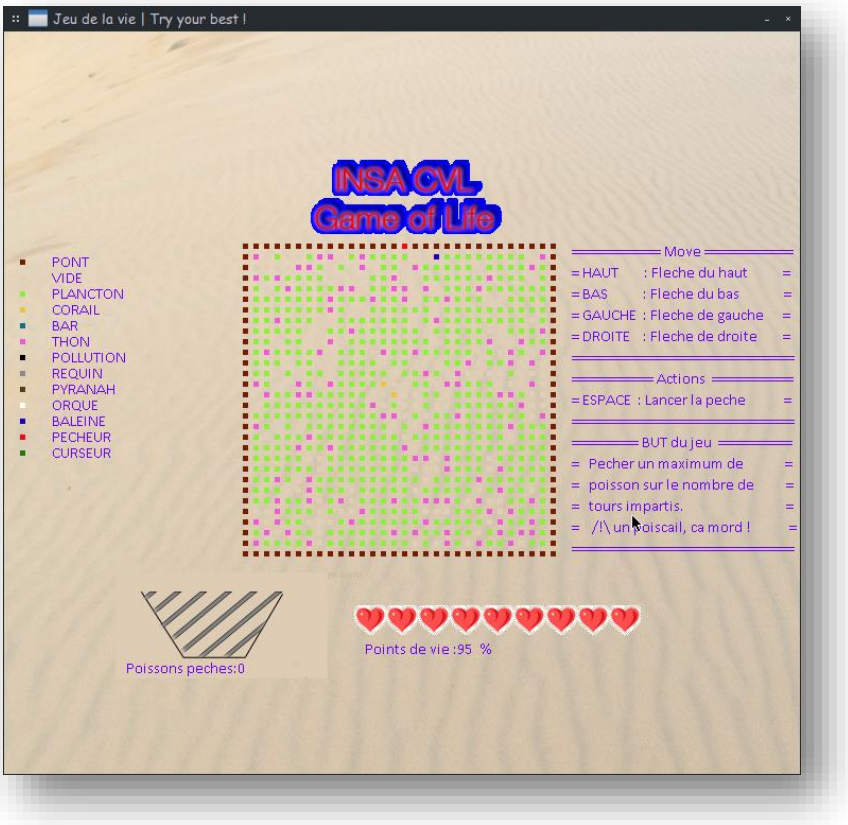
INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES CENTRE VAL DE LOIRE

RAPPORT DE PROJET

Algorithmie et programmation C



Professeurs encadrants :	Etudiants :
M. CLEMENTE	MADDAH Amine
M. TOINARD	FARAON Clément

3A STI - 2015

Introduction

À l'époque de son invention (par John Horton Conway en 1970), le jeu de la vie fut un jeu ne nécessitant aucun joueur : ce fut un automate cellulaire, où les états futurs sont créés à partir des états présents conformément à des règles préétablies.

Le jeu de la vie que nous avons conçu est également un automate cellulaire, c'est à dire qu'il peut fonctionner de lui-même, cependant, afin de donner au jeu un côté ludique et amusant, nous avons implémenté un pêcheur qui devra traverser toute la carte afin de rejoindre l'autre côté de cette dernière.

Ce programme s'inscrit avant tout dans le cadre de notre cursus en sécurité informatique à l'INSA Centre Val de Loire, il s'agit plus particulièrement d'un projet algorithmique et de programmation qui a pour but de nous donner l'occasion de pratiquer et développer nos capacités à réfléchir sur un problème d'algorithmique donné et d'en définir une implémentation judicieuse dans un langage particulier (le C ici).

Au cours de notre projet, nous avons très tôt rencontré des problématiques à partir desquelles nous avons fait des choix d'implémentation plus ou moins judicieux, ensuite, nous avons dû réfléchir longuement à quelques algorithmes importants du jeu afin d'éviter un maximum de problèmes ultérieurs, que nous détaillerons dans la partie II. En outre, l'équilibre des espèces nous a posé énormément de problèmes. De ce fait, nous nous sommes efforcés d'améliorer au mieux le game play du jeu.

I. Analyse du problème.

Dans cette partie, nous allons d'abord dans une première partie soulever un certain nombre de problème que nous tâcherons ensuite de résoudre dans une seconde partie.

I.1. Recueil des problématiques rencontrées.

I.1.1. Problématique 1 : Comment concevoir un programme facilement modifiable ?

On s'est d'abord interrogé s'il faudrait qu'on regroupe tout dans un seul ou plusieurs fichiers C, car au début du projet, il n'y avait rien qui nous poussait à adopter une approche modulaire.

I.1.2. Problématique 2 : la carte.

Vient ensuite le tour de la structure la plus importante du jeu : la carte du jeu. Il était possible d'utiliser ou bien une liste chaînée ou bien un tableau. Cependant, il n'y avait pas un réel intérêt pour la liste chaînée, puisque notre carte restait globalement statique. De plus, nous trouvons le choix du tableau plus facile à concevoir dans le cadre d'une future évolution du programme en intégrant des échanges de données comme cela peut être le cas en réseau.
En découle a posteriori la question suivante: faut-il choisir un tableau unidimensionnel ou un tableau bidimensionnel ?

I.1.3. Problématique 3 : Structure des animaux.

