Simulation d'un écosystème maritime en 2D

Ze-Xuan Xu, Ibrahim Maatoug, Ali Bahja, Christopher Bilba, Zakaria Sabraoui

Table des matières

I. Introduction

Les écosystèmes marins représentent un élément vital de la biosphère terrestre, jouant un rôle crucial dans la régulation du climat, la fourniture de ressources alimentaires et la préservation de la biodiversité. L'objectif principal de cette recherche est de développer une simulation informatique capable de modéliser les interactions complexes au sein d'un écosystème marin en deux dimensions.

II. Notion

— background : c'est la partie qui ne change jamais, cette partie est composée de case étant immuable.

— foreground : c'est la partie qui contient les poissons de notre écosystème, cette partie change à chaque tic.

III. Aspect Méthodologie

A. Description

Pour représenter les composants de l'écosystème marin, notre approche de modélisation repose sur une structure basée sur des entités. Chaque organisme marin est représenté individuellement en tant qu'une entité, avec ses propres attributs tels que nom, l'âge, la faim et les caractéristiques comportementales. Les interactions entre les entités, y compris la prédation, la compétition pour la nourriture et les comportements de reproduction, sont modélisées à l'aide de règles définies en fonction des connaissances écologiques existantes.

De plus, l'environnement marin est également modélisé dynamiquement tels que la disponibilité des ressources alimentaires et les rayonnements solaires. Ces variables environnementales peuvent influencer les comportements des organismes et les taux de survie, de croissance et de reproduction.

B. Technique

Pour reproduire la dynamique temporelle et spatiale de l'écosystème en 2D, nous utilisons des techniques de simulation basées sur des modèles individus-environnement. Ces techniques reposent sur la mise à jour itérative des états des agents et de l'environnement à chaque pas de temps, en fonction des règles définies et des conditions environnementales actuelles.

Explication des techniques de simulation :

- Simulation à pas de temps discret : Nous divisons le temps en intervalles discrets et mettons à jour les états des agents et de l'environnement à chaque pas de temps. Cela nous permet de capturer les changements dynamiques dans l'écosystème au fil du temps.
- Modélisation spatiale en 2D : Nous représentons l'espace marin en deux dimensions, en utilisant des coordonnées cartésiennes pour définir la position des agents et des cellules environnementales. Cette approche nous permet de tenir compte des interactions spatiales entre les organismes et leur environnement.
- Algorithmes de déplacement et de recherche : Nous utilisons des algorithmes de déplacement et de recherche pour simuler les déplacements des organismes marins dans leur environnement, ces algorithmes sont utilisés pour la recherche de nourriture et l'évitement des prédateurs.
- Modélisation des interactions : Nous définissons des règles spécifiques pour modéliser les interactions entre les agents, telles que la prédation, la compétition pour les ressources et reproduction. Ces règles déterminent les conséquences des interactions sur les états individuels des entités.

En combinant ces techniques de simulation, nous sommes en mesure de reproduire de manière simpliste la dynamique temporelle et spatiale de l'écosystème marin en 2D, en capturant les interactions complexes entre les organismes et leur environnement.

IV. Aspect Théoriques

Lors de la modélisation de l'écosystème marin en 2D, plusieurs concepts écologiques fondamentaux guident notre approche. Ces concepts sont essentiels pour capturer la dynamique complexe des interactions entre les organismes et leur environnement.

A. Cycles alimentaires

Nous avons dû implémenter la notion de faim et plusieurs espèces de poissons pour créer une chaîne alimentaire avec des relations de proie et prédateur entre chaque espèce pour s'assurer que le cycle alimentaire puisse fonctionner.

B. Compétition

Plusieurs espèces ont une même proie et donc ainsi cela créera une compétition entre les espèces.Mais comme les entités de la même espèce ont tous les mêmes proies cela créera aussi une compétition entre les membres de l'espèce

C. Cycle de vie

Chaque espèce a une durée de vie différente et les membres de la même espèce ne meurent pas au même âge, ce qui nous a poussé à créer un intervalle de durée de vie (fidèle à la réalité) dont l'age maximum est compris dedans assigner aléatoirement.

