

# Projet C++

Rapport de projet : Lefebvre Léo et Coudon Estelle

Sujet choisi : Distributeur de nourriture



## Introduction

Dans un monde où la possession d'animaux de compagnie ne se limite plus aux traditionnels chiens et chats, une problématique majeure émerge : comment automatiser le nourrissage de compagnons aux besoins diamétralement opposés ? Si nourrir un chat pendant un week-end est trivial, qu'en est-il de votre baleine domestique qui réclame sa dose de plancton à 4h du matin, ou de votre cheval de course qui a besoin de foin calibré au gramme près ?

Les distributeurs actuels du marché manquent cruellement de polyvalence. Notre projet "Bureau d'Études POO" vise à combler ce vide technologique en proposant une solution universelle. L'objectif était de concevoir un système complet (mécanique, électronique et logiciel) capable de gérer intelligemment la distribution de nourriture pour des espèces aussi variées qu'une souris, un chat, un chien, un cheval ou une baleine, en adaptant les rations selon les caractéristiques physiologiques et comportementales de l'animal.

## Conception Mécanique et Électronique

Le système repose sur une architecture matérielle robuste conçue sur mesure pour supporter les contraintes du projet.

Mécanique : Le stockage des denrées est assuré par un réservoir central. L'extraction de la nourriture est effectuée par une vis sans fin située à la base du réservoir, acheminant les croquettes vers la gamelle via un tube de sortie. Pour garantir un couple suffisant (notamment pour des aliments denses), la vis est entraînée par un moteur pas à pas, couplé à un système de pignons offrant un rapport de réduction de 1/3.

Électronique : L'intelligence du système réside sur une carte électronique PCB conçue spécifiquement pour ce projet :

- **Alimentation** : La carte est alimentée en 12V, tension nécessaire au pilotage du driver moteur A4988. Un convertisseur DC/DC abaisse ensuite cette tension à 5V pour alimenter la logique de commande.
- **Unité de calcul** : Le cœur du système est un microcontrôleur ESP32 (intégré à la carte), choisi pour sa puissance et sa connectivité WiFi native.
- **Interface et Capteurs** : Pour faciliter la maintenance, les périphériques sont connectés via des ports et connecteurs dédiés :
  - Un capteur laser VL53L0X mesure la distance restante dans le réservoir pour déduire le niveau de remplissage.

- Un module de 4 boutons tactiles TTP224 assure la navigation utilisateur.
- Un écran rond LCD GC9A01 offre une interface graphique moderne pour le suivi et la configuration.

## **Architecture Logicielle et Fonctionnement**

### Démarrage et Connectivité

À l'allumage, l'objet NetworkApp initialise une connexion WiFi pour interroger un serveur NTP. Cela permet au distributeur de récupérer l'heure exacte, nécessaire pour la gestion des heures de distributions. Simultanément, les données de l'animal et les horaires sont chargés depuis la mémoire non-volatile (ROM) de l'ESP32 via la librairie Preferences, assurant la persistance des données après extinction.

### Interface Utilisateur (UI)

L'interaction repose sur une machine à états gérée dans la boucle principale Application::Run().

- Dashboard : Affiche en temps réel l'heure, le nom de l'animal et une jauge circulaire indiquant le niveau de croquettes restant.
- Navigation : L'utilisateur navigue via les boutons (Haut/Bas/Entrée/Retour) à travers trois menus principaux : "Horaires", "Animal" et "Distribution Manuel".
- Saisie : Nous avons implémenté des claviers virtuels (numériques et alphabétiques) dessinés directement sur l'écran, permettant de modifier les champs comme le nom ou le poids.

### Logique Métier et Polymorphisme

Le cœur "POO" du projet réside dans la gestion des animaux. Une classe mère abstraite Species définit les attributs communs (régime alimentaire, espérance de vie) et les méthodes virtuelles (cri). Des classes filles (Dog, Cat, Mouse, Horse, Whale) héritent de cette structure en surchargeant les spécificités, comme le cri de l'animal (getSound()) ou le coefficient de ration.

La classe Animals compose l'animal "actif" en possédant un pointeur vers une instance de Species. Cela permet de changer dynamiquement l'espèce de l'animal configuré sans redémarrer le système.

### Gestion de la Distribution

La classe DistributionManager compare à chaque cycle l'heure actuelle avec les trois créneaux programmés. Elle calcule la dose à délivrer en fonction des paramètres de l'animal (Poids et Coefficient de l'espèce) et pilote le moteur via la classe Motor. Une sécurité logicielle, via la classe d'exception DistribException, empêche la distribution si le capteur détecte que le réservoir est vide.