Les animaux représentent la principale entité présente dans le jeu, c'est pourquoi regrouper les paramètres définissant chaque animal semble crucial.

I.1.4. Problématique 4 : Les constantes.

Les animaux et l'environnement du jeu est régi par un certain nombre de valeurs constantes dans le temps. Le besoin de les placer toutes ensemble semble évident et indispensable afin de conserver une organisation claire et fluide.

I.1.5. Problématique 5 : Tests des conditions et remplissage de la carte.

Outre les constantes, le jeu est régi par un certain nombre de lois et conditions de survie qui le rendent vif et dynamique, comment peut-on agencer au mieux nos différents tests ?.

I.1.6. Problématique 6 : Déroulement des tours.

Comment doit se dérouler le jeu ? Parcourir le tableau à l'aveugle ? Ou trouver un choix plus judicieux, plus fidèle à un jeu aléatoire?

I.1.7. Problématique 7 : Cases vides et les murs.

Faut-il créer des structures spécifiques aux cases vides et aux murs ? N'y-t-il pas un choix plus judicieux ?

I.1.8. Problématique 8 : Le pêcheur.

Faut-il implémenter le pêcheur comme tous les autres animaux ou lui réserver une structure à part ? Peut-on trouver un moyen efficace d'abstraire un maximum le typage des espèces ?

I.1.9. Problématique 9 : Destruction des ponts et la mort du pêcheur.

Peut-on détruire les ponts crées de manière simple et qui fonctionne parfaitement avec ce qui a déjà été fait ? De même pour la mort du pêcheur.

I.1.10. Problématique 10 : Choix de l'interface graphique.

Enfin, comment ou avec quels outils l'interface graphique sera développée ?

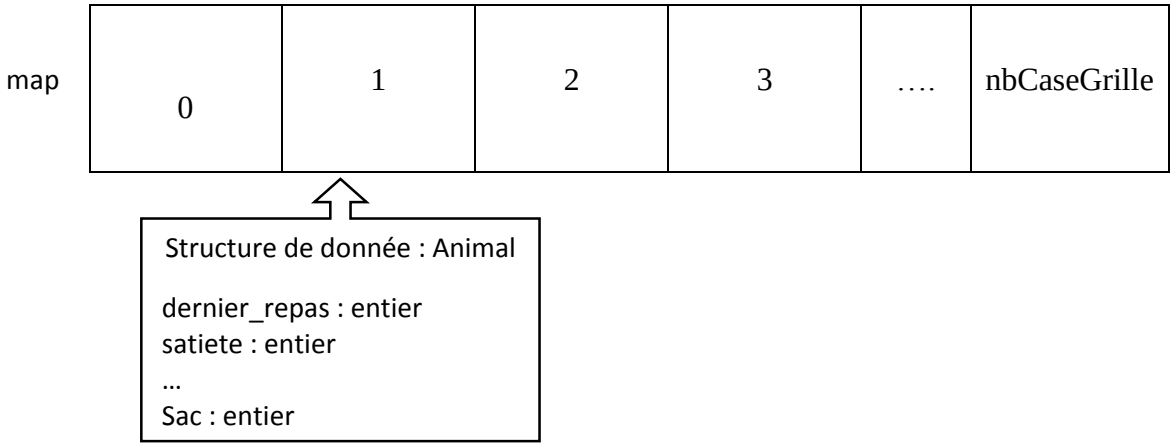
I.2. Choix d'implémentation.

I.2.1. Choix de l'approche modulaire.

La programmation modulaire nous a séduits malgré le fait qu'au départ il fut possible de tout regrouper dans un ou deux fichiers, on se doutait bien que sur le long terme cela n'allait pas tenir. De plus, avec l'intégration de la couche graphique, on s'est aperçu qu'on avait bien raison de faire ainsi. Pour chaque idée principale du jeu, nous avons créé un .c et .h. Exemple : pour la prédation, nous avons créé les fichiers predation.h et predation.c qui contiennent les fonctions et constantes en rapport avec la prédation du jeu.

I.2.2. Un tableau unidimensionnel.

Nous avons choisi d'opter pour l'approche la plus facile à gérer : un tableau. En effet, l'utilisation d'un tableau à une dimension car cela reste plus fidèle au fonctionnement de la mémoire.



I.2.3. Une simple structure d'entiers, et de type spéciaux.

Nous avons opté pour une structure simple regroupant tous les paramètres de chaque animal : durée de vie, gestation, fréquence de reproduction etc. Par ailleurs, nous avons pris le soin de choisir des entiers non signés pour la plupart de ses paramètres car ils sont tous censés avoir une valeur positive ou nulle.

