Mini Projet pour les CIR CIPA3

ISEN YNCREA OUEST – NANTES / NILS BEAUSSÉ 2024/2025

Principe :

En tant que CIR vous maitrisez déjà le C de manière assez approfondie.

De fait, nous vous proposons ici d'enrichir vos connaissances à travers un projet assez libre de réalisation de mini jeux vidéo qui aura pour but d'enrichir vos connaissances techniques, notamment via plusieurs aspects :

- La programmation multi-processeurs via les threads et mutex.
- L'utilisation de bibliothèque SDL, qui est un classique (vous maitrisez déjà de par le cours de C++, la bibliothèque QT, la SDL est une autre approche, plus portée sur les jeux que QT).
- Une introduction à certains concepts de l'IA bio-inspirée

Le projet est en autonomie, l'idée étant que vous vous documentiez sur les méthodes permettant d'arriver à tout ça. Ce document à vocation à vous donner les consignes et à vous diriger vers les bons points de départ.

Le projet est en monôme, binôme ou trinôme : à vous de voir ce qui vous va le mieux pour coder efficacement. Plus le groupe est important, plus la communication entre vous et la répartition des tâches sera essentielle, ce qui est une compétence. Plus le groupe est restreint, plus votre travail individuel sera important et plus vous devrez coder vite et efficacement pour avancer (ce qui est une autre compétence).

À vous de voir ce qui vous sied le mieux.

Le projet donnera lieu à une note finale qui se surajoutera aux évaluations de C qu'ont vos autres camarades : l'idéal sera également de présenter votre jeu à vos camarades pour les plus motivés d'entre vous (non-obligatoire) (ce qui permettra aussi de motiver vos camarades avec vos créations 😊).

Note : dans tous les cas vous aurez les mêmes évaluations classiques du C que le reste de vos camarades, mais au vu de votre niveau nous pensons que vous êtes tout à fait apte à être évalué sans revoir tous les TPs de C débutant de 0.

Note 2 : Le but est de vous faire acquérir des connaissances et compétences intéressantes pour vous dans le domaine pro, pour votre futur, etc. De fait**, je vous recommande fortement**

d'éviter de copier bêtement des choses qui fonctionnent déjà sur le net (car il y en a plein, évidemment), ça ne vous apportera rien. Idem avec l'utilisation de ChatGPT.

On ne progresse que lorsqu'on se heurte aux problèmes et qu'on tente de les résoudre par nous même, si on ne fait que copier une solution ou recopier ce qu'une IA nous murmure, ça ne fait que rentrer par une oreille pour ressortir par l'autre.

Consigne :

Nous allons coder **un simulateur d'environnement marin en deux dimensions**. Dans lequel notre joueur pourra éventuellement venir naviguer pour y combattre des monstres, y récolter des ressources etc.

Selon votre avancement, on pourra intégrer une notion de jeux multijoueurs en réseau, mais l'idée initiale est de venir construire un système d'environnement « vivant » crédible.

Vous pouvez coder en C ou profiter des concepts objets du C++ si vous le désirez.

Voici les consignes :

