Mayira.Y – Arnaud.L
S2B" – IUT Bordeaux 1 – avril/mai 2011

Jeu De La Vie

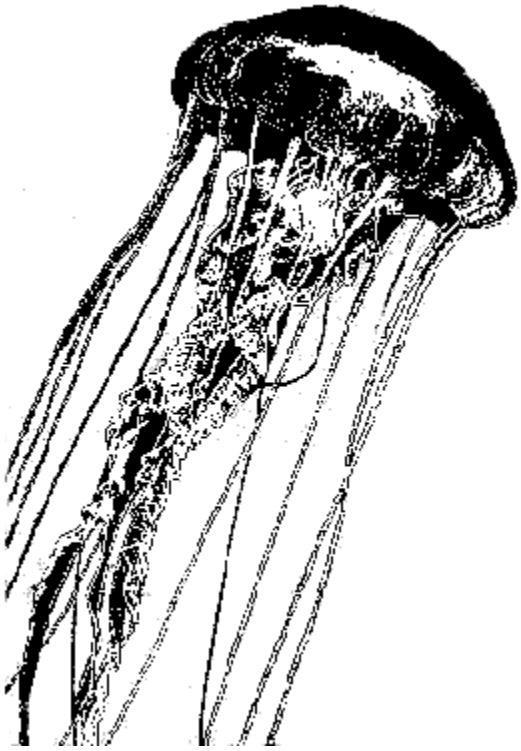


Table des matières

1 principe du Jeu De La Vie :

1.1 Déroulement du Jeu :

Le Jeu De La Vie reprend le thème des océans en mettant en scène les méduses apparentées a des Cellules qui effectuent chacune leur cycle de vie selon trois règles :

- Si la méduse est vivant et qu'elle possédée au minimum deux voisines vivantes et au maximum trois voisines vivant, alors cette méduse reste vivant au prochaine cycle.

- Si la méduse est morte et qu'elle possédée au minimum trois voisines vivantes et au maximum trois voisines vivant, alors cette méduse revivra au prochaine cycle.

- Dans les autres cas, la méduse meurt au cycle suivant.

Les méduses du milieu peuvent avoir une santé normal, mais elles peuvent être aussi résistante aux oursins qui perturbent leur activité. Une fois que ces derniers ont infecté l'une d'entre elle, ils effectuent leur temps d'incubation puis la tue en y laissant des enfants oursins derrière eux. Bien sure ces parasites ne vivent pas indéfiniment dans les eaux car si ils n'atteignent pas assez vite leur proie, ils se désintègrent complètement

1.2 Notice d'utilisation :

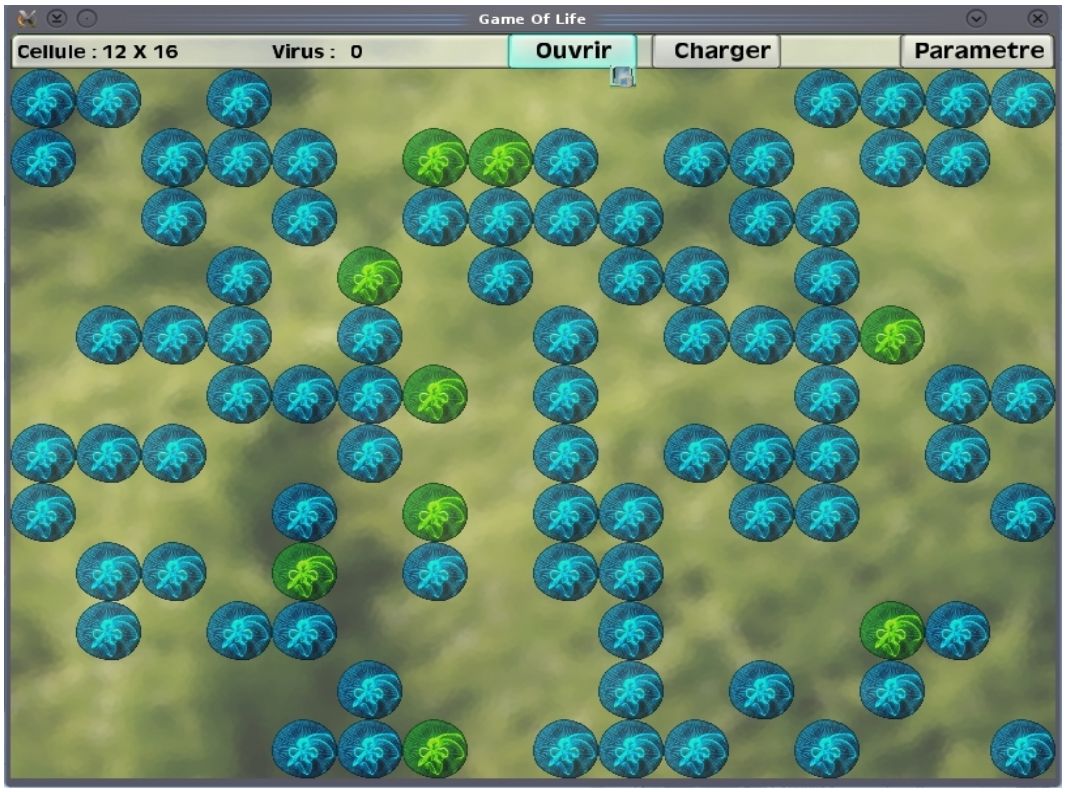
Après avoir téléchargé le fichier compressé JeuDeLaVie.zip, vous trouverez à la racine du dossier « JeuDeLaVie » un LISEZ-MOI.txt qui explique comment compiler correctement le jeu et les types d'erreurs que vous risquerez de croisé durant l'installation.