Structure animal :

Variable représentant le nombre de tours depuis dernier repas :
dernier_repas : ***entier***

Variable représentant le nombre de tours depuis dernière reproduction :
derniere_reproduction : ***entier***

Variable représentant le nombre de tours de survie supplémentaire à partir du dernier repas
satiete : ***entier***

Variable représentant la taille que l'animal peut avoir de rempli.
taille_bide : ***entier***

Variable représentant l'énergie dépensée pour vivre.
Metabolisme : ***entier***

Variable représentant l'énergie dépensée pour se reproduire :
Gestation : ***entier***

Variable représentant la taille procurée aux prédateurs.
Taille : ***entier***

Variable représentant le nombre de déplacements maximum par tour.
saut_max : ***entier***

Variable représentant le nombre de tours que doit attendre un animal avant de se reproduire depuis la dernière reproduction.
frequence_reproduction : ***entier***

Variable représentant la durée de vie de chaque animal.
duree_survie : ***type durée (entier)***

Variable représentant le type de l'animal.
espece : ***type Type (entier)***

Les conversions entiers/types se font ainsi :

mur=-1, vide=0, plancton=1, corail=2, bar=3, thon=4, pollution=5, piranha=6, requin=7, orque=8, baleine=9.

I.2.4. #define et enum.

Nous avons décidé pour les constantes des paramètres de chaque animal de les définir dans un .h : le define.h, à l'aide des directives C #define et les énumérations enum. En outre, nous avons essayé par ce biais de n'avoir aucune valeur codée en dur.

Nous avions également pensé à l'élaboration d'un fichier de configuration (ce sont les fichiers conf.h et conf.c de notre code source), cependant nous avons eu énormément de mal à

équilibrer le système, et on estime que le joueur n'a nul besoin de les modifier, particulièrement en raison du déséquilibre que cela puisse provoquer.

I.2.5. Les switch.

Nous avons souvent opté pour un switch qui est bien plus élégant que des tests en if, et présente un réel intérêt quand il s'agit de définir et assigner des constantes à un type de données. Les switch et les enum fonctionnent très bien ensemble.

I.2.6. Un choix du déroulement du tour.

Nous avons opté pour choix simple et logique : un tant que qui fait avancer les tours jusqu'à un certain nombre et un pour imbriqué qui permet de lancer les fonctions importantes du jeu : satiété, prédation, reproduction, et déplacement.

Cependant, pour l'ordre des cases parcourues a été établi de manière hasardeuse par le biais d'un tableau fonctionnant comme une pile qu'on dépile au fur à mesure du parcours des cases de notre carte.

I.2.7. Les cases vides et les murs, de simples animaux.

Pour ces deux-là, nous avons choisi de les considérer comme des animaux à part entières, des animaux bien sûr immobiles et qui ne meurent jamais. Cela a beaucoup facilité leur implémentation et leur gestion. D'ailleurs, nous avons utilisé l'animal vide comme élément d'initialisation de notre carte.

I.2.8. Des paramètres pour le pécheur.

Nous avons choisi pour lui deux principales informations : sac et nombre de vie. Le sac afin de construire des ponts : chaque animal pêché permet la construction d'un pont, et le nombre de vie afin de rendre le jeu plus amusant et vif.

Le pêcheur est également à la base considéré comme un animal. Bien qu'on aurait dû dès le départ créer une structure pour lui seulement afin de faciliter plus tard l'implémentation du multi-pêcheur.

I.2.9. Des ponts détruits à l'image d'un animal mangé, et un pêcheur avec une durée de vie.

Grâce à notre approche modulaire et notre choix de profiter au maximum de la puissance du switch, la destruction des ponts et la noyade du pêcheur ne furent pas difficiles à implémenter : le pont est un animal (c'est un mur spécial) et le pêcheur par le biais de son nombre de point de vie, peut mourir de manière progressive afin de lui laisser du temps de jeu.

I.2.10. Le choix final de la SDL.

Bien qu'on eût parti sur du ncurses, nous avons finalement décidé d'implémentation la couche graphique avec la libraire SDL 1.2.

