Sarah Azzi, Alexandre Crestien, Hazel Cunuder

2025

Wa-Tor

Simulation toroïdale d'un écosystème marin

Sarah Azzi, Alexandre Crestien, Hazel Cunuder

2025

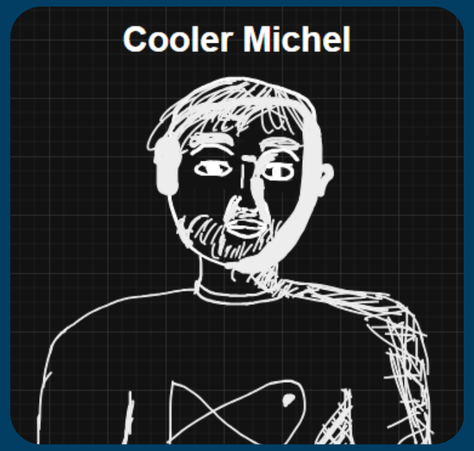
Introduction

Création d'une simulation basée sur le modèle Wa-Tor pour étudier les interactions entre différentes espèces marines. Cette simulation servira d'outil pour les biologistes marins afin de comprendre et prédire les évolutions des populations dans un environnement donné

Utilisation de Git, Python, PostgreSQL, Tkinter et Matplotlib.

L'équipe

Sarah Azzi



Alexandre Crestien

Hazel Cunuder

Sarah Azzi, Alexandre Crestien, Hazel Cunuder

2025

Wa-Tor

Objectif : Simuler la vie et l'évolution d'un écosystème marin.