Au lancement, une courte introduction de 17 seconde apparaîtra, vous pourrez toujours la passer en cliquant sur le bouton « passer l'intro », pressant la touche ECHAP ou en quittant la fenêtre. Ensuite vient le jeu, composé de deux fenêtres :

- Une fenêtre de jeu : contenant la grille de méduse (qui joueront le rôle des Cellules), les oursins (qui joueront le rôle des Virus) et des informations concernant la taille de la grille de cellule et le nombre de Virus mis en jeu.

- Une fenêtre de menu : regroupant l'ensemble des buttons et sliders permettant de changer les différents paramètres du jeu.

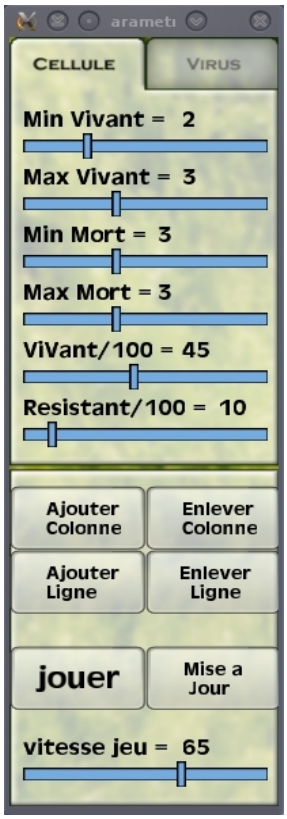
1.2.1 Fenêtre de jeu :



Au commencement, la Grille est initialisé à 16x12 de Cellules avec 45% vivants et 10% résistantes définit aléatoirement. Le jeu est mis sur pause de base, vous pourrez le voir sur la fenêtre de menu avec un bouton qui indique « jouer » lorsque la simulation est arrêté et indique « pause » quand la simulation est en court. Vous pourrez lancer ou arrêter le cours du jeu en cliquant sur ce bouton (jouer/pause) ou en pressant SPACEBAR dans la fenêtre de jeu. Pour rendre une Cellule vivante, faite un clic gauche sur grille du jeu, la Cellule sélectionnée par le pointeur de la sourie changera d'état pour devenir vivante, et inversement, par un clic droit, la Cellule sélectionnée par le pointeur de la sourie deviendra morte. Vous pourrez fermer la fenêtre de menu pour avoir un Bureau moins encombré de fenêtre, et a tout moment vous pourrez rouvrir le menu en cliquant sur le bouton Menu « Parametre ».