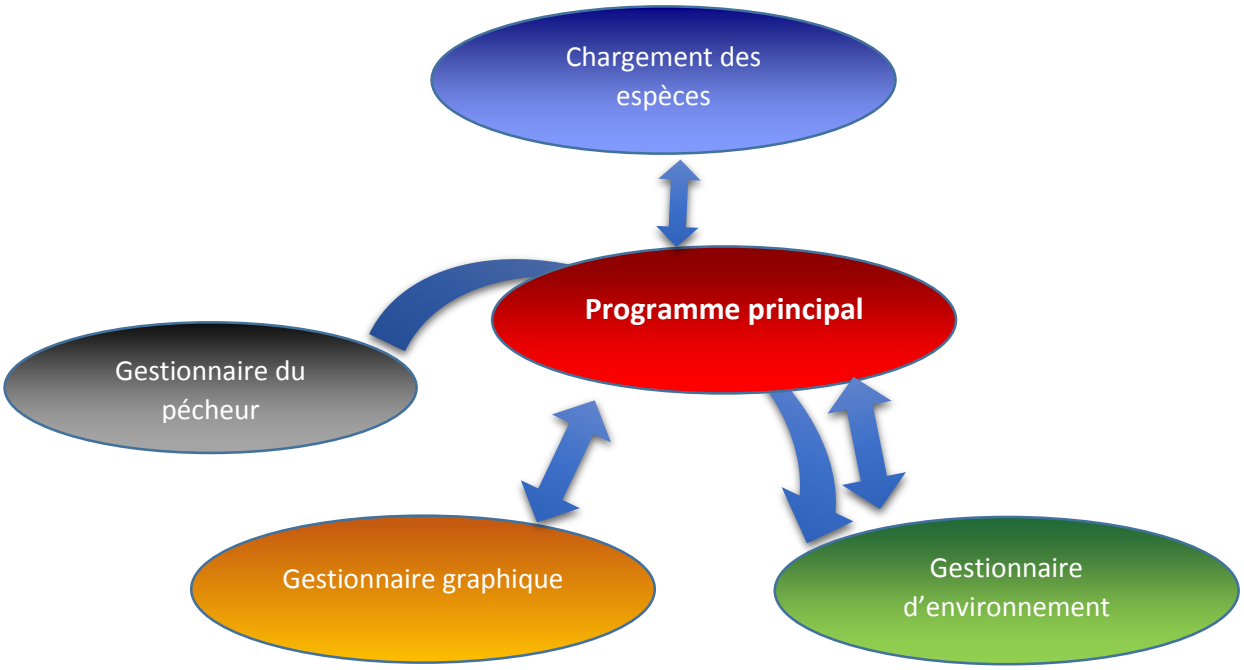
II. Conception

La conception est une part importante du projet puisqu'elle définit le bon déroulement de celui-ci. C'est pourquoi nous avons au plus tôt analysé les différentes problématiques possibles que nous pouvions rencontrés et comment nous allions les résoudre. Nous avons donc imaginé une conception en différents niveau logiciels afin de permettre une plus grande abstraction et donc d'être plus efficace lors de la découverte d'une nouvelle problématique.

Notre analyse conceptuelle qui en découle se divise en cinq grandes fonctionnalités différentes :

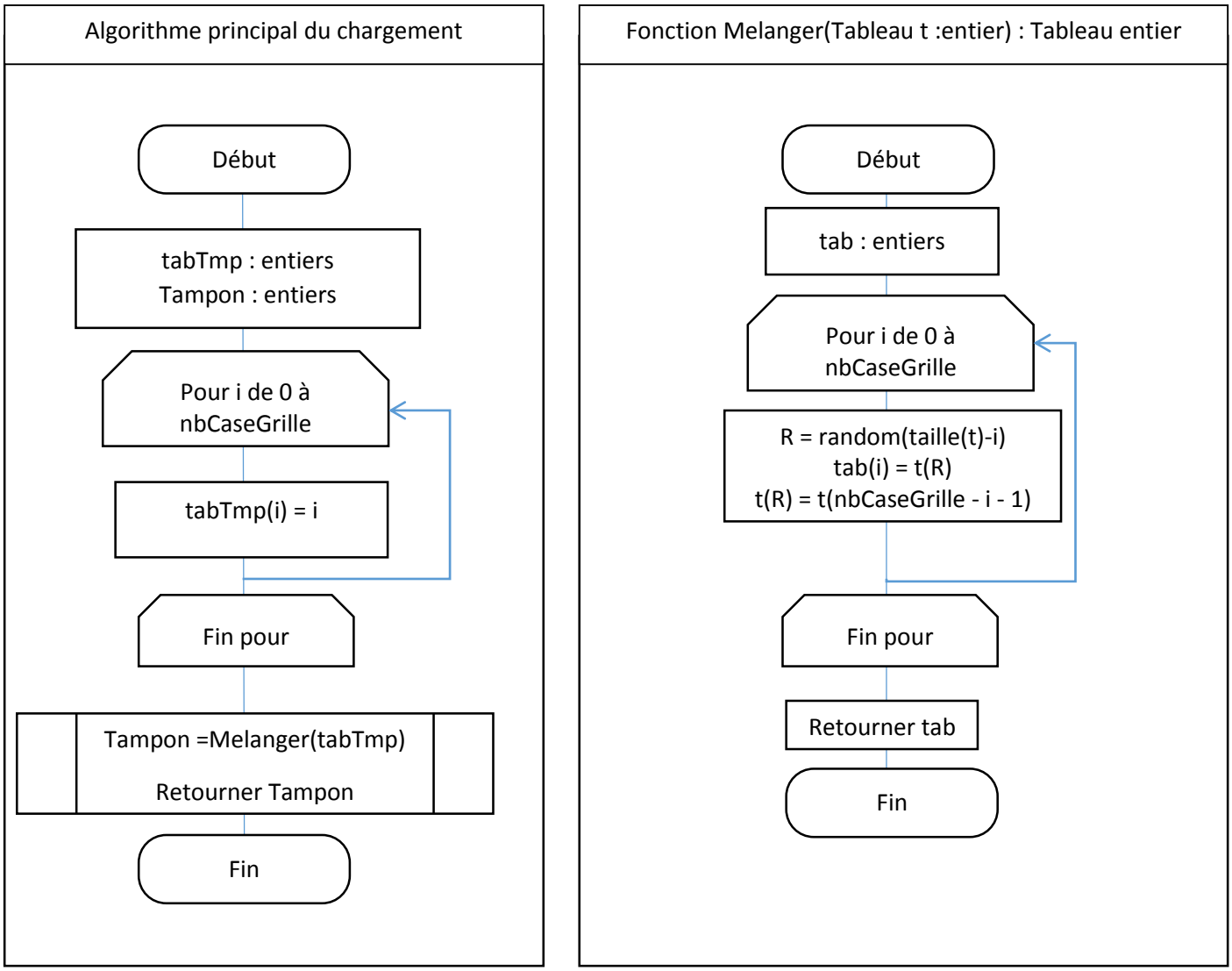
- Le programme principal
- Le chargement des espèces
- La gestion de l'écosystème
- La gestion graphique
- La gestion du pêcheur