D. Reproduction

Comme les membres d'une espèce finiront tôt ou tard par mourir, il faudra qu'ils puissent se reproduire pour que leur espèce puisse continuer à exister, ainsi, nous avons introduit la notion de reproduction dans notre code.

V. Aspect Technique

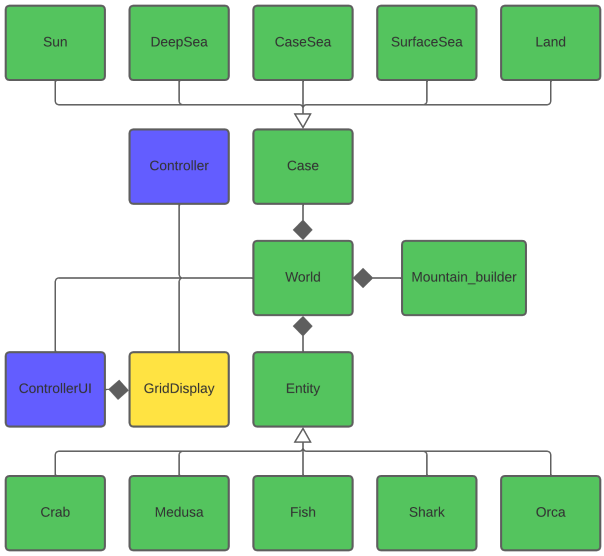

Figure 1. Modèle (vert), vue (jaune), contrôleur (bleu)

Notre code utilise la structure MVC, on a séparé la partie modèle, vue et contrôleur dans notre code.

- **Modèle :** contient deux matrices *numpy* ou l'un sert pour modéliser le background et donc l'affichage en arrière-plan, ainsi que l'autre qui modélise le foreground et donc la partie de l'écosystème avec les espèces aquatiques.
- **Contrôleur :** manipule la partie modèle pour que les poissons bougent, vivent, mangent et meurent.
- **Vue :** en fonction de la partie modèle, affiche le background et le foreground.

A. *modèle*

Notre modèle contient :

- Classe *Entity* et toutes les classes qui héritent de cette classe qui représentent les différents types de poissons. Ces classes dans *world_entities.py*.
- Classe *Case* et toutes les classes qui héritent de cette classe qui représentent les différents types de case qui peuvent se trouver sur notre écosystème dans *world_cases.py*.
- Classe *Mountain_builder* qui crée des montagnes dans les fonts marins de notre background dans *World.py*.
- Classe *World* qui gère les deux matrices numpy et contient des fonctions pour créer les deux matrices et manipuler la matrice correspondant au foreground dans *World.py*.

1) *Background*

La matrice pour la partie background est composé de sous-classes de Case. Elle est générée d'une manière aléatoire à chaque nouvelle partie pour la taille des différentes couches telle que la couche ciel, mer, mer profonde, etc.
De plus, la taille des cotes et les pentes des contes sont aussi générée aléatoirement.
Enfin, notre classe Mountain_builder créer aléatoirement des montagnes dans notre matrice background.

2) *Foreground*

Les cases de la matrice pour la partie foreground contiennent soit zéro comme un entier, soit une sous-classe de *Entity*. Elle est générée de manière aléatoire selon une configuration qui se trouve dans le fichier *entities_settings.txt* donc chaque ligne correspond au format :
Entity min_number max_number min_age max_age
Notre programme utilise le fichier de configuration pour créer le fichier entities.txt qui contient les poissons à ajouter dans la matrice du *foreground*.

3) *Mountain_builder*

Créateur de montagne qui construit aléatoirement une montagne de type **A** ou **B** avec une hauteur aléatoire. Les montagnes de **type A** sont des montagnes *triangle-rectangle* et les montagnes de **type B** sont des montagnes *triangulaires simples*.