Deux boutons permettent de sauvegarder ou de charger une seule simulation : « ouvrir » et « charger ». En cliquant sur charger seul la grille de cellule, le groupe de virus et l'etat du jeu (pause ou joueur) sont modifiés. Si vous charger la sauvegarde au debut du jeu, vous verez une simulation deja préparé assez sympatique. Si vous avez sauvegardez avant, vous pouvez récupérer cette fameuse sauvegarde en changant le fichier « kill.sav » en « game.sav » dans le dossier « save/ ».

1.2.2 Fenêtre de menu :



Le menu de paramétrage du jeu est organisé en deux onglets :

- Cellule :

4 sliders permettent de définir le minimum et maximum de voisines vivantes pour qu'une Cellule reste en vie ou devienne vivante. Les 2 Sliders suivants règlent le pourcentage de Cellules Vivantes et de cellules résistantes au prochain ajout le ligne et de colonne : par exemple si on met ces deux sliders a 100%, les prochaines lignes ou colonnes de Cellules ajoutés seront toutes vivantes et toutes résistantes.

- Virus :

3 boutons pour gérer l'intégration des Virus dans le jeu, en ajoutant un, en enlevant un, ou en les supprimant tous. 3 sliders modifient le nombre d'enfant à la descendance, le temps d'incubation et le temps de vie pour chaque cellule.

commun à ces deux onglets, 4 sliders gèrent l'ajout et la suppression de lignes et de colonnes de Cellule. Vient ensuite deux boutons, un boutons précédemment présenté pour jouer ou mettre sur pause, et un bouton pour mettre a jour la grille de Cellule. Petite précision importante : la mise a jour s'effectue seulement si il y a eu une modification dans le pourcentage des vivants ou le pourcentage des résistants, ainsi on pourra facilement réinitialiser la grille a zéro vivant par exemple. Un dernier slider définit simplement le pourcentage de vitesse de jeu.

2 Développement & Architecture :

2.1 Architecture du programme :

L'architecture est basé sur la méthode de conception du Model View:

- Model : il représente le comportement du programme c'est a dire les diverses traitements des données.
- View : elle correspond à l'interface avec laquelle l'utilisateur va pouvoir interagit. Dans un premier temps elle affiche les résultats renvoyés par le modèle (fonction Draw), puis elle recevoit toutes les actions de l'utilisateur (fonction treatEvent : clic de souris, sélection d'une entrée, boutons, etc).

2.1.1 Le Model :

Appelé GameModel, il est composé de deux données essentielles : GrilleCellule et de GroupeVirus, mais aussi il est composé d'une GameSave.

GrilleCellule est dépendant de Cellule et GroupeVirusest dépendant de Virus.

Virus et Cellule héritent tous deux de la classe Unite car elles ont en commun des coordonnées dans un plan.

2.1.2 Les Views:

Dans le Programme, il y a deux views : MenuView et GameView, deux classe qui hérite de Element du fait que une fenetre a une position et une taille. Ces deux classes contiennent (agrégation) la classe GameModel pour avoir un lien avec les données. Elles sont composées aussi de Button, mais seul MenuView est composée de Slider. Button et Slider héritent de Element car ils ont des coordonnées et une taille spécifique dans les fenétres.

2.2 Représentation UML :

2.3 Rôle des objets :

GameModel	C'est le model du jeu, c'est a dire c'est lui qui definit la prochaine étape pour les objets du jeu, et assure l'action entre l'utilisateur et le jeu. Évidement il contient les deux regroupement : GrilleCellule et GroupeVirus
Unite	Classe mère qui contient des coordonnées (_x, y) dont les classes filles seront