Chaque couche communique avec des couches différentes selon le schéma ci-dessous :



II.1. Chargement des espèces

Le caractère aléatoire de la sélection des animaux est une étape importante pour plus de réalisme. En effet, à chaque tour de boucle, le programme choisi au hasard sur la carte un animal et celui-ci effectue les actions citées précédemment. La sélection de ces animaux est une unique sélection aléatoire. Le programme charge à chaque début de tour toutes les espèces de la carte dans une mémoire tampon qui fonctionne un peu comme une pile. Le chargement s'effectue selon l'algorithme suivant :



Une fois la fonction de chargement appelée, on récupère un tableau d'entiers qui sont les index des animaux placés dans un ordre totalement aléatoire. Explications des algorithmes.

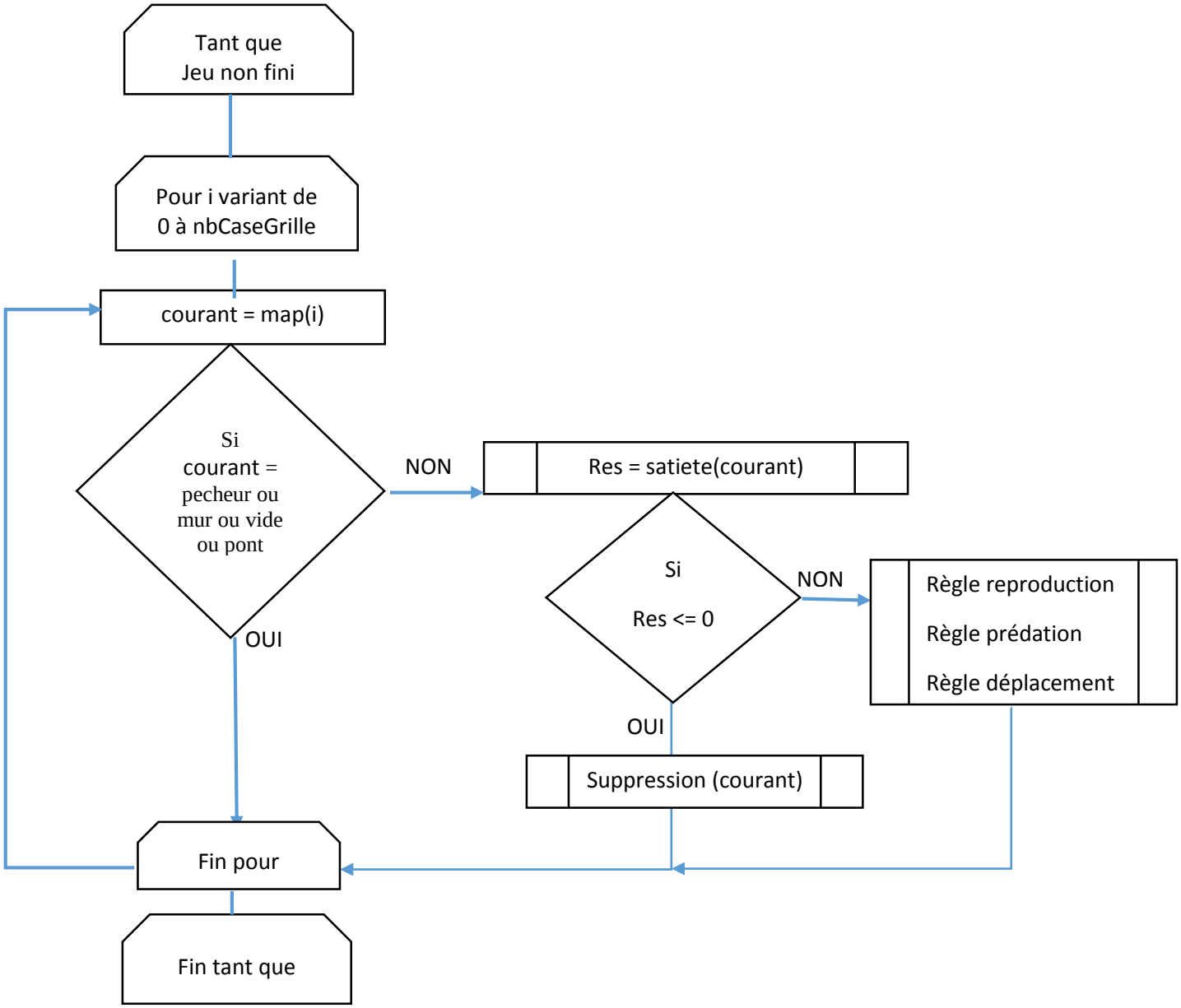
L'algorithme principal (à gauche ci-dessus) est l'algorithme appelée lorsque l'on souhaite charger le tampon du programme principal. La carte contient des cases comprenant des index de 0 à nbCaseGrilles. Dans la première partie, le programme génère un tableau d'index triés par ordre croissant de 0 à nbCaseGrilles. Le programme appel ensuite une fonction qui permet de mélanger ce tableau trié afin de simuler une « vie » dans l'éco système (les animaux jouent dans un ordre aléatoire et non dans un ordre défini). La fonction qui mélange les index, (à droite ci-dessus) prend en paramètre le tableau d'index trié. Elle va ensuite chercher aléatoirement une valeur dans ce tableau, la stocker dans la première case du tampon et échanger cette case avec la dernière case valide du tableau afin de ne plus reprendre cet index au tour de boucle suivant. Ainsi, on obtient au final notre mémoire tampon qui n'a plus qu'à être parcourue afin d'effectuer les actions sur tous les animaux.