B. *Contrôleur*

Le programme contient une boucle qui appelle en permanence le contrôleur pour qu'il modifie le modèle selon l'état donné et définie le nombre de planctons dans l'eau en fonction du rayonnement du soleil.

Cette boucle gère aussi les actions exécutées par l'utilisateur en les stockant dans une sorte de liste partagée avec un thread qui s'occupe de l'affichage et de la gestion des événements.

FIGURE 2. Boucle contrôleur

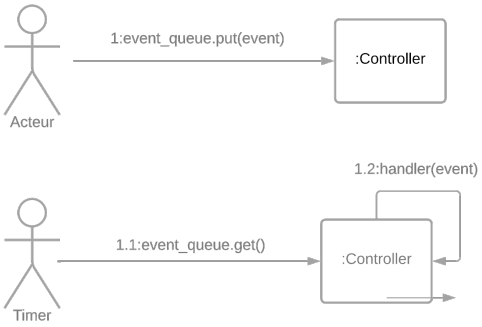

FIGURE 3. boucle

1) *update_entities*

demande au contrôleur de mettre à jour le status de chaque entité avec *__status_update* et faire bouger les entités dans le modèle d'un tour avec *__entity_update*.

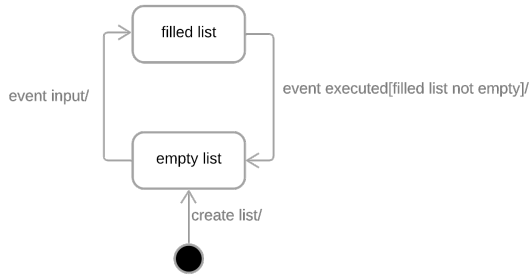

FIGURE 4. boucle

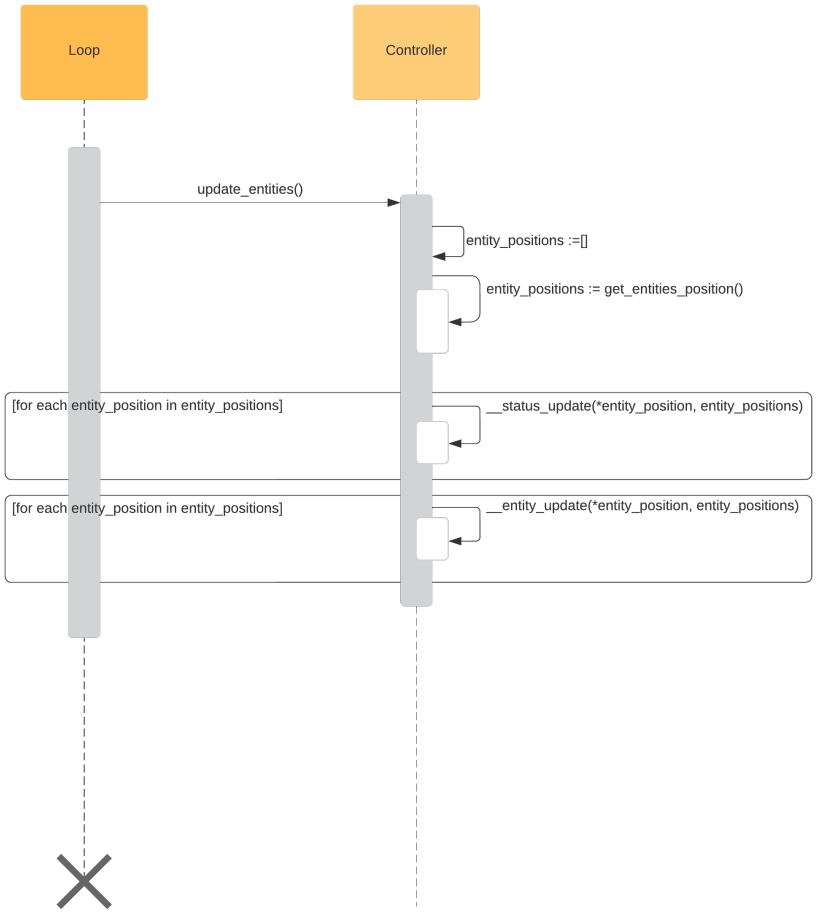

FIGURE 5. status update

a) __status_update

Cette méthode incrémente l'âge de l'entité, augmente la faim, décrémente le temps attente pour donner naissance, nourrit une entité de type Fish s'il est à la surface selon la concentration de plancton et vérifie si l'entité meurt de faim ou de vieillesse. Cette fonction est une implémentation pour le cycle de vie, cycles alimentaires et reproduction.

b) __entity_update