	des des acteurs du jeu.
Cellule	représente les unité principal du jeu, cette classe definit les états et santé d'une cellule
GrilleCellule	est un regroupement de Cellules, en effet cette classe à un vector< vector<Cellule> > (une matrice dynamique). Cette classe contient deux réels qui définissent la taille (_wC et _hC) pour chaque Cellule de la matrice, 4 entiers pour les minimum et maximum de Cellule voisines nécessaire pour appliqué la loi de vie, ainsi que 4 réel pour définir le pourcentage aléatoire l'état et la santé des nouvelles cellules. Un point important, la matrice est entiérement rempli de cellule, seule leur état les fera apparaître ou disparaître. Si une cellule précise est resistante, elle le restera jusqu'à que le joueur change le pourcentage de resistante.
Virus	représente l'unité pertubateur du jeu, un booléen indique si le virus a infecté une Cellule, deux réels qui deffinissent sa direction car c'est la seul unité qui se déplace librement,
GroupeVirus	est un regroupement de Virus, en effet cette classe possède un vector<Virus> (un tableau dynamique). Elle contient deux réels qui définissent la taille (_wVirus et _hVirus) pour chaque Virus du jeu, un entier qui indique le nombre d'enfant a la descendance et deux réels pour la temps de vie et d'incubation.
GameSave	Cette classe gère la gestion d'une sauvegardement d'une simulation en cours et le chargement de la sauvegarde précédente simplement
GameView	affiche la fenetre du jeu avec un RenderWindow. Un pointeur vers le le GameModel permet d'associé les action de l'utilisateur une méthode précis. Un pointeur ver le MenuView permet seulement savoir si la fenetre de menu est ouverte ou fermée afin de de l'ouvrir seulement si elle a été fermé. Deux rééls permettent de récupérer les coordonnées de la sourie durant le traitement d'évenement.
MenuView	affiche la fenetre du menu avec un RenderWindow. Un pointeur vers le le GameModel permet d'associé les action de l'utilisateur une méthode précis. Il posséde des boutons et reglettes pour faire les différents réglages du jeu.
Element	Classe mère qui contient des coordonnées (_x, y) et une taille (_w, _h), dont les classes filles seront des objets graphiques du jeu.
Button	représente l'objet bouton du jeu, il posséde un Sprite pour contenir l'image du bouton et un String qui est le texte apparaissant sur le bouton. Il permet d'effectué une action par un click relaché du joueur.
Slider	Représente l'objet reglette du jeu, il posséde deux Sprite pour l'image de font et l'image de la partie mobile. Cela permet d'effectué un reglage précis d'une variable entre une valeur min et une valeur max.

D'autre fonctions du jeu :

Intro	Une classe qui permet de mettre en place l'introduction du jeu. En effet en structure est elle pratiquement identique à la classe GameView mis a part qu'elle n'a pas besoin d'accédé a une structure de donnée spécifique. Elle est simplement déclaré en premier puis aussitôt désalloué dans le main.
util	Regroupe quelque petite fonction utile dans le programme, notament des retours de valeurs aléatoires, et des conversion d'entier en chaine de caractère
images	Regroupe des fonction qui permet de simplifier le chargement, la modification de Sprites et le chargement d'un String. Il y a aussi des fonctions qui permettent d'affiche les objets du jeu.
event	Regroupe des fonctions qui permet de simplifier les gestion d'évenements comme les différents type de clic de la sourie ou les touche préssé du clavier. Il permet de géré aussi la gestion d'action sur les boutons et les reglettes du jeu.
enum	Regroupe les trois enum utilisé le long du jeu : l'etat d'une cellule, la santé d'une cellule, et le type d'onglet

