Bien commencer son projet : le *jeu de la vie*

12 décembre 2012

1 Introduction

Bien démarrer son projet informatique n'est pas évident lorsque l'on a jamais fait de gestion de projet. Souvent, l'on se lance tête baissée dans le projet, et on se casse les dents car lorsqu'il a arrive le moment où il faut se réorienter, il faut retaper presque entièrement le code car on ne l'a pas pensé de manière suffisamment modulaire et fragmenté.
L'erreur que tout le monde fait au début du projet est de ne pas suffisamment penser à la forme de son projet avant de commencer à coder. Le but de ce document est d'expliquer une manière de comment penser son projet du *jeu de la vie*. Au début, la chose la plus importante à faire est de se réunir autour d'un verre (d'eau !) avec du papier et un crayon, et de discuter de comment on va s'y prendre. On appelle ça établir un cahier des charges.
Il est important de lister tout le travail à faire, de discuter point par point de la bonne manière de représenter les données et les manipuler, puis d'imaginer de loin ce que l'algo devra faire.
L'auteur a lui-même participé à des projet qui ne sont pas passé par cette étape et qui se sont, de manière évidente, terminés en eau de boudin.
Le but est de vous éviter ce désagrément en vous donnant le squelette du projet.

2 La base : le jeu vidéo

Tous les jeux–vidéo sont conçus de la même manière, en quatre temps :

1. l'initialisation : cette étape sert à initialiser toutes les variables du jeu. Nous verrons lesquelles en ce qui nous concerne,
2. le calcul du jeu : à cette étape, on calcule, à l'instant t, l'état dans lequel sera la jeu à l'instant suivant,
3. le dessin du jeu : à ce moment, on dessine le jeu sur la fenêtre, cela sera fait apparemment avec la bibliothèque *ncurses*,
4. la destruction : elle intervient à la fin du jeu pour libérer la mémoire qui lui a été allouée afin de la rendre au système d'exploitation.

Le première et la dernière étape ne se font qu'une fois par lancement de jeu, tandis que les deux du centres sont prises dans une boucle pseudo–infinie, en effet, tant que le jeu est lancé, on continue de le faire évoluer et de la dessiner.

Ça, c'est la théorie. Notre projet étant un poil plus simple qu'un *Call of Duty*, il sera un peu différent.
Primo, l'étape de la destruction n'interviendra pas. C'est une étape qui est nécessaire lorsque l'on utilise l'allocation dynamique de la mémoire, chose que nous n'avons pas vue.
Secundo, l'étape de calcule et de dessin du jeu peuvent être faits en même temps dans un soucis d'optimisation (nous y reviendrons) mais il est préférable pour les débutants de le faire séparément.

Présenter le projet comme fait précédemment fait apparaitre une première manière de se répartir les tâches : une personne étudie la manière d'afficher le jeu, tandis que l'autre s'occupe des fonctions qui vont calculer le jeu de la vie. Nous reviendrons sur la répartition des tâches.

3 Le jeu de la vie de CONWAY

3.1 Le plateau de jeu

Mais pour pouvoir dessiner et calculer quelque chose, encore faut-il s'être mis d'accord sur ce qu'il faut dessiner et calculer. Pour ceux qui on déjà jeté un œil à la page wiki consacrée, vous aurez remarqué que le jeu de la vie est représenté graphiquement sous la forme d'un quadrillage, chaque case contenant une cellule.
Une manière intuitive de représenter les choses est donc de le faire sous la forme d'un tableau à double entrée, ce qui a ça de bien qu'il est facile à manipuler et à imaginer. Les booléens n'existant pas en *C*, on les simule par des entiers. Ainsi, une cellule morte pourrait être représentée par 0 et une vivante par 1.
Bien sûr, ce n'est pas la seule manière de représenter les choses, et certains téméraires pourrons s'aventurer vers des structures de données plus complexes comme les listes chainées, voire un type dérivé du graphe, mais mieux vaut que vous vous laissiez la surprise de la découvrir en Algo avancé au deuxième semestre.

3.2 Les règles

Les règles de l'automate de Conway sont fort simples :
- une cellule est soit morte, soit vivante,
- une cellule morte entourée par exactement trois voisines naît,
- une cellule vivante entourée par deux ou trois voisines, reste vivante,
- une cellule entourée de moins de deux voisines ou plus de trois, meurt ou reste morte.

Il nous faut donc calculer le nombre de voisines et, sur cette base, déterminer si la cellule doit rester dans son état, ou passer dans l'état suivant.

3.3 Pré-calculer le tableau

Un problème qui va rapidement se poser dans le calcul du plateau de jeu est l'influence du changement d'état d'une cellule sur celui de ses voisines suivantes. L'état de toutes les cellules ne pouvant pas être changé en une seule opération, il faut empêcher qu'une cellule en mourant ou en naissant, empêche sa voisine de mourir ou naitre alors qu'elle l'aurait dû. Il faut donc pré-calculer l'état de toutes les cellules avant de 'appliquer les changements.

4 Commencer à coder

4.1 La répartition des tâches

Comme nous l'avons vu plus haut, une manière intuitive de répartir les tâches est de dire qu'une personne va s'occuper d'afficher le jeu, et l'autre, de le calculer. Une fois qu'on s'est mis d'accord sur quoi calculer et quoi afficher, les deux branches peuvent se développer de manière totalement indépendante. Si l'on a fait le choix de travailler sur un plateau représenté par un tableau à double entrée il suffit décrire un jeu de fonctions pour calculer le tableau et un jeu pour l'afficher.