II.2. le gestionnaire d'environnement

Lors de l'analyse du problème nous avons conclus deux aspects importants de la conception. Le contrôleur d'environnement est la première grosse couche logicielle implémentée. En effet, celle-ci a pour but de « réguler » l'équilibre de l'écosystème.

Le contrôleur d'environnement est un ensemble de fonctions agissant sur les structures de données Animal stockées dans le tableau unidimensionnel. Tout bonnement, celui-ci sert à contrôler et faire appliquer les différentes règles du jeu de la vie sur chaque structure de donnée présente dans la carte (zone mémoire réservée par un tableau unidimensionnel pointant sur les structure de données Animal).

A chaque début de nouveau tour, la couche logicielle de la gestion de l'environnement suit l'algorithme suivant :



Cet algorithme est l'algorithme très simplifié du jeu de la vie adapté à notre solution.

Explications :

A chaque parcours de la boucle for, on vient sélectionner l'animal, la structure de donnée, et appliquer les différentes règles comme la règle de survie ou encore la règle de reproduction ce qui permet de modifier les caractéristique des animaux sélectionnés. Si l'animal courant sélectionné en mémoire est du type mur ou du type pécheur ou du type vide ou du type pont, alors on termine le tour puisqu'aucunes actions légitimes est nécessaire pour l'aspect gestion de l'environnement.

II.3. le gestionnaire graphique

Le gestionnaire graphique est la deuxième couche logicielle la plus importante du projet. Il s'agit d'une couche totalement indépendante du gestionnaire d'environnement et qui permet l'affichage du jeu ainsi que des différents composants graphique (carte, pécheur, légende …). Nous avons pensé le gestionnaire graphique le plus général possible et le plus extérieur au programme possible afin de permettre une évolution graphique ultérieur sans altérer au mieux le programme principal. Le gestionnaire graphique utilise donc un espace partagé avec le gestionnaire d'environnement lui permettant de récupérer la position des animaux. En d'autres termes, il observe un accès complet à la carte et peut donc déterminer les espèces présentes sur la carte et dessiner les pixels de la couleur adéquate en fonction de l'espèce présente sur la case courante.

Cette couche graphique partage aussi avec le gestionnaire du pécheur quelques informations stockés dans le programme principal comme le filet de pêche contenant les index des poissons capturés ou la canne à pêche contenant les index de la carte sur lesquels passe la canne à pêche.

En somme, l'algorithme permettant l'affichage des pixels de couleurs est un simple parcours complet de la carte affichant en fonction du modulo de la taille d'un côté de la carte les pixels. Quand le parcours atteint le modulo de la taille de la carte, le curseur se repositionne au début décalé en ordonnée afin de continuer le parcours et donc l'affichage du reste de la carte.

II.4. Le gestionnaire du pécheur

Le pécheur est la dernière grande partie de notre logiciel. Cette surcouche se fixe au programme principal est permet par l'intermédiaire d'une mémoire partagée entre les différents autres gestionnaires. Ainsi, seront stockés les index où est positionnée la canne à pêche ou le filet par exemple.

La canne à pêche est une zone mémoire dynamiquement allouée à qui correspond la structure de donnée suivante :
Tableau Canne_à_pêche : entier
Taille(Canne_à_pêche) : fixée à 5

Soit le tableau suivant une représentation de la carte :

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80

Soit le pécheur en position 5, on remarque que la canne à pêche est pointée vers le bas et passe sur les indices 15,25,35,45 et 55 . Ces indices serviront par exemple à la couche graphique afin de dessiner la canne à pêche à l'écran ou au gestionnaire du pécheur afin de pécher le dernier poisson ciblé, entre autre le poisson en dernière case, ici 55.

Ainsi, la structure de donnée équivalente sera le tableau unidimensionnel :

Tableau Canne_à_pêche

15	25	35	45	55

Le filet quant à lui est géré par le gestionnaire du pécheur et non dans la gestion graphique ce qui explique l'absence d'affichage lors de la jetée du filet. Le filet occupe une surface bien définie qui est un carré parfait à une certaine distance selon le schéma suivant :

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80

Soit le pécheur en position 5, dans cet exemple le filet est lancé à une distance de 2 par rapport au pêcheur. Le carré représente le filet. Ce filet est stocké en mémoire dans une tableau fixe à deux dimensions.

Une fois le filet déployé, le programme tire au sort 8 poissons dans celui-ci, ces poissons pêchés, qui sont des index de cases sur la carte seront parcourus un à un afin de les supprimer et de les ajouter dans le sac du pêcheur.