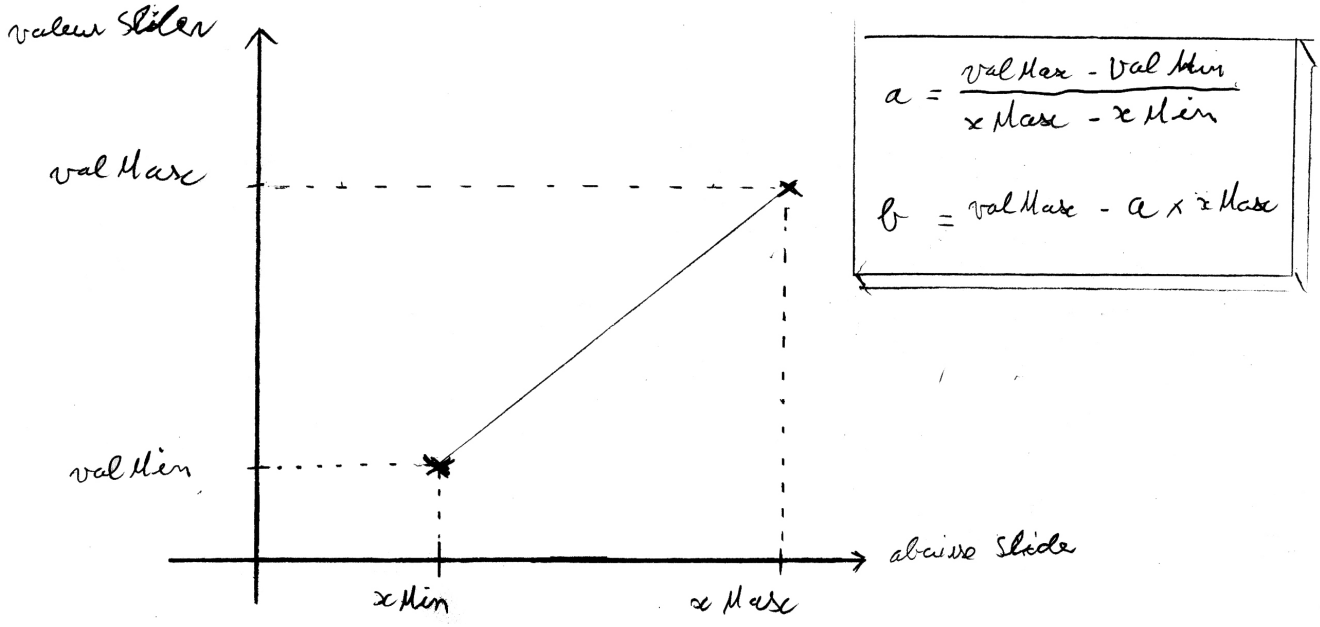
3 Explications algorithmiques & défauts :

3.1 Algorithmes intéressants et délicats :

3.1.1 Le Mouvement des Sliders :

Un premier algorithme intéressant à expliquer est celui de l'action des Sliders qui se fait en plusieur étape. Pour cela on fait appel à 3 fonction. La premiere est moveSlider qui se situe dans le fichier event.cc, il gére l'action du slider en fonction de la sourie : lors d'un simple clic sur la partie mobile du slider, on met _move un booléen contenue dans la classe Slider. Toujours dans cette fonction lorsque ce booléen est a vrai, on replace la partie mobile a la position de la sourie (seulement sur l'axe des abcisses) pour on fait appel a la fonction updateSlider. UpdateSlider met simplement a jour la position de la partie mobile en vérifiant d'abord si il depasse pas les limite fixé, et met a jour l'atribut _valeur avec un calcule de la position du slider : il s'agit en fait d'une fonction affine qui correspond a la valeur que retourne le slider en fonction de la position en abscisse de la partie mobile. Donc en fixant une valeur max et min, et un xmin et un xmin, on récupére la fonction de la forme « ax + b » ou « a » est appelé _ord et « b » est appelé _coef dans la classe Slider

Voici un schéma montrant comment on interpréte graphiquement le calcule de la valeur du slider pour une position..



Ensuite un simple accesseur en lecture sur cette valeur va permetttre de la récupéré pour mettre a jours les objets qui en sont dépendant.
Par contre si on clic sur le Slider mais hors de la partie mobile, on fait direct appel updateSlider.

3.1.2 L'alignement des Virus :

Lors qu'on ajoute où enléve ou retires des Cellules, il est très important que les Virus reste aligné sur la grille virtuelle que occupent les Cellules. Il est sure que cette algorithme nous a posé problème car on etait partie sur une solution un peu compliqué. En fait on voulait pour une Cellule i connaître combien il fallait enlever ou ajouter à ses coordonnées, en récupérant les coordonnées de la cellule dont il est le plus proche, et en calculant la diffénce entre avant et aprés la modification de la matrice. Le probléme avec cette algorithme c'est que cela ne fonctionnait pas correctement car le calcule etait assez compliqué et donné par des résultats toujours juste.
Ainsi on a remarqué que plus le virus s'éloigné de l'origine (haut gauche) plus le déplacement est grand lors d'ajout ou de retrait de Cellule. C'est comme ça qu'on est arrivé à une simplification en ajoutant ou en enlevant au coordonnées du Virus la distance par rapport a son origine (_x) divisé par la taille de la grille de cellule.