Cette méthode demande à l'entité l'action qu'il veut faire en appelant la fonction *brain* de l'entité, qui à son tour décidera selon l'environnement une action entre [**Idle, Flee, Predation, Mate, Stay**] et retourne l'action à exécuter à *__entity_update*. Par la suite, Cette fonction va appeler une fonction correspondante à l'action à exécuter.

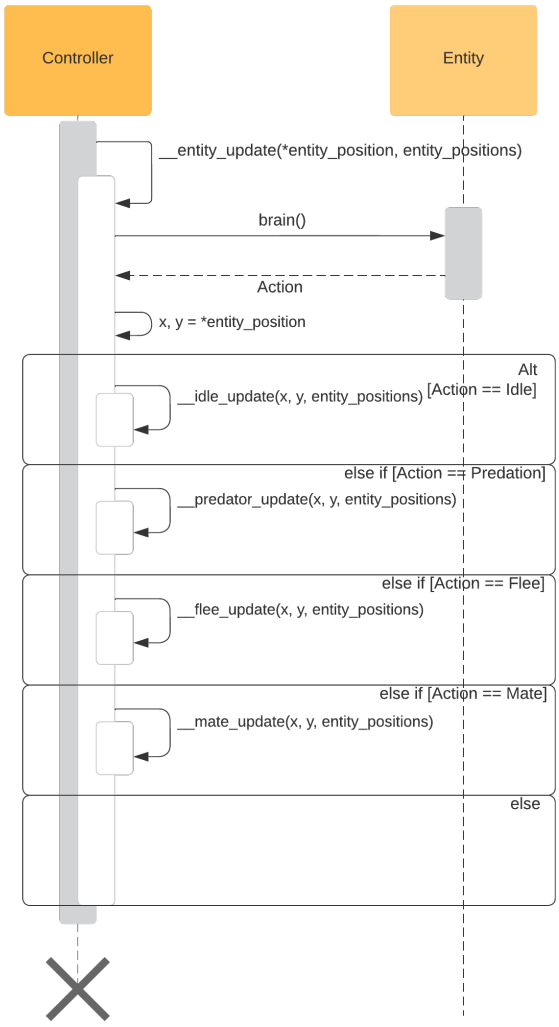

FIGURE 6. entity_update

2) plankton_update

Met à jour la concentration de plancton selon le rayonnement solaire.

3) Gestion des événements

Comme expliquer au-dessus, les événements sont stockés par la boucle principale dans une liste et sont extraits de cette liste dans une autre boucle dans la vue. Cette boucle arrête affichage, change de fenêtre ou affiche les graphes et les détails en fonction des événements.

Les actions de l'utilisateur sont converties en événements par *pygame*.

Le reste des événements qui n'agissent pas sur la vue ne sont pas gérés par la partie vue, mais plutôt par *ControllerUI* qui fait partie du contrôleur.

C. Vue

La partie vue contient une boucle qui est exécutée par un second thread. Cette boucle gère l'affichage de la vue. Elle possède plusieurs booléens pour décider la fenêtre à afficher, les événements concernant le changement de fenêtre et les booléens à modifier.