III. Expérimentations et résultats

III.1. Ecosystème et configuration

Le paramétrage de l'environnement est une tâche complète qui nous a demandé une phase de tests assez poussée. Il serait trop long de lister la quantité phénoménale de combinaisons effectué dans le but de stabiliser l'écosystème cependant à force d'essais, nous avons réussi à affiner un écosystème relativement stable puisque à la fin du jeu, un nombre suivant d'espèce pour jouer reste en vie.
Voici donc notre fichier de configuration avec les différentes variables :

Nombre de tours maximum avant la fin d'une partie :
_NOMBRE_DE_TOURS_ 1000000

Insertion proportionnée des animaux sur la carte
_NB_ORQUE_ = _TAILLE_TOTALE_*0.01
_NB_THON_ =_TAILLE_TOTALE_*0.2
_NB_PYRANHA_ = _TAILLE_TOTALE_*0.01
_NB_CORAIL_ = _TAILLE_TOTALE_*0.01
_NB_BAR_ = _TAILLE_TOTALE_*0.05
_NB_REQUIN_ = _TAILLE_TOTALE_*0.055
_NB_BALEINE_ = _TAILLE_TOTALE_*0.06
_NB_PLANCTON_ = _TAILLE_TOTALE_*0.01 /* 10% de plancton */
_NB_POLLUTION_ = _TAILLE_TOTALE_*0.15 /* 15% de pollution */

Durée de vie des animaux :
d_vide=0, **d_corail** = 5, **d_bar**=5, **d_pyranha**=5, **d_requin**=5, **d_orque**=5, **d_baleine**=5, **d_thon**=25, **d_pollution** = 5, **d_plancton**=100,

Définition de la taille du bide de chaque animal :

_BIDE_PLANCTON_=3,

_BIDE_BAR_=20,

_BIDE_POLLUTION_ =21**,**

_BIDE_CORAIL_ =5,

_BIDE_PYRANHA_ =5,

_BIDE_REQUIN_ =6,

_BIDE_ORQUE_ =6,

_BIDE_THON_ =40,

_BIDE_BALEINE_ =20,

Définition de la taille procurée lorsqu'un animal est mangé :

_TAILLE_PLANCTON_ = 1

_TAILLE_CORAIL_ =1

_TAILLE_BAR_ =3

_TAILLE_THON_ =4

_TAILLE_POLLUTION_ =5

_TAILLE_PYRANHA_ =6

_TAILLE_REQUIN_ =7

_TAILLE_ORQUE_ =8

_TAILLE_BALEINE_ =9

Gestation :

_GESTATION_BAR_ =0

_GESTATION_POLLUTION_ =1

_GESTATION_PLANCTON_ =20

_GESTATION_THON_ =10

_GESTATION_PYRANHA_ =100

_GESTATION_REQUIN_ =100

_GESTATION_ORQUE_ =100

_GESTATION_BALEINE_ =100

_GESTATION_CORAIL_ =100

Fréquence reproduction :

_FREQUENCE_REPRODUCTION_PLANCTON_ =0

_FREQUENCE_REPRODUCTION_POLLUTION_ =1

_FREQUENCE_REPRODUCTION_BALEINE_ =2

_FREQUENCE_REPRODUCTION_PYRANHA_ =3

_FREQUENCE_REPRODUCTION_THON_ =4

_FREQUENCE_REPRODUCTION_ORQUE_ =5

_FREQUENCE_REPRODUCTION_REQUIN_ =6

_FREQUENCE_REPRODUCTION_BAR_=40

_FREQUENCE_REPRODUCTION_CORAIL_=1000

Définition du métabolisme :

_METABOLISME_POLLUTION =0

_METABOLISME_PLANCTON =1

_METABOLISME_PYRANHA =2

_METABOLISME_REQUIN =3

_METABOLISME_ORQUE =4

_METABOLISME_BALEINE =5

_METABOLISME_THON =6

_METABOLISME_BAR=200

_METABOLISME_CORAIL_=1000

Satiete :