Comment : Entitées gérées par un monde

Résultat attendu : 3 espèces qui vivent en équilibre

Outils collaboratifs : Trello, Github, Discord

Environnement de travail : Visual Studio Code, bash

Langages utilisés : Python (modules: tkinter, matplotlib), SQL

Sarah Azzi, Alexandre Crestien, Hazel Cunuder

2025

Architecture

01. /src

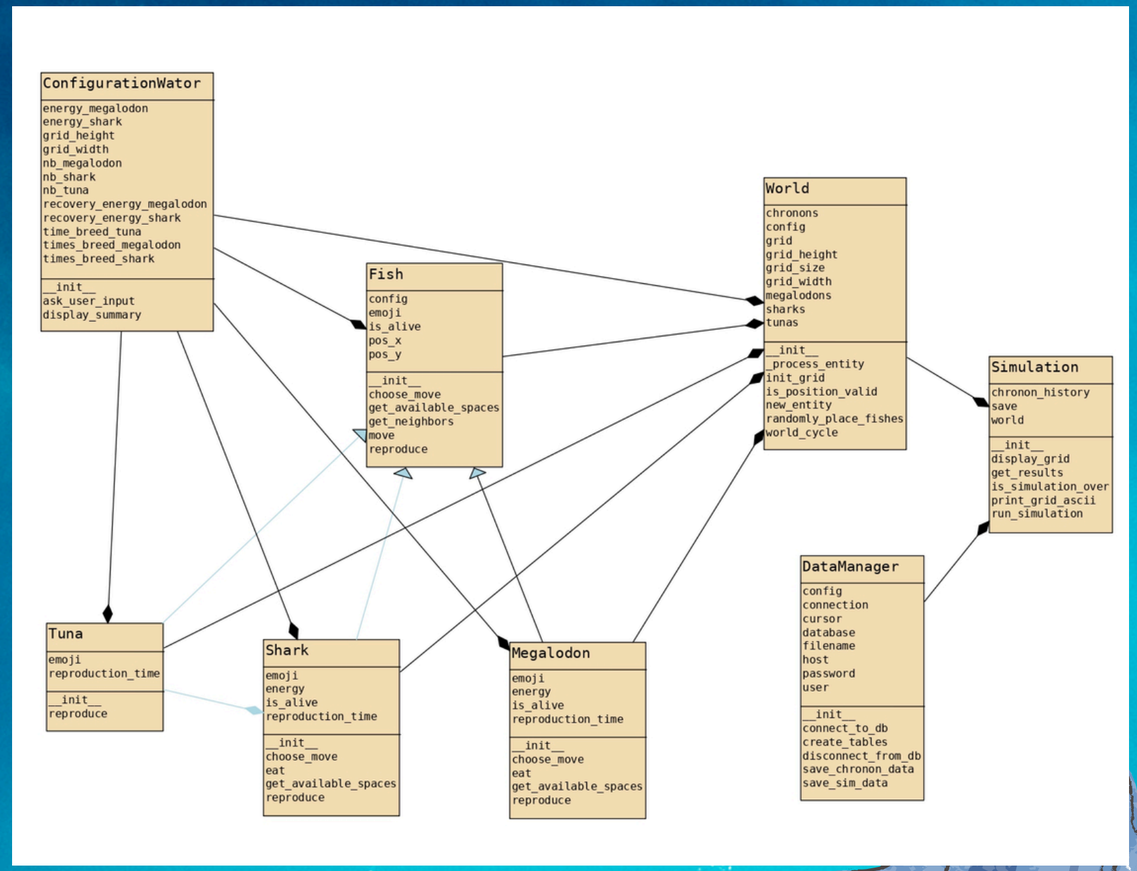
02. /utils/

03. /entities/

04. /main.py

05. /interface/

06. /simulation.py



Sarah Azzi, Alexandre Crestien, Hazel Cunuder

2025

Simulation

```python
class Simulation:
    def __init__(self, world: World, graph: SimulationGraph):
        """
        Initialize the simulation with a world and setup data tracking.
        Creates a DataManager instance for saving results and initializes an empty list to track population history over time.

        Parameters:
            world (World): The world instance containing the ecosystem grid and entities
        """
        self.world = world
        self.save = DataManager()
        self.chronon_history: list = []
        self.graph = graph
```

01. run_simulation

02. _simulation_step

03. is_simulation_over

04. get_results

Sarah Azzi, Alexandre Crestien, Hazel Cunuder

Configuration ⚙

```python
class ConfigurationWator:
    def __init__(self, interactive: bool = True):

        if interactive:
            self.grid_width = self.ask_user_input(
                "Enter the grid's width: ",
                10,
                min_value=5,
                max_value=100
            )

```

La classe ConfigurationWator sert de **panneau de contrôle** pour la simulation. Elle centralise tous les paramètres vitaux (taille du monde, populations initiales, règles de biologie) en un seul endroit. Son rôle est de garantir que la simulation démarre avec des valeurs valides.

Sarah Azzi, Alexandre Crestien, Hazel Cunuder

2025

World

01. Crée le Monde

02. Place les poissons

03. Déplace poissons

04. Gère les chronons

05. Crée les bébés

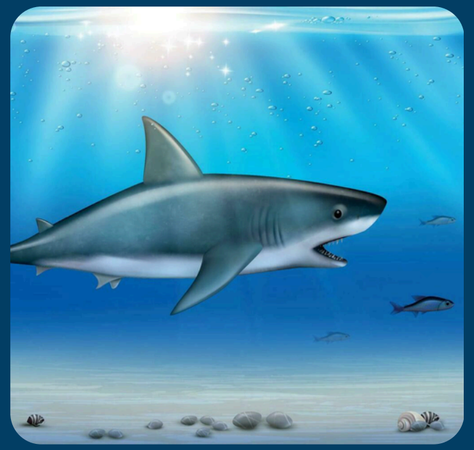
```python
def world_cycle(self):
    """
    Execute one time step of the simulation for all entities.
    Processes each species in order (megalodons, sharks, tunas), moving living entities and handling reproduction.
    Removes dead entities from species lists and adds newborns after all movements are complete to avoid processing them twice in the same cycle.
    Increments the chronon counter to track total simulation time.
    We process species separately to maintain clear predator-prey hierarchy during each cycle.
    """
    new_megalodons: list[Megalodon] = []
    new_sharks: list[Shark] = []
    new_tunas: list[Tuna] = []

    for megalodon in self.megalodons:
        if megalodon.is_alive:
            self._process_entity(megalodon, new_megalodons)

    self.megalodons = [m for m in self.megalodons if m.is_alive]

    for shark in self.sharks:
        if shark.is_alive:
            self._process_entity(shark, new_sharks)

    self.sharks = [s for s in self.sharks if s.is_alive]

    for tuna in self.tunas:
        if tuna.is_alive:
            self._process_entity(tuna, new_tunas)

    self.tunas = [t for t in self.tunas if t.is_alive]

    self.megalodons.extend(new_megalodons)
    self.sharks.extend(new_sharks)
    self.tunas.extend(new_tunas)

    self.chronons += 1
```