Pour l'affichage de la vue, on utilise la librairie *pygame* pour la création d'une fenêtre et l'affichage du background et foreground sur la fenêtre. l'affichage utilise la partie modèle, plus précisément les deux matrices background et foreground. La vue affiche d'abord le background puis le foreground pour que les poissons soient affichés au-dessus des cases.

1) Affichage du Background

Selon la classe de l'instance qui occupe chaque case de la matrice background, on affiche l'image correspondante à la classe, ainsi que notre background est formé d'image carre qui forme notre écosystème.

Comme la matrice qui s'occupe du background est constante, l'affichage sera toujours la même.

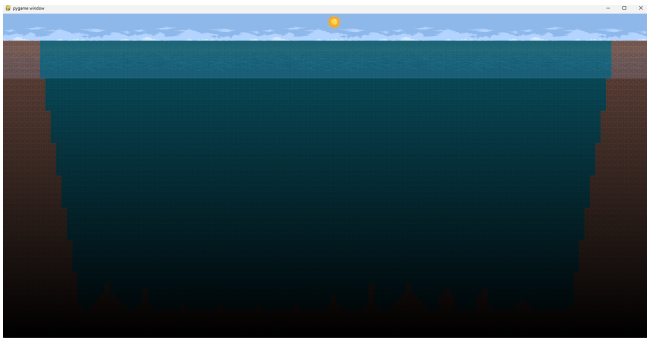

FIGURE 7. Background

2) Soleil

L'affichage du soleil change en fonction du paramètre concernant le rayonnement solaire.L'utilisateur peut modifier ce paramètre en appuyant sur les touches W et X.

3) Menu informatif sur les touches

Le menu informatif est affiché lorsque l'utilisateur appuis sur la touche **DOWN**.

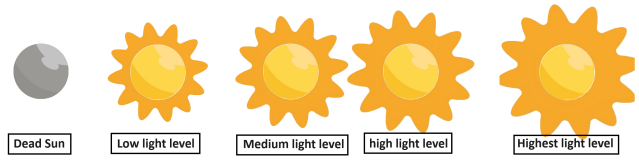

FIGURE 8. Soleil

Ce menu présente toutes les touches pour activer ou désactiver les fonctionnalités optionnelles.

4) Affichage du Foreground

La touche **UP** permet de revenir en arrière.
Dans la matrice foreground, si une case contient une sous-classe de Entity, on affiche par-dessus l'image de la case de la matrice background avec l'image de l'espèce maritime correspondante. Cela est dû au fait que la matrice peut avoir des cases remplies par un 0 et donc une case sans poisson.

5) Graphe

La touche **G** affiche le graphe. La boucle principale collecte les données de la matrice foreground et les sauvegarde. Ainsi, le graphe, si activé, est affiché en fonction des données stockées.

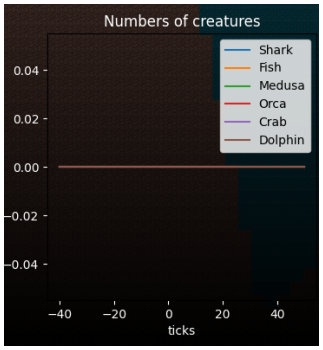

FIGURE 9. graphe

6) Affichage des informations d'un poisson choisi

La touche **T** affiche le tableau des informations d'un poisson sélectionné. La flèche **gauche** et **droite** pour sélectionner entre les poissons.
Le tableau ci-dessous affiche les données du poisson sélectionné :

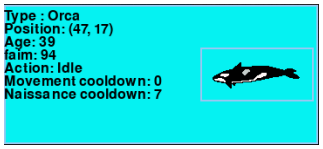

FIGURE 10. target data

7) Bar de faim

La touche **H** affiche la bar de faim.
Cette option permet d'afficher ou non sur tous les poissons une bar de faim au-dessus d'eux.

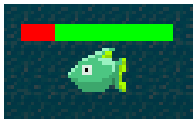

FIGURE 11. hunger bar