4 Rapports des diverses erreurs :

Le premier bug est celui fait par le petit cadre de selection, qui indique a quel endroit de la grille pointe la sourie, en effet il avait un probléme dans la synchronisation vertical. Il est possible que ça vienne du fait que la taille du model est de 800x600 et que la taille de la fenetre de 830x600 car il y a une une barre d'état d'une taille de 30 pixel donc pour l'axe vertical, on ajoute 30 pixels. C'est pour ça que rectangle de selection a été retiré du jeu pour évité de pertubé l'utilisateur lorsqu'il donne vie a une Cellule.

Il y a pas mal de bug avec les Virus car on l'a implémenté avant les Cellule du fait qu'il est plus facile d'apprendre a utilisé un vector simple (vector<Virus>) qu'un vector imbriqué dans un vector (

vector<vector<Cellule>>). au lieu pourat aussi remarqué ce meme problème de synchronisation vertical pour les Virus, en fait lorsque le Virus doit normalement infecter une cellule d'indicde (i, j), dans le jeu il infecte la cellule d'indice(i-1, j). Ce default de synchronisation aussi effet sur l'alignement vertical des virus avec l'ajout de Cellule. Un bug sur les Virus, récement apparut durant la redaction du rapport, est lors d'ajout de Virus dans le jeu : durant le lancement, de la simulation, certains Virus ne veulent pas s'ajouter, ou alors sont aussitôt retirés.

Un dernier bug repéré, est lorsque on modifie la matrice de Cellule en cliquant sur les boutons dans le menu, si on reste appuyé l'ajout de Cellule se fait en continue. Mais si on reste appuyé et que la sourie sort de de la fenetre de menu, la modification se fait en continue.

4.1 Proposition d'amélioration :

Pour des amélioration, on commencerait evidement par rêgler les diverses problémes sur les Virus précédemment évoqué. Par ailleur, il serait possible serai d'ajouté une classe OngletMenu, qui gère automatiquement des onglets dans le Menu qui contiendrait des vectors de Button et de Slider. Dès les début du projet, j'avais pensé a faire plusieur simulation en même temps, mais cette implémentation n'entré pas dans les temps qu'on avait prévu, et on craignait d'une mauvaise gestion de la mémoire. Cela aurait permis de mettre en place Une classe OngletGame qui disposera d'un Vector de GrilleCellule et d'un vector de GroupeVirus avec une sauvegarde pour chaque onglet.

Dailleur pour les sauvegarde, on aurait pu mettre un RenderWindow dans GameSave pour que lorsqu'on fait appel à charger ou sauvegarder la partie, cela affiche une fenetre avec avec par exemple trois sauvegarde possible. Pour cela il aurait suffit de faire un pointeur vers un GameSave, et juste faire un new quand on veut ouvrir la fenetre de sauvegarde et delete pour la fermer.

Avec les Cellule on aurait pu éventuellement rajouté la notion de contamination de Cellule voisine seulement si les voisines sont en concact direct au moment de l'infection, et on aurait pu définir à cette ocasion « une puissance d'infection » de la part des Virus, qui fait que plus elle est forte, plus la maladie se propage.

Une derniere amélioration possible serait de bien aligné la position de la fenetre de menu lors de l'ouverture de celle çi depuis le GameView, mais le souci vient de la librairié SFML 1.4 (celle que j'utilise) car il y a aucun accesseur sur la position d'une fenetre dans l'ecran.

On pourat retenir …

Que ce projet nous a parmis de nous familliarisé d'avantage avec la programmation orienté objet, en effet on a commencé par créer un diagramme de classe UML, de générer le code à partir de ce diagramme pour obtenir le squelette du programme. Il fallait ensuite faire beaucoup d'implémentation dans le code lors de l'évolution du projet, ce qui nous a fait découvrir différent type d'erreur classique qu'on retrouve en codant avec des classes. On a aussi manipulé une librairie inconnu, qu'on a pu comprendre le fonctionnement grace au notion apprise en cours.
En comparant au projet précédent fait au semestre 1, Cela nous a permis aussi de faire une meilleur gestion de la mémoire avec l'apprentissage des pointeurs, et des différentes façon de mettre en paramètre une variable. Cela nous a appris aussi a savoir mieux gérer l'organisation, et la maniere dont il faut dévellopper le code pour gagné un peu plus de temps pour y ajouté des idées supplémentaires.