## **Bilan et Conclusion**

### **Objectifs Atteints**

Nous avons réussi à produire un prototype fonctionnel répondant au cahier des charges amusant que nous nous étions fixés et l'architecture logicielle exploite les concepts de la POO :

- Encapsulation : Gestion stricte des accès aux composants (Moteur, Capteurs).
- Héritage et Polymorphisme : Intégration fluide de nouvelles espèces (la Baleine et le Cheval sont traités par le même code de distribution que le Chat).
- Persistance : Sauvegarde efficace des configurations utilisateur.

### **Difficultés Rencontrées**

Le développement a été parsemé de défis techniques instructifs :

- Boucle de distribution infinie : Initialement, la distribution se déclenchaît plusieurs fois durant la même minute car la condition d'heure restait valide pendant 60 secondes. Nous avons résolu cela en ajoutant un "flag" (lastFedMinute) qui ne se réinitialise que lorsque la minute change.
- Refactoring (Legacy Code) : Le projet a démarré sur la base d'un code procédural (maquette personnelle). La transition vers une architecture orientée objet stricte a demandé un effort conséquent de réécriture et de découpage en classes.
- Complexité de l'UI : La gestion graphique des claviers et des menus déroulants a été chronophage. Bien que fonctionnelle, l'implémentation actuelle manque de généricité (code dupliqué pour gérer les différents menus).
- Gestion des erreurs : Avant l'implémentation des exceptions personnalisées, le moteur actionnait la vis même si le réservoir était vide et le format des heures n'était pas vérifié. L'ajout de DistribException a permis de sécuriser le processus.

### **Améliorations Futures**

Par manque de temps, nous n'avons pas pu refaire totalement le code de l'interface graphique pour le rendre plus générique et utiliser le potentiel des caractéristiques de l'animal (influence du comportement sur la ration...). À l'avenir, nous envisageons de connecter le distributeur à une application mobile (via Wifi), permettant de configurer les horaires et l'identité de l'animal directement depuis un smartphone, rendant l'expérience utilisateur encore plus intuitive.

Ce projet nous a permis de concrétiser les concepts théoriques du C++ appliqués à un système embarqué réel, tout en gérant les contraintes matérielles liées à l'interaction hardware/software.