Sarah Azzi, Alexandre Crestien, Hazel Cunuder

2025

Fish 🐠

01. Tuna



02. Shark

03. Megalodon

Sarah Azzi, Alexandre Crestien, Hazel Cunuder

2025

Fish 🐟

```python
class Fish:
    def __init__(self, pos_x: int , pos_y: int, config: ConfigurationWator)-> None:
        """
        Initialize a fish entity with position and configuration.
        Sets the fish's position, configuration, alive status, and default emoji representation.

        Parameters:
            pos_x (int): The x-coordinate of the fish's position in the grid
            pos_y (int): The y-coordinate of the fish's position in the grid
            config (ConfigurationWator): Configuration object containing parameters for the fish
        """

        self.pos_x: int = pos_x
        self.pos_y: int = pos_y
        self.config = config
        self.is_alive: bool = True
        self.emoji: str = " "
```

Attributs

```python
def get_available_spaces(self, grid: list[list[Fish | None]]) -
    """
    Get a list of available (empty) neighboring spaces in the g
    Checks neighboring cells and collects coordinates of those

    Parameters:
        grid (list[list[Fish | None]]): The current state of th

    Returns:
        list[tuple[int, int]]: List of (x, y) coordinates of av
    """
    neighbors:  list[tuple[int, int]] = self.get_neighbors(grid
    available:  list[tuple[int, int]] = []

    for (x, y) in neighbors:
        if grid[y][x] is None :
            available.append((x, y))
    return available
```

Méthodes

Sarah Azzi, Alexandre Crestien, Hazel Cunuder

2025

Tuna

```python
class Tuna (Fish):
    def __init__(self, pos_x: int, pos_y: int, config: ConfigurationWator)-> None:
        super().__init__(pos_x, pos_y, config)
        self.reproduction_time: int = config.time_breed_tuna
        self.emoji: str ="🐟"

    def reproduce(self, pos_x: int, pos_y: int) -> Tuna | None:
        if self.reproduction_time <= 0:
            self.reproduction_time = self.config.time_breed_tuna
            return Tuna(pos_x, pos_y, self.config)
        self.reproduction_time -= 1
        return None
```

La classe Tuna représente l'espèce de base de la simulation.
Elle hérite de la logique générale de Fish (déplacement simple dans les cases vides) et ajoute un mécanisme de reproduction basé sur un timer.
Son rôle est d'assurer la croissance naturelle de l'écosystème et de servir de ressource aux prédateurs.

Sarah Azzi, Alexandre Crestien, Hazel Cunuder

2025

Shark

```python
def choose_move(self, available_moves: list[tuple[int, int]], grid: list[list[Fish | None]]) ->
    if not available_moves:
        return (self.pos_x, self.pos_y)

    if isinstance(grid[available_moves[0][1]][available_moves[0][0]], Tuna):
        x, y = random.choice(available_moves)
        tuna = grid[y][x]
        self.eat(tuna)
        return (x, y)

    return super().choose_move(available_moves, grid)

def eat(self, tuna: Tuna) -> None:
    tuna.is_alive = False
    self.energy += self.config.recovery_energy_shark
```

La classe Shark représente le prédateur intermédiaire : elle étend Fish avec une logique de chasse aux thons, une gestion d'énergie (gain en mangeant, perte sinon, mort par famine) et un timer de reproduction pour faire apparaître de nouveaux requins.

Sarah Azzi, Alexandre Crestien, Hazel Cunuder

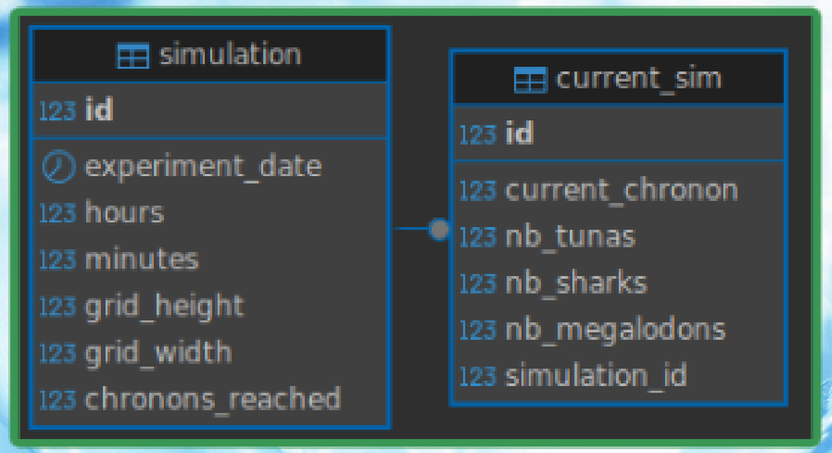
2025

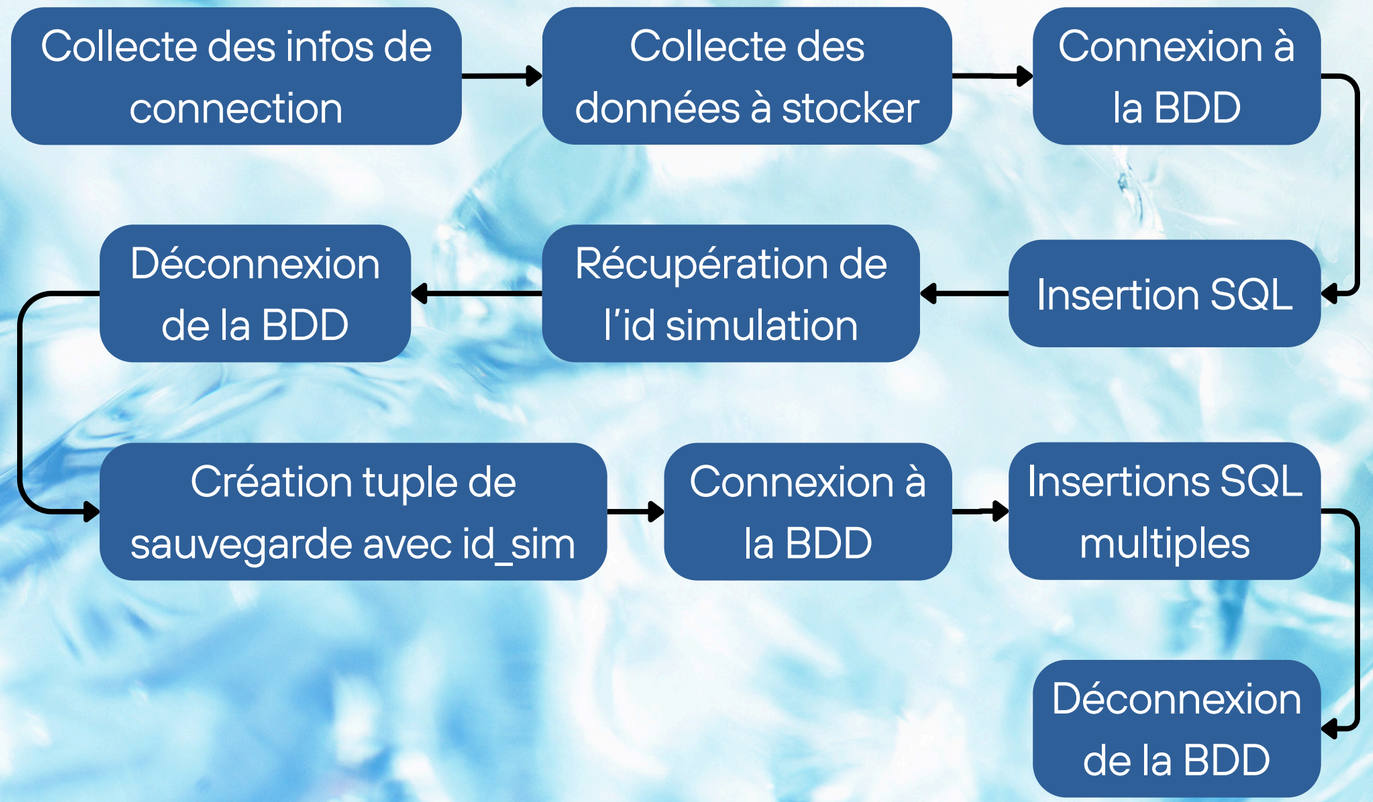
Class Megalodon

```
27      def get_available_spaces(self, grid: list[list[Fish | None]]) -> list[tuple[int,
41          neighbors = self.get_neighbors(grid)
42          preys = []
43          empty_box = []
44
45          for (x, y) in neighbors:
46              cell = grid[y][x]
47
48              if cell is None:
49                  empty_box.append((x, y))
50
51              elif isinstance(cell, (Tuna, Shark)) and cell.is_alive:
52                  preys.append((x, y))
```

Le Megalodon est l'ajout majeur de notre simulation. C'est le sommet de la chaîne alimentaire. Contrairement au requin qui ne mange que des thons, le Megalodon chasse toutes les autres espèces. Son code reflète cette dominance .

Sauvegarde

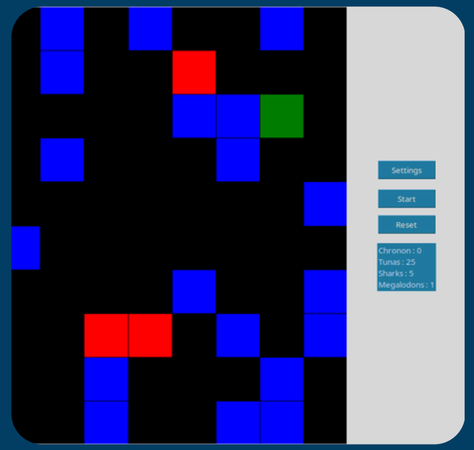

simulation
123 id
experiment_date
123 hours
123 minutes
123 grid_height
123 grid_width
123 chronons_reached
current_sim
123 id
123 current_chronon
123 nb_tunas
123 nb_sharks
123 nb_megalodons
123 simulation_id


Collecte des infos de connection
Collecte des données à stocker
Connexion à la BDD
Insertion SQL
Récupération de l'id simulation
Déconnexion de la BDD
Création tuple de sauvegarde avec id_sim
Connexion à la BDD
Insertions SQL multiples
Déconnexion de la BDD

Sarah Azzi, Alexandre Crestien, Hazel Cunuder

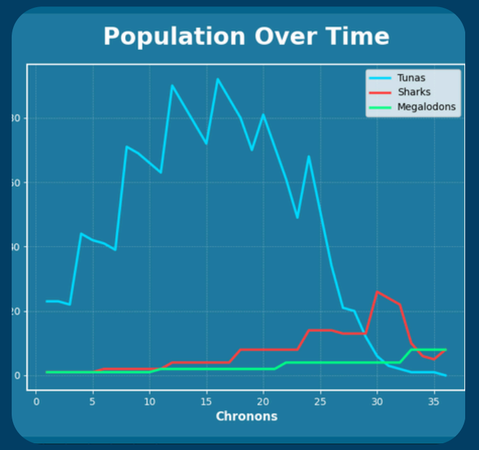
2025

Affichage



01. Simulation

02. Config



03. Graphique

Sarah Azzi, Alexandre Crestien, Hazel Cunuder

2025

Difficultées rencontrées

- Intégration et cohésion du groupe
- Bug d'affichage persistants avec nos sharks
- Dictature (Typage et Conventions)
- Découverte de Tkinter
- Quelques soucis avec Git

Leçons apprises

- Logique de test
- POO
- Apprentisage de git
- Travail en collaboration (PR)
- Gestion imprévus
- Travail sous pression

Sarah Azzi, Alexandre Crestien, Hazel Cunuder

2025

A vos questions

Merci pour votre attention !