Encore une fois, cette manière de voir les choses, bien que la plus intuitive, n'est absolument pas la seule, et chacun voit midi à sa porte. Le tout est de se partager le travail.
En divisant le travail, on devient plus efficace(concept de *crowdsourcing*), et on améliore la maintenance du code source (un concept similaire à celui de la division du travail chez Ford).

4.2 La division du travail

La division du travail consiste à laisser une entité ne faire qu'une tâche bien précise. Cela permet à l'entité de ne s'occuper d'une seule chose et donc, de pouvoir répondre de la manière la plus efficace possible au problème posé. Par exemple, un ouvrier qui visse une vis cruciforme et une vis plate sur une automobile, devra changer sans arrêt d'outil. Cela induit un effort physique et intellectuel plus important que si un ouvrier s'occupait d'une seule vis. L'exemple est un peu caricatural, mais l'idée est là. De la même manière, en informatique, il est plus facile de répondre à un problème simple que compliqué. L'idée est donc de diviser le problème compliqué en deux problèmes plus simples, eux mêmes divisés en deux problèmes plus simples, eux mêmes... (Récursivité !).
Nous avons déjà pu voir une première décomposition du problème entre calcul et affichage, mais la division du travail doit s'effectuer non seulement entre chaque membre du groupe, mais également dans le code source.
Dans le calcul du plateau de jeu, par exemple, il faut parcourir le tableau pour déterminer le nombre de voisines de chaque cellule pour déterminer si elle vit ou meurt. On peut donc imaginer une fonction qui parcourrait le tableau, une qui déterminerai le nombre de voisines, et une qui calculerait sont état suivant.
L'important étant toujours qu'une fonction ne soit trop longue et ne fasse des choses trop compliquées. La fonction `main()` ne doit comporter que très peu de lignes et doit se contenter d'appeler des fonctions plus spécialisés qui elles même appellent des fonctions plus spécialisées, etc... L'idéal est que le `main()` ne comporte que la déclaration des variables nécessaires au jeu et l'appel des trois fonctions citées au début du paragraphe.

5 Optimiser son code

Une fois que l'on a un jeu de la vie fonctionnel, et seulement à cette condition, on peut commencer à optimiser l'algo.
Je reprends le bout que j'avais laissé en l'air tout à l'heure : le calcul et l'affichage en même temps. En effet, le calcul et l'affichage du plateau de jeu nécessitent tous les deux un parcours entier du tableau, ce qui n'est pas un problème pour un tableau de 10×10 mais l'est déjà plus pour un tableau 100×100 et évidemment plus encore pour un tableau de 1000×1000. Il est donc temps de se demander comment calculer et afficher le tableau d'une traite. En suite, si l'on a encore du temps, on pourrait se demander comment confondre l'étape de pré-calcul et d'affichage, et ensuite, comment prédire comment vont évoluer des groupes entiers de cellules, etc... J'insiste sur ce fait : l'optimisation doit venir qu'une fois que le projet fonctionne.

6 Rédiger le rapport et le présenter

Rédiger le rapport et répéter la présentation sont souvent des étapes négligées et faites à la vavite, et c'est une grave erreur car la présentation du projet compte souvent autant voire plus dans une note que le code. En effet, il n'est pas spécialement grave de n'avoir pas atteint l'objectif, si l'on est capable de montrer qu'il y a eu un travail derrière. C'est souvent à la présentation que ce travail est mis le plus en valeur. La présentation et le rapport on deux visées différentes : la première sert à

entrainer à la présentation d'un produit, chose que tous les développeurs ont à faire pour le vendre au client. Le rapport, quant à lui, vise surtout à rendre compte de l'avancement du projet :

- Comment on a réfléchit le projet ?
- Comment on a implémenté la chose ?
- Quelles difficultés l'on a rencontré ?
- Quelles solutions on pu être apportées ou auraient pu ?
- Comment le projet pourrait évoluer ?
- Comment on s'y prendrait pour implémenter cette évolution ?
- Etc...

La présentation, elle, doit permettre une démonstration du projet, et éventuellement du code source. Il peut reprendre certaines questions (quels problèmes rencontrés ? Comment les solutionner ?) du rapport, mais se présente de manière moins chronologique. Il est **très** important de répéter une présentation (j'ai cru comprendre que beaucoup avaient compris) et surtout de connaitre par cœur le déroulement que celui-ci va prendre. Il n'est pas question d'apprendre son texte par cœur mais de connaitre son plan, les idées et leur enchainement. Les bugs doivent êtres cachés, et pour cela, repérés lors d'une répétition.
Privilégiez des configurations, des réglages dont vous êtes sûrs qu'ils fonctionnent, et si un bug est inévitable, avancez une explication, de préférence pas trop trop capilo-tractée.
Une dernière chose : prévoyez toujours deux PC, la loi de Murphy n'est pas une légende.

Cette aide ne se prétend pas exhaustive et ne vise qu'à vous donner des conseils, donc par nature, toujours un peu subjectifs. Cependant, ce sont les mêmes que je me suis vus prodigués au début, et j'ai pu faire l'amère expérience que ne pas les suivre pouvaient mettre bien dans la mouise. Sur ce, il ne me reste plus qu'à vous souhaiter bonne chance.