1) L'environnement 2D doit être assez grand pour que le personnage y navigue relativement longtemps
2) **Il doit être persistant,** un peu comme la carte de votre jeu QT en C++ l'année passée. En clair : le jeu continue à vivre à un endroit même si cet endroit n'est pas affiché à l'écran.
3) Il doit être adapté à des architectures **multiprocesseurs**. Pour ce faire, on fera en sorte de séparer les taches en threads indépendants, en prenant garde au fait qu'avoir trop de thread n'est pas non plus pertinent. On utilisera ensuite les mutex pour éviter que les threads n'écrivent en même temps sur les variables.
4) On utilisera la bibliothèque SDL 2 pour l'affichage, le clavier, le son, etc. (mais pas pour les mutex, ni pour les threads, on utilisera des threads plus simples que ceux proposés par la SDL)
5) Le premier objectif est d'implémenter **un comportement cohérent de groupe de poisson**. Par chance il existe des systèmes de comportements très simple qui sont dérivés de l'étude des poissons ou des étourneaux, on appelle ça des « **boids** » (voir plus bas)
    a. Ces poissons devront interagir avec l'environnement (évitement d'obstacle, du joueur)
    b. Et avec eux même via les équations des boids (ce qui permet l'émergence de banc de poissons)
6) Par la suite vous enrichirez l'environnement (cailloux, plantes, récifs)
7) Selon votre avancement vous pourrez implémenter également des végétaux, avec tout ce que ça implique (naissance, croissance, mort, émission de spore, nécessité de pousser à tel ou tel endroit etc.)
8) On pourra intégrer enfin un avatar du jour et d'éventuels antagonistes ou des objectifs de ramassages de ressources, comme vous voulez : vous êtes assez libre des règles finales du jeu qui s'effectue dans cet environnement : soyez créatif.
9) Pour les plus avancées d'entre vous, quelques idées :
    a. Des termites marins ont construit des colonies sous l'eau, elle récolte des ressources afin d'agrandir leurs demeures. Implémentez ceci via le fameux *« algorithme des fourmis ».*

b. Implémenter la possibilité de jouer en réseau avec un camarade sur le même jeu.

Aide technique :

La SDL 2 :

La SDL est une bibliothèque permettant le dessin en 2D, la gestion de l'audio, des entrées sorties claviers et souris, etc. Elle est codée en C, mais dispose de correspondance de ses fonctions pour le C++.

Elle diffère de la bibliothèque QT que vous avez déjà vue dans le sens où elle est spécialisée dans l'affichage 2D et le support au JV.

La première étape sera donc de vous familiariser avec la bibliothèque SDL 2 et de réussir à la faire tourner sur votre PC.

Adresse :

https://www.libsdl.org/

Wiki avec tout ce qu'il faut pour en comprendre les tenants et les aboutissants et pour commencer :

https://wiki.libsdl.org/SDL2/FrontPage

En plus des tutos du wiki il existe de très nombreux tutos très complets pour vous aider :

https://zestedesavoir.com/tutoriels/1014/utiliser-la-sdl-en-langage-c/

https://devopssec.fr/article/cours-introduction-sdl-2

https://alexandre-laurent.developpez.com/tutoriels/sdl-2/

Vos poissons :

La notion d'êtres simulés simple au comportement proche de la réalité est l'une des bases de **l'IA développementale.**

Les « Boids » sont un concept de combinaison de comportement élémentaire (répulsion, rapprochement, etc.) inspiré des étourneaux et des poissons et qui reproduit à merveille leur comportement.

Pour vous donner une idée, voici quelques vidéos présentant le concept :

Le phénomène dans la vraie vie :

https://youtu.be/WAA6sdWrV20

https://www.youtube.com/watch?v=DOL6LHOT46w

Quelques exemples de code de boids :

https://www.youtube.com/watch?v=bqtqltqcQhw

https://www.youtube.com/watch?v=QbUPfMXXQIY

Des informations généralistes et des aides pour commencer à coder :

https://fr.wikipedia.org/wiki/Boids

https://vanhunteradams.com/Pico/Animal_Movement/Boids-algorithm.html

Une simulation de boids online :

https://boids.cubedhuang.com/

Les threads :

Les threads sont des parties de programme (généralement une fonction) qui s'exécutent en parallèle du programme principal.

- ➔ Ils sont lancés par le programme principal.
- ➔ Ils s'exécutent
- ➔ Et peuvent éventuellement être attendu par le programme principal.

Ils tournent en parallèle et peuvent même tourner sur un autre processeur (c'est le système d'exploitation qui décide où les attribuer au mieux).

Dans notre cas, l'attendu au départ sera de lancer **un thread par entité autonome** (par poisson par exemple) pour que celui-ci vive sa vie indépendamment du reste.