*Figure 1 : Distributeur en action*

# Annexes

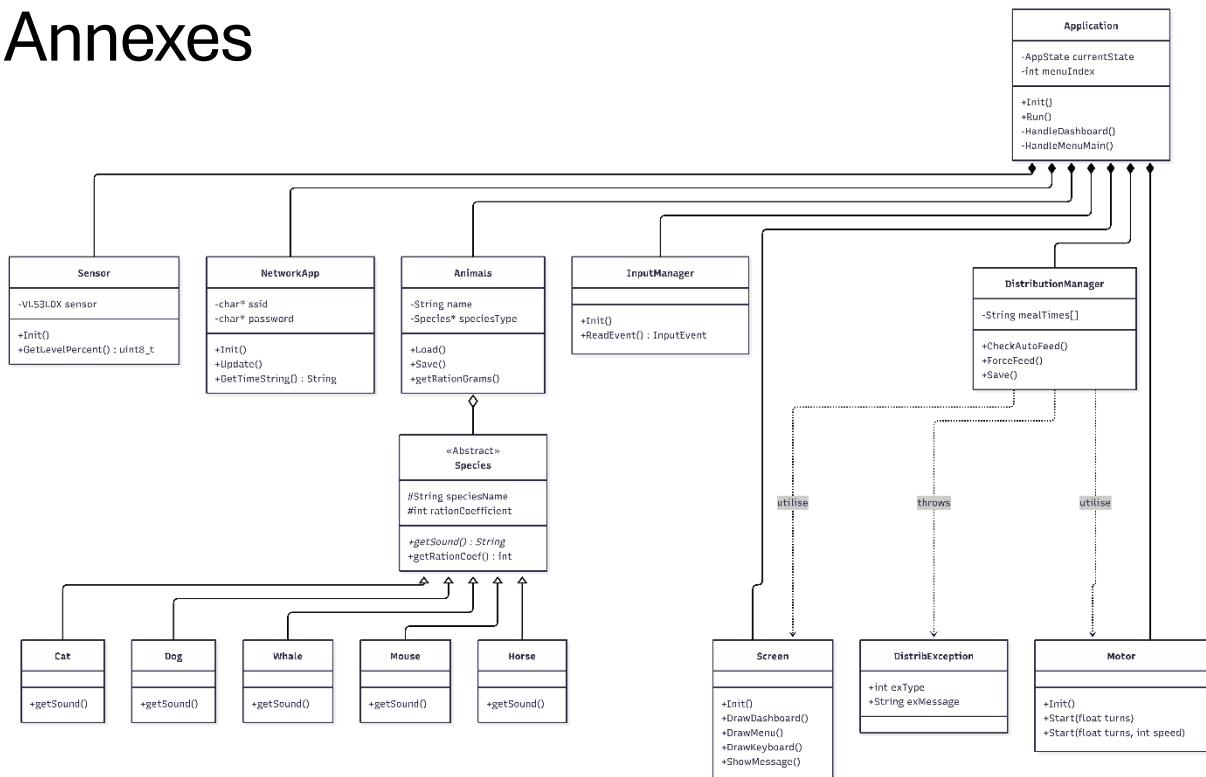


Figure 2 : Diagramme de classe

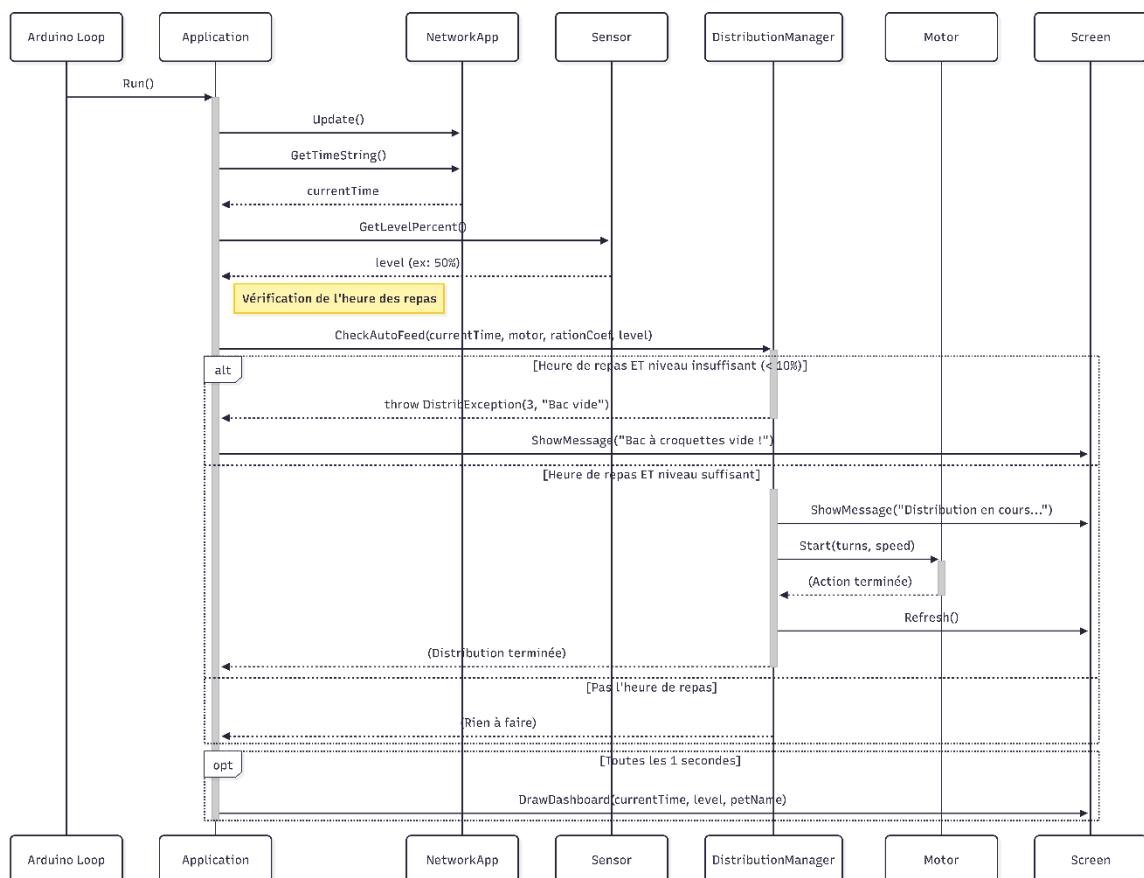


Figure 3 : Diagramme de séquence