState` de la cellule. Les états `newly_alive` et `newly_dead` correspondent respectivement à un changement d'état lorsque la cellule était auparavant `dead` et `alive`. À l'inverse, une cellule restant vivante (resp. morte) passe de `newly_alive` à `alive` (resp., de `newly_dead` à `dead`). On pourra par exemple utiliser un opérateur ternaire pour obtenir des affectations élégantes.
11. La méthode `alive()` retourne vrai si la cellule est dans l'état `alive` ou `newly_alive`.

1. Un petit rappel se trouve à la slide 4 du cours 3 (pointeurs).

E. Le monde..

12. Compléter les définitions des méthodes dans `World.h` afin de respecter la *const correctness*.
13. Définir les droits d'accès des méthodes et des attributs de `World`.
14. Protéger le fichier `World.h` contre l'inclusion cyclique.
15. Ajouter un attribut de classe représentant la grille 2D de cellules du monde. Quelque soit la représentation choisie pour cette grille, on veillera à :
    — ne pas avoir de fuites mémoires en fin de programme ;
    — ne pas avoir de représentation redondante des informations.

F. ..et son implémentation

Créer et implémenter `World.cpp`.

16. Le constructeur initialise la grille de $w \times h$ objets `Cell` en les positionnant aux bons endroits. Il initialise également VIBes avec :

```
vibes::beginDrawing();
vibes::newFigure("World");
vibes::setFigureProperties("World",
    vibesParams("x", 100, "y", 100, "width", 1000, "height", 500));
vibes::axisLimits(0, m_width, 0, m_height);
```

17. Le destructeur quitte proprement la vue graphique.
18. La méthode `set_alive(..)` rend vivantes les cellules positionnées aux coordonnées renseignées en argument.
19. La méthode `display(..)` met à jour l'affichage du monde en ré-affichant les cellules.
20. La méthode `translate_coords(..)` translate les coordonnées contenues dans `coords` de (dx, dy). L'argument `coords` est donné en lecture/écriture.
21. Dans un programme principal, créer et afficher un monde. On prendra $w = 80$ et $h = 40$.

G. Le jeu

Le jeu consiste à simuler l'évolution du monde.

22. Ajouter une procédure `live()` en méthode de la classe `World`. Cette procédure calcule les nouveaux états des cellules du monde, à partir de l'état précédent du monde. Chaque cellule évolue selon la relation (1).
23. Dans le programme principal, simuler le monde pour un temps discret entier allant de 0 à 1000. Pour obtenir une visualisation fluide à l'écran, on utilisera une pause en fin de chaque itération avec :

```
usleep(50000.); // animation speed
```

H. Les armes

À ce stade, la quiétude est reine dans ce monde vide, calme et silencieux.

24. Un *spaceship* (un vaisseau) est un amas de cellules se déplaçant sans faire de débris, à moins d'entrer en collision.
    Dans `World`, ajouter une méthode, `void create_spaceship(int x, int y)`, créant un vaisseau aux positions (x, y). Le motif d'un vaisseau positionné en $(0, 0)$ est donné par :

    ```
    {-1,0}, {-2,1}, {-0,2}, {-1,2}, {-2,2}
    ```

25. Faire surgir un vaisseau en $(40, 20)$ à $t = 0$.

Le vaisseau est en fait un éclaireur d'une Alliance rebelle dans un monde tourmenté. Il vient confirmer les craintes de l'Alliance : la mise au point par l'Empire d'un canon d'une force stratégique considérable.

Les canons (ou lanceurs, ou encore lances-navires) sont en quelque sorte des oscillateurs lâchant des débris de manière régulière. Ces débris sont considérés comme des vaisseaux, dans le sens où ils résultent en un amas de cellules qui se déplacent.

26. Dans `World`, ajouter une méthode `void create_cannon(int x, int y)` créant un canon aux positions (x, y). Le motif d'un canon positionné en $(0, 0)$ est donné par :

    ```
    { 1,5},{ 2,5},{ 1,6},{ 2,6},
    {35,3},{36,3},{35,4},{36,4},
    {13,3},{14,3},{12,4},{16,4},
    {11,5},{17,5},{11,6},{15,6},
    {17,6},{18,6},{11,7},{17,7},
    {12,8},{16,8},{13,9},{14,9},
    {25,1},{23,2},{25,2},{21,3},
    {22,3},{21,4},{22,4},{21,5},
    {22,5},{23,6},{25,6},{25,7},
    ```

27. Créer, dans l'état initial du monde, un canon positionné en $(0, 0)$. Observer ses projectiles.

Face à cette menace inédite, l'Alliance envisage l'impensable : une « mission opérationnelle suicide », dans laquelle un second vaisseau viendra percuter l'un des projectiles du canon. Les généraux comptent sur une réaction en chaîne des explosions, menant à la destruction de l'arme suprême.

28. Faire surgir un vaisseau en $(60, 2)$ à $t = 30$.