Évidemment cela pose quelques problèmes : quand un thread veut toucher à une donnée commune à plusieurs thread (une structure passée en argument de plusieurs thread et qui serait dans le main par exemple, ou une variable globale), alors il peut y avoir deux écritures en parallèle, ce qui est très mauvais d'un point de vue binaire (une suite de bit dont le milieu est modifié pendant que le centre l'est par quelqu'un d'autre donnera très probablement n'importe quoi, surtout si un troisième est en train de lire à ce moment-là). C'est pour ceci que les threads vont toujours avec un mécanisme nous permettant de verrouiller les variables communes. On parle de **mutex**.

Quelques tutoriels sur les threads et mutex :

https://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html

http://www.bien-programmer.fr/pthreads.htm

https://franckh.developpez.com/tutoriels/posix/pthreads/

Si vous êtes en C++, sachez qu'ils sont intégrés de base dans la bibliothèque standard de C++, qui en simplifie pas mal l'usage :

http://www.iro.umontreal.ca/~dift1169/cours/ift1169/communs/Cours/2P/1_03/C12_1169_2P.pdf

https://bousk.developpez.com/cours/multi-thread-mutex/

Les ressources :

Pour votre programme vous pouvez évidemment dessiner à la main les objets à afficher, etc. Mais il est aussi possible de recourir à des ressources graphiques disponibles sur internet.

Voici quelques exemples :

https://craftpix.net/freebies/

https://itch.io/game-assets/free

Idem pour l'audio :

https://pixabay.com/sound-effects/search/game/

etc.

Plantes :

Il existe peu de méthode « standard » pour simuler la vie de populations de plantes. À vous de trouver des idées.

Quelques pistes :

- Dans la vraie vie, les plantes se nourrissent des éléments contenus dans le sol, principalement de trois éléments (N, P et K : azote, phosphore et potassium)
- Ces éléments sont en quantité finie, mais la plante, quand elle meurt, les relâche (potentiellement de manière dispersée dans la zone).
- Les racines de la plante peuvent extraire doucement des éléments de la roche mère en dessous (ce qui permet petit à petit au fur et à mesure que les plantes meurent de régénérer le sol)
- Certaines plantes vivent bien dans un environnement riche en l'un des éléments et pauvre en d'autres, et inversement. D'autres plantes croissent partout, mais très lentement, et d'autres encore ne vivent très bien qu'en sol très riche.
- De fait, une première idée peut être de séparer votre terrain en différent type de sol contenant ces trois éléments. Puis de définir des espèces qui poussent plus ou moins vite dans ces sols + une durée de vie.
- Enfin il faut définir le fait que les spores se répandent.

Bonne chance 😊

Courant :

Probablement ce qui semble le plus simple, mais ce qui peut être plus tricky qu'on ne le pense. Un courant est un mouvement temporaire de l'eau dans une zone.

Modéliser les mouvements de l'eau en divisant votre terrain par « case » peut être une idée. Le mouvement se répercutant alors dans les cases voisines, etc. Il faudra alors tenir compte de ce mouvement de l'eau (qui n'est qu'une valeur finalement) pour l'ajouter aux entités mobiles qui circule dans la case.

La chose peut être très simple (des courants fixes dans des zones à intervalle régulier), ou assez complexe. À vous de voir.

Termites marines :

L'algorithme des « fourmis » permet de modéliser la récolte des ressources de la part de fourmis dans un environnement, il est énormément documenté et c'est celui-ci qu'il faudra appliquer.

Référence pour commencer :

https://fr.wikipedia.org/wiki/Algorithme_de_colonies_de_fourmis

- Il peut être malin de mélanger ceci à la propagation des plantes, ou de forcer les poissons à les éviter, par exemple.

Réseau :

Il existe de très très nombreuses manières de communiquer en réseau.

Pour un jeu je vous recommande ENET :

http://enet.bespin.org/

Qui est assez élémentaire.

Conclusion :

Beaucoup de possibilités donc, ne vous laissez pas décourager par l'ampleur de la tâche, fixer vous un objectif simple pour commencer : par exemple un programme de base joli visuellement avec des bancs de poissons, et améliorez le ensuite incrémentalement dans la direction de la consigne, mais en réalisant les choses pas à pas et objectif par objectif

Bonne chance à tous !