_SATIETE_POLLUTION=-2

_SATIETE_BALEINE =1

_SATIETE_BAR =2

_SATIETE_PYRANHA =3

_SATIETE_ORQUE =4

_SATIETE_THON =5

_SATIETE_CORAIL =6

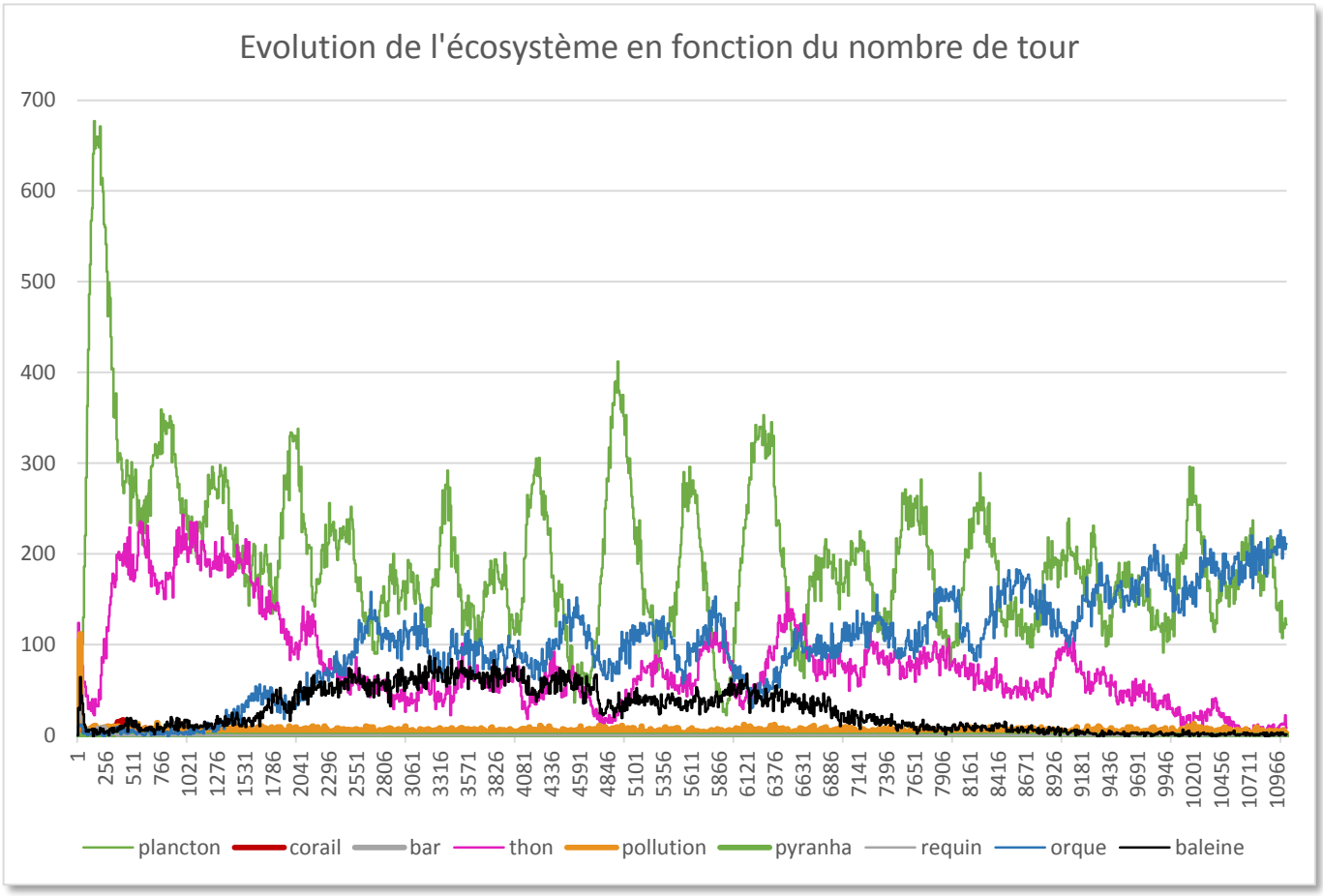
_SATIETE_REQUIN =7

_SATIETE_PLANCTON = 10000

Ces constantes sont donc définies et inchangeable car une modification entrainerait une perturbation majeure de l'écosystème.

III.2. Interprétations des résultats obtenus

Les paramètres réglés ci-dessous obtiennent des résultats suffisamment exploitables pour permettre une longue partie. Ainsi, sur presque la totalité des graphiques on obtient un système stable sur plus de 8000 tours. Seulement 5 espèces survivent au-delà comme le plancton, le thon, la baleine, l'orque et la pollution. Afin de permettre une jouabilité nous avons dû modifier quelques règles notamment sur les animaux autorisés à manger le pont ou les animaux pêchables. Ainsi, le pêcheur peut pêcher toutes les espèces sauf le plancton et le bar. On remarque que les espèces dont le nombre augmente engendrent la réduction de l'espèce opposée. Le système observe alors un certain équilibre perturbé par quelques variations de la quantité d'une population qui impacte forcément son espèce prédateur ou son espèce nourricière. L'allure des courbes reste convenable et on note des oscillations presque régulière ce qui dénote une certaine stabilité au cours du temps. Seule la baleine s'effondre vers le 7000 ème tour.



Conclusion

Ce projet de programmation et d'algorithmie nous a permis de découvrir un grand nombre de facettes du langage C en partant du fonctionnement du préprocesseur notamment avec les énumérations jusqu'aux problèmes de dépendances de libraires comme avec la librairie SDL.

Nous nous sommes également beaucoup enrichit par le biais de l'approche modulaire et de la réflexion algorithmique avant l'implémentation de nos structures et fonctions. L'analyse fonctionnelle a été plutôt compliquée à réaliser de notre point de vue du fait du manque d'expérience en pratique de programmation. Malgré quelques faux pas, nous avons pour la plupart du temps été méthodiques et nos choix d'implémentations se sont avérés assez judicieux pour les problématiques rencontrées ce qui nous a permis au final d'avoir une bonne version jouable du jeu de la vie, qui nécessite néanmoins encore des corrections afin d'en faire un jeu plutôt attrayant.

Dans l'ensemble ce projet aura été une bonne expérience sur le plan humain avec la gestion de projet où on s'est confronté aux difficultés de la conduite de projet et de la prise de décision. Cependant, le travail d'équipe nous a renforcé ce qui a permis de surmonter les difficultés rencontrés, mais aussi sur le plan de l'apprentissage de la conception d'un